Automated Reasoning I*

Uwe Waldmann

Winter Term 2019/2020

Topics of the Course

Preliminaries

abstract reduction systems
well-founded orderings

Propositional logic

syntax, semantics
calculi: CDCL-procedure, ...
implementation: 2-watched literals, clause learning

First-order predicate logic

syntax, semantics, model theory, ...
calculi: resolution, tableaux, ...
implementation: sharing, indexing

First-order predicate logic with equality

term rewriting systems
calculi: Knuth-Bendix completion, dependency pairs

Emphasis on:

logics and their properties,
proof systems for these logics and their properties:
soundness, completeness, complexity, implementation.

*This document contains the text of the lecture slides (almost verbatim) plus some additional information, mostly proofs of theorems that are presented on the blackboard during the course. It is not a full script and does not contain the examples and additional explanations given during the lecture. Moreover it should not be taken as an example how to write a research paper – neither stylistically nor typographically.

Parts of this document are based on lecture notes by Harald Ganzinger and Christoph Weidenbach.


1 Preliminaries

Before we start with the main subjects of the lecture, we repeat some prerequisites from mathematics and computer science and introduce some tools that we will need throughout the lecture.

1.1 Mathematical Prerequisites

\( \mathbb{N} = \{0, 1, 2, \ldots\} \) is the set of natural numbers (including 0).
\( \mathbb{Z}, \mathbb{Q}, \mathbb{R} \) denote the integers, rational numbers and the real numbers, respectively.

Relations

An \( n \)-ary relation \( R \) over some set \( M \) is a subset of \( M^n \): \( R \subseteq M^n \).

For two \( n \)-ary relations \( R, Q \) over some set \( M \), their union (\( \cup \)) or intersection (\( \cap \)) is again an \( n \)-ary relation, where

\[
R \cup Q := \{ (m_1, \ldots, m_n) \in M^n \mid (m_1, \ldots, m_n) \in R \text{ or } (m_1, \ldots, m_n) \in Q \}
\]

\[
R \cap Q := \{ (m_1, \ldots, m_n) \in M^n \mid (m_1, \ldots, m_n) \in R \text{ and } (m_1, \ldots, m_n) \in Q \}.
\]

A relation \( Q \) is a subrelation of a relation \( R \) if \( Q \subseteq R \).

We often use predicate notation for relations:

Instead of \( (m_1, \ldots, m_n) \in R \) we write \( R(m_1, \ldots, m_n) \), and say that \( R(m_1, \ldots, m_n) \) holds or is true.

For binary relations, we often use infix notation, so

\( (m, m') \in < \iff <(m, m') \iff m < m' \).

Words

Given a non-empty alphabet \( \Sigma \), the set \( \Sigma^* \) of finite words over \( \Sigma \) is defined inductively by

(i) the empty word \( \varepsilon \) is in \( \Sigma^* \),
(ii) if \( u \in \Sigma^* \) and \( a \in \Sigma \) then \( ua \) is in \( \Sigma^* \).

The set of non-empty finite words \( \Sigma^+ \) is \( \Sigma^* \setminus \{\varepsilon\} \).

The concatenation of two words \( u, v \in \Sigma^* \) is denoted by \( uv \).
The length $|u|$ of a word $u \in \Sigma^*$ is defined by

(i) $|\varepsilon| := 0$,

(ii) $|ua| := |u| + 1$ for any $u \in \Sigma^*$ and $a \in \Sigma$.

### 1.2 Abstract Reduction Systems


Throughout the lecture, we will have to work with reduction systems,

- on the object level, in particular in the section on equality,
- and on the meta level, i.e., to describe deduction calculi.

An abstract reduction system is a pair $(A, \rightarrow)$, where

- $A$ is a non-empty set,
- $\rightarrow \subseteq A \times A$ is a binary relation on $A$.

The relation $\rightarrow$ is usually written in infix notation, i.e., $a \rightarrow b$ instead of $(a, b) \in \rightarrow$.

Let $\rightarrow' \subseteq A \times B$ and $\rightarrow'' \subseteq B \times C$ be two binary relations. Then the composition of $\rightarrow'$ and $\rightarrow''$ is the binary relation $(\rightarrow' \circ \rightarrow'') \subseteq A \times C$ defined by

\[ a \ (\rightarrow' \circ \rightarrow'') \ c \quad \text{if and only if} \quad a \rightarrow' b \text{ and } b \rightarrow'' c \text{ for some } b \in B. \]

\[ \rightarrow^0 = \{(a, a) \mid a \in A\} \quad \text{identity} \]
\[ \rightarrow^{i+1} = \rightarrow^i \circ \rightarrow \quad \text{i + 1-fold composition} \]
\[ \rightarrow^+ = \bigcup_{i>0} \rightarrow^i \quad \text{transitive closure} \]
\[ \rightarrow^* = \bigcup_{i\geq0} \rightarrow^i = \rightarrow^+ \cup \rightarrow^0 \quad \text{reflexive transitive closure} \]
\[ \rightarrow\!\sim = \rightarrow^{-1} = \{(b, c) \mid c \rightarrow b\} \quad \text{inverse} \]
\[ \leftrightarrow = \rightarrow \cup \leftarrow \quad \text{symmetric closure} \]
\[ \leftrightarrow^+ = (\leftrightarrow)^+ \quad \text{transitive symmetric closure} \]
\[ \leftrightarrow^* = (\leftrightarrow)^* \quad \text{refl. trans. symmetric closure} \]

or equivalence closure

$b \in A$ is reducible, if there is a $c$ such that $b \rightarrow c$.

$b$ is in normal form (irreducible), if it is not reducible.

$c$ is a normal form of $b$, if $b \rightarrow^* c$ and $c$ is in normal form.

Notation: $c = b\downarrow$ (if the normal form of $b$ is unique).
A relation $\rightarrow$ is called

*terminating*, if there is no infinite descending chain $b_0 \rightarrow b_1 \rightarrow b_2 \rightarrow \ldots$.

*normalizing*, if every $b \in A$ has a normal form.

**Lemma 1.1** If $\rightarrow$ is terminating, then it is normalizing.

Note: The reverse implication does not hold.

### 1.3 Orderings

Important properties of binary relations:

Let $M \neq \emptyset$. A binary relation $R \subseteq M \times M$ is called

*reflexive*, if $R(x, x)$ for all $x \in M$,

*irreflexivity*, if $\neg R(x, x)$ for all $x \in M$,

*antisymmetric*, if $R(x, y)$ and $R(y, x)$ imply $x = y$ for all $x, y \in M$,

*transitive*, if $R(x, y)$ and $R(y, z)$ imply $R(x, z)$ for all $x, y, z \in M$,

*total*, if $R(x, y)$ or $R(y, x)$ or $x = y$ for all $x, y \in M$.

A strict partial ordering $\succ$ on a set $M \neq \emptyset$ is a transitive and irreflexive binary relation on $M$.

Notation:

$\prec$ for the inverse relation $\succ^{-1}$

$\succeq$ for the reflexive closure ($\succ \cup =$) of $\succ$

An $a \in M$ is called *minimal*, if there is no $b \in M$ with $a \succ b$.

An $a \in M$ is called *smallest*, if $b \succ a$ for all $b \in M \setminus \{a\}$.

Analogously:

An $a \in M$ is called *maximal*, if there is no $b \in M$ with $a \prec b$.

An $a \in M$ is called *largest*, if $b \prec a$ for all $b \in M \setminus \{a\}$.

Notation:

$M^{\prec x} = \{ y \in M \mid y \prec x \}$.

$M^{\preceq x} = \{ y \in M \mid y \preceq x \}$.

A subset $M' \subseteq M$ is called *downward closed*, if $x \in M'$ and $x \succ y$ implies $y \in M'$.
Well-Foundedness

Termination of reduction systems is strongly related to the concept of well-founded orderings.

A strict partial ordering $\succ$ on $M$ is called well-founded (or Noetherian), if there is no infinite descending chain $a_0 \succ a_1 \succ a_2 \succ \ldots$ with $a_i \in M$.

Well-Foundedness and Termination

**Lemma 1.2** If $\succ$ is a well-founded partial ordering and $\to \subseteq \succ$, then $\to$ is terminating.

**Lemma 1.3** If $\to$ is a terminating binary relation over $A$, then $\to^+$ is a well-founded partial ordering.

**Proof.** Transitivity of $\to^+$ is obvious; irreflexivity and well-foundedness follow from termination of $\to$. $\square$

Well-Founded Orderings: Examples

Natural numbers. $(\mathbb{N}, \succ)$

Lexicographic orderings. Let $(M_1, \succ_1), (M_2, \succ_2)$ be well-founded orderings. Then let their lexicographic combination

$\succ = (\succ_1, \succ_2)_{\text{lex}}$

on $M_1 \times M_2$ be defined as

$$(a_1, a_2) \succ (b_1, b_2) :\iff a_1 \succ_1 b_1 \text{ or } (a_1 = b_1 \text{ and } a_2 \succ_2 b_2)$$

(analogously for more than two orderings)

This again yields a well-founded ordering (proof below).

Length-based ordering on words. For alphabets $\Sigma$ with a well-founded ordering $\succ_\Sigma$, the relation $\succ$ defined as

$$w \succ w' :\iff |w| > |w'| \text{ or } (|w| = |w'| \text{ and } w \succ_\Sigma,\text{lex} w')$$

is a well-founded ordering on the set $\Sigma^*$ of finite words over the alphabet $\Sigma$ (Exercise).

Counterexamples:

$(\mathbb{Z}, \succ)$

$(\mathbb{N}, <)$

the lexicographic ordering on $\Sigma^*$
Basic Properties of Well-Founded Orderings

Lemma 1.4 \((M, \succ)\) is well-founded if and only if every non-empty \(M' \subseteq M\) has a minimal element.

Proof. (i) “\(\Leftarrow\)”: Suppose that \((M, \succ)\) is not well-founded. Then there is an infinite descending chain \(a_0 \succ a_1 \succ a_2 \succ \ldots\) with \(a_i \in M\). Consequently, the subset \(M' = \{a_i \mid i \in \mathbb{N}\}\) does not have a minimal element.

(ii) “\(\Rightarrow\)”: Suppose that the non-empty subset \(M' \subseteq M\) does not have a minimal element. Choose \(a_0 \in M'\) arbitrarily. Since for every \(a_i \in M'\) there is a smaller \(a_{i+1} \in M'\) (otherwise \(a_i\) would be minimal in \(M'\)), there is an infinite descending chain \(a_0 \succ a_1 \succ a_2 \succ \ldots\) \(\square\)

Lemma 1.5 \((M_1, \succ_1)\) and \((M_2, \succ_2)\) are well-founded if and only if \((M_1 \times M_2, \succ)\) with \(\succ = (\succ_1, \succ_2)_{lex}\) is well-founded.

Proof. (i) “\(\Rightarrow\)”: Suppose \((M_1 \times M_2, \succ)\) is not well-founded. Then there is an infinite sequence \((a_0, b_0) \succ (a_1, b_1) \succ (a_2, b_2) \succ \ldots\). Let \(A = \{a_i \mid i \geq 0\} \subseteq M_1\). Since \((M_1, \succ_1)\) is well-founded, \(A\) has a minimal element \(a_n\). But then \(B = \{b_i \mid i \geq n\} \subseteq M_2\) can not have a minimal element, contradicting the well-foundedness of \((M_2, \succ_2)\).

(ii) “\(\Leftarrow\)”: obvious. \(\square\)

Monotone Mappings

Let \((M_1, >_1)\) and \((M_2, >_2)\) be strict partial orderings. A mapping \(\varphi : M_1 \to M_2\) is called monotone, if \(a >_1 b\) implies \(\varphi(a) >_2 \varphi(b)\) for all \(a, b \in M_1\).

Lemma 1.6 If \(\varphi\) is a monotone mapping from \((M_1, >_1)\) to \((M_2, >_2)\) and \((M_2, >_2)\) is well-founded, then \((M_1, >_1)\) is well-founded.
Well-founded Induction

Theorem 1.7 (Well-founded (or Noetherian) Induction) Let \((M, \succ)\) be a well-founded ordering, let \(Q\) be a property of elements of \(M\).

If for all \(m \in M\) the implication

\[
\text{if } Q(m') \text{ for all } m' \in M \text{ such that } m \succ m',^1 \text{ then } Q(m).^2
\]

is satisfied, then the property \(Q(m)\) holds for all \(m \in M\).

Proof. Let \(X = \{ m \in M \mid Q(m) \text{ false} \}\). Suppose, \(X \neq \emptyset\). Since \((M, \succ)\) is well-founded, \(X\) has a minimal element \(m_1\). Hence for all \(m' \in M\) with \(m' \prec m_1\) the property \(Q(m')\) holds. On the other hand, the implication which is presupposed for this theorem holds in particular also for \(m_1\), hence \(Q(m_1)\) must be true so that \(m_1\) can not be in \(X\). Contradiction. \(\square\)

Well-founded Recursion

Let \(M\) and \(S\) be sets, let \(N \subseteq M\), and let \(f : M \to S\) be a function. Then the restriction of \(f\) to \(N\), denoted by \(f|_N\), is a function from \(N\) to \(S\) with \(f|_N(x) = f(x)\) for all \(x \in N\).

Theorem 1.8 (Well-founded (or Noetherian) Recursion) Let \((M, \succ)\) be a well-founded ordering, let \(S\) be a set. Let \(\phi\) be a binary function that takes two arguments \(x\) and \(g\) and maps them to an element of \(S\), where \(x \in M\) and \(g\) is a function from \(M^{\prec x}\) to \(S\).

Then there exists exactly one function \(f : M \to S\) such that for all \(x \in M\)

\[
f(x) = \phi(x, f|_{M^{\prec x}})
\]

Proof. The proof consists of four parts.

Part 1: For every downward closed subset \(N \subseteq M\) there is at most one function \(f : N \to S\) such that \(f(x) = \phi(x, f|_{M^{\prec x}})\).

Proof: Assume that there exist a downward closed subset \(N \subseteq M\) and two different functions \(f_1\) and \(f_2\) from \(N\) to \(S\) with this property. Therefore, the set \(N' := \{ x \in N \mid f_1(x) \neq f_2(x) \}\) is non-empty. By well-foundedness, \(N'\) has a minimal element \(y\). By minimality of \(y\), \(f_1|_{M^{\prec y}} = f_2|_{M^{\prec y}}\). Therefore \(f_1(y) = \phi(y, f_1|_{M^{\prec y}}) = \phi(y, f_2|_{M^{\prec y}}) = f_2(y)\), contradicting the assumption.

---

^1 induction hypothesis
^2 induction step
Part 2: If $N_1$ and $N_2$ are downward closed subsets of $M$ and the functions $f_1 : N_1 \to S$ and $f_2 : N_2 \to S$ satisfy $f_i(x) = \phi(x, f_i |_{M^{\prec_i}})$ for all $x \in N_i$ ($i = 1, 2$), then $f_1(x) = f_2(x)$ for all $x \in N_1 \cap N_2$.

Proof: Define $N_0 := N_1 \cap N_2$ and $f'_i = f_i |_{N_0}$ for $i = 1, 2$. Clearly $N_0$ is downward closed and for all $x \in N_0$ and $i = 1, 2$ we have $f'_i(x) = f_i(x) = \phi(x, f_i |_{M^{\prec_i}}) = \phi(x, f'_i |_{M^{\prec_i}})$. By part 1, there is at most one function from $N_0$ to $S$ with this property, so $f'_1 = f'_2$, and therefore $f_1(x) = f_2(x)$ for all $x \in N_1 \cap N_2$.

Part 3: For every $y \in M$ there exists a function $f_y : M^{\preceq y} \to S$ such that $f_y(x) = \phi(x, f_y |_{M^{\prec y}})$ for all $x \in M^{\preceq y}$.

Proof: We use well-founded induction over $\succ$. Let $y \in M$. By the induction hypothesis, for every $z \prec y$ there exists a function $f_z : M^{\preceq z} \to S$ such that $f(x) = \phi(x, f |_{M^{\prec}})$ for all $x \in M^{\preceq z}$. By part 2, all functions $f_x$ agree on the intersections of their domains. Define the function $f_y : M^{\preceq y} \to S$ by $f_y(x) = f_x(x)$ for $x \prec y$ and by $f_y(y) = \phi(y, f_y |_{M^{\prec y}})$.

Part 4: There exists a function $f : M \to S$ such that $f(x) = \phi(x, f |_{M^{\prec}})$ for all $x \in M$.

Proof: Define $f : M \to S$ by $f(x) = f_x(x)$.

The claim of the theorem follows now from part 1 (for $N := M$) and part 4.

The well-founded recursion scheme generalizes terminating recursive programs.

Note that functions defined by well-founded recursion need not be computable, in particular since for many well-founded orderings the sets $M^{\prec x}$ may be infinite.

### 1.4 Multisets

Let $M$ be a set. A multiset $S$ over $M$ is a mapping $S : M \to \mathbb{N}$. We interpret $S(m)$ as the number of occurrences of elements $m$ of the base set $M$ within the multiset $S$.

**Example.** $S = \{a, a, a, b, b\}$ is a multiset over $\{a, b, c\}$, where $S(a) = 3$, $S(b) = 2$, $S(c) = 0$.

We say that $m$ is an element of $S$, if $S(m) > 0$.

We use set notation ($\in, \subseteq, \cup, \cap$, etc.) with analogous meaning also for multisets, e.g.,

\[
\begin{align*}
m \in S \iff & S(m) > 0 \\
(S_1 \cup S_2)(m) := & S_1(m) + S_2(m) \\
(S_1 \cap S_2)(m) := & \min\{S_1(m), S_2(m)\} \\
(S_1 - S_2)(m) := & \begin{cases} S_1(m) - S_2(m) \text{ if } S_1(m) \geq S_2(m) \\ 0 \text{ otherwise} \end{cases} \\
S_1 \subseteq S_2 \iff & S_1(m) \leq S_2(m) \text{ for all } m \in M
\end{align*}
\]
A multiset $S$ is called finite, if
\[ |\{ m \in M \mid S(m) > 0 \}| < \infty. \]

From now on we only consider finite multisets.

**Multiset Orderings**

Let $(M, \succ)$ be an abstract reduction system. The multiset extension of $\succ$ to multisets over $M$ is defined by
\[ S_1 \succ_{\text{mul}} S_2 \text{ if and only if } \]

- there exist multisets $X$ and $Y$ over $M$ such that
  \[ \emptyset \neq X \subseteq S_1, \]
  \[ S_2 = (S_1 - X) \cup Y; \]
  \[ \forall y \in Y \exists x \in X: x \succ y \]

**Lemma 1.9 (König’s Lemma)** Every finitely branching tree with infinitely many nodes contains an infinite path.

**Theorem 1.10**

(a) If $\succ$ is transitive, then $\succ_{\text{mul}}$ is transitive.

(b) If $\succ$ is irreflexive and transitive, then $\succ_{\text{mul}}$ is irreflexive.

(c) If $\succ$ is a well-founded ordering, then $\succ_{\text{mul}}$ is a well-founded ordering.

(d) If $\succ$ is a strict total ordering, then $\succ_{\text{mul}}$ is a strict total ordering.

**Proof.** see Baader and Nipkow, page 22–24.

The multiset extension as defined above is due to Dershowitz and Manna (1979).

There are several other ways to characterize the multiset extension of a binary relation. The following one is due to Huet and Oppen (1980):

Let $(M, \succ)$ be an abstract reduction system. The (Huet/Oppen) multiset extension of $\succ$ to multisets over $M$ is defined by
\[ S_1 \succ_{\text{HO}}^{\text{mul}} S_2 \text{ if and only if } \]

- $S_1 \neq S_2$ and
- $\forall m \in M: (S_2(m) > S_1(m))$
- $\Rightarrow \exists m' \in M: m' > m$ and $S_1(m') > S_2(m')$
A third way to characterize the multiset extension of a binary relation $\succ$ is to define it as the transitive closure of the relation $\succ_{\text{mul}}^1$ given by

$$S_1 \succ_{\text{mul}}^1 S_2 \text{ if and only if there exists } x \in S_1 \text{ and a multiset } Y \text{ over } M \text{ such that }$$

$$S_2 = (S_1 - \{x\}) \cup Y,$$

$$\forall y \in Y: x \succ y$$

For strict partial orderings all three characterizations of $\succ_{\text{mul}}^1$ are equivalent:

**Theorem 1.11** If $\succ$ is a strict partial ordering, then

(a) $\succ_{\text{mul}} = \succ_{\text{mul}}^{\text{HO}},$

(b) $\succ_{\text{mul}} = (\succ_{\text{mul}}^1)^+.$

**Proof.** (a) see Baader and Nipkow, page 24–26. (b) Exercise. 

Note, however, that for an arbitrary binary relation $\succ$ all three relations $\succ_{\text{mul}}, \succ_{\text{mul}}^{\text{HO}},$ and $(\succ_{\text{mul}}^1)^+$ may be different.

### 1.5 Complexity Theory Prerequisites

A decision problem is a subset $L \subseteq \Sigma^*$ for some fixed finite alphabet $\Sigma.$

The function $\text{chr}(L, x)$ denotes the characteristic function for some decision problem $L$ and is defined by $\text{chr}(L, u) = 1$ if $u \in L$ and $\text{chr}(L, u) = 0$ otherwise.

**P and NP**

A decision problem is called solvable in polynomial time if its characteristic function can be computed in polynomial time. The class $P$ denotes all polynomial-time decision problems.

We say that a decision problem $L$ is in $NP$ if there is a predicate $Q(x, y)$ and a polynomial $p(n)$ such that for all $u \in \Sigma^*$ we have

(i) $u \in L$ if and only if there is a $v \in \Sigma^*$ with $|v| \leq p(|u|)$ and $Q(u, v)$ holds, and

(ii) the predicate $Q$ is in $P.$
Reducibility, NP-Hardness, NP-Completeness

A decision problem \( L \) is \textit{polynomial-time reducible} to a decision problem \( L' \) if there is a function \( g \) computable in polynomial time such that for all \( u \in \Sigma^* \) we have \( u \in L \) iff \( g(u) \in L' \).

For example, if \( L \) is polynomial-time reducible to \( L' \) and \( L' \in \text{P} \) then \( L \in \text{P} \).

A decision problem is \textit{NP-hard} if every problem in NP is polynomial-time reducible to it.

A decision problem is \textit{NP-complete} if it is NP-hard and in NP.
2 Propositional Logic

Propositional logic

- logic of truth values
- decidable (but NP-complete)
- can be used to describe functions over a finite domain
- industry standard for many analysis/verification tasks (e.g., model checking),

2.1 Syntax

- propositional variables
- logical connectives
  ⇒ Boolean combinations

Propositional Variables

Let Π be a set of propositional variables.

We use letters $P$, $Q$, $R$, $S$, to denote propositional variables.

Propositional Formulas

$F_\Pi$ is the set of propositional formulas over Π defined inductively as follows:

$$F, G ::= \bot \quad \text{(falsum)}$$
$$\top \quad \text{(verum)}$$
$$P, \quad P \in \Pi \quad \text{(atomic formula)}$$
$$\neg F \quad \text{(negation)}$$
$$(F \land G) \quad \text{(conjunction)}$$
$$(F \lor G) \quad \text{(disjunction)}$$
$$(F \to G) \quad \text{(implication)}$$
$$(F \leftrightarrow G) \quad \text{(equivalence)}$$
Notational Conventions

As a notational convention we assume that $\neg$ binds strongest, and we remove outermost parentheses, so $\neg P \lor Q$ is actually a shorthand for $((\neg P) \lor Q)$.

Instead of $((P \land Q) \land R)$ we simply write $P \land Q \land R$ (and analogously for $\lor$).

For all other logical connectives we will use parentheses when needed.

Formula Manipulation

Automated reasoning is very much formula manipulation. In order to precisely represent the manipulation of a formula, we introduce positions.

A position is a word over $\mathbb{N}$. The set of positions of a formula $F$ is inductively defined by

\[
\text{pos}(F) := \{\varepsilon\} \text{ if } F \in \{\top, \bot\} \text{ or } F \in \Pi \\
\text{pos}(\neg F) := \{\varepsilon\} \cup \{1p \mid p \in \text{pos}(F)\} \\
\text{pos}(F \circ G) := \{\varepsilon\} \cup \{1p \mid p \in \text{pos}(F)\} \cup \{2p \mid p \in \text{pos}(G)\} \\
\text{where } \circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}.
\]

The prefix order $\leq$ on positions is defined by $p \leq q$ if there is some $p'$ such that $pp' = q$.

Note that the prefix order is partial, e.g., the positions $12$ and $21$ are not comparable, they are “parallel”, see below.

By $<$ we denote the strict part of $\leq$, that is, $p < q$ if $p \leq q$ but not $q \leq p$.

By $\parallel$ we denote incomparable positions, that is, $p \parallel q$ if neither $p \leq q$ nor $q \leq p$.

We say that $p$ is above $q$ if $p \leq q$, $p$ is strictly above $q$ if $p < q$, and $p$ and $q$ are parallel if $p \parallel q$.

The size of a formula $F$ is given by the cardinality of pos($F$): $|F| := |\text{pos}(F)|$.

The subformula of $F$ at position $p \in \text{pos}(F)$ is recursively defined by

\[
F|_\varepsilon := F \\
(\neg F)|_p := F|_p \\
(F_1 \circ F_2)|_p := F_i|_p \text{ where } i \in \{1, 2\} \text{ and } \circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}.
\]
Finally, the replacement of a subformula at position \( p \in \text{pos}(F) \) by a formula \( G \) is recursively defined by

\[
F[G]_p := G \\
(\neg F)[G]_p := \neg(F[G]_p) \\
(F_1 \circ F_2)[G]_p := (F_1[G]_p \circ F_2) \\
(F_1 \circ F_2)[G]_p := (F_1 \circ F_2[G]_p)
\]

where \( \circ \in \{\land, \lor, \rightarrow, \leftrightarrow\} \).

**Example 2.1** The set of positions for the formula \( F = (P \rightarrow Q) \rightarrow (P \land \neg Q) \) is \( \text{pos}(F) = \{\varepsilon, 1, 11, 12, 2, 21, 22, 221\} \).

The subformula at position 22 is \( F|_{22} = \neg Q \) and replacing this formula by \( P \leftrightarrow Q \) results in \( F|_{P \leftrightarrow Q}_{22} = (P \rightarrow Q) \rightarrow (P \land (P \leftrightarrow Q)) \).

**Polarities**

A further prerequisite for efficient formula manipulation is the polarity of a subformula \( G \) of \( F \). The polarity determines the number of “negations” starting from \( F \) down to \( G \). It is 1 for an even number, \(-1\) for an odd number and 0 if there is at least one equivalence connective along the path.

The *polarity* of a subformula \( G = F|_p \) at position \( p \) is \( \text{pol}(F, p) \), where \( \text{pol} \) is recursively defined by

\[
\text{pol}(F, \varepsilon) := 1 \\
\text{pol}(\neg F, 1p) := -\text{pol}(F, p) \\
\text{pol}(F_1 \circ F_2, ip) := \text{pol}(F_1, p) \text{ if } \circ \in \{\land, \lor\} \\
\text{pol}(F_1 \rightarrow F_2, 1p) := -\text{pol}(F_1, p) \\
\text{pol}(F_1 \rightarrow F_2, 2p) := \text{pol}(F_2, p) \\
\text{pol}(F_1 \leftrightarrow F_2, ip) := 0
\]

**Example 2.2** Let \( F = (P \rightarrow Q) \rightarrow (P \land \neg Q) \). Then \( \text{pol}(F, 1) = \text{pol}(F, 12) = \text{pol}(F, 221) = -1 \) and \( \text{pol}(F, \varepsilon) = \text{pol}(F, 11) = \text{pol}(F, 2) = \text{pol}(F, 21) = \text{pol}(F, 22) = 1 \).

For the formula \( F' = (P \land Q) \leftrightarrow (P \lor Q) \) we get \( \text{pol}(F', \varepsilon) = 1 \) and \( \text{pol}(F', p) = 0 \) for all \( p \in \text{pos}(F') \) different from \( \varepsilon \).
2.2 Semantics

In classical logic (dating back to Aristotle) there are “only” two truth values “true” and “false” which we shall denote, respectively, by 1 and 0.

There are multi-valued logics having more than two truth values.

Valuations

A propositional variable has no intrinsic meaning. The meaning of a propositional variable has to be defined by a valuation.

A Π-valuation is a map

\[ A : \Pi \rightarrow \{0, 1\}. \]

where \( \{0, 1\} \) is the set of truth values.

Truth Value of a Formula in \( A \)

Given a Π-valuation \( A \), its extension to formulas \( A^* : F_\Pi \rightarrow \{0, 1\} \) is defined inductively as follows:

\[
\begin{align*}
A^*(\bot) &= 0 \\
A^*(\top) &= 1 \\
A^*(P) &= A(P) \\
A^*(\neg F) &= 1 - A^*(F) \\
A^*(F \land G) &= \min(A^*(F), A^*(G)) \\
A^*(F \lor G) &= \max(A^*(F), A^*(G)) \\
A^*(F \to G) &= \max(1 - A^*(F), A^*(G)) \\
A^*(F \leftrightarrow G) &= \text{if } A^*(F) = A^*(G) \text{ then } 1 \text{ else } 0
\end{align*}
\]

For simplicity, the extension \( A^* \) of \( A \) is usually also denoted by \( A \).
2.3 Models, Validity, and Satisfiability

Let $F$ be a Π-formula.

We say that $F$ is true under $A$ ($A$ is a model of $F$; $F$ is valid in $A$; $F$ holds under $A$), written $A \models F$, if $A(F) = 1$.

We say that $F$ is valid or that $F$ is a tautology, written $\models F$, if $A \models F$ for all Π-valuations $A$.

$F$ is called satisfiable if there exists an $A$ such that $A \models F$. Otherwise $F$ is called unsatisfiable (or contradictory).

Entailment and Equivalence

$F$ entails (implies) $G$ (or $G$ is a consequence of $F$), written $F \models G$, if for all Π-valuations $A$ we have

\[
\text{if } A \models F \text{ then } A \models G,
\]

or equivalently

\[A(F) \leq A(G).\]

$F$ and $G$ are called equivalent, written $F \equiv G$, if for all Π-valuations $A$ we have

$A \models F$ if and only if $A \models G$,

or equivalently

$A(F) = A(G)$.

$F$ and $G$ are called equisatisfiable, if either both $F$ and $G$ are satisfiable, or both $F$ and $G$ are unsatisfiable.

Proposition 2.3 $F \models G$ if and only if $\models (F \rightarrow G)$.

Proof. ($\Rightarrow$) Suppose that $F$ entails $G$. Let $A$ be an arbitrary Π-valuation. We have to show that $A \models F \rightarrow G$. If $A(F) = 1$, then $A(G) = 1$ (since $F \models G$), and hence $A(F \rightarrow G) = \max(1 - 1, 1) = 1$. Otherwise $A(F) = 0$, then $A(F \rightarrow G) = \max(1 - 0, A(G)) = 1$ independently of $A(G)$. In both cases, $A \models F \rightarrow G$.

($\Leftarrow$) Suppose that $F$ does not entail $G$. Then there exists a Π-valuation $A$ such that $A \models F$, but not $A \models G$. Consequently, $A(F \rightarrow G) = \max(1 - A(F), A(G)) = \max(1 - 1, 0) = 0$, so $(F \rightarrow G)$ does not hold under $A$. \[\square\]
Proposition 2.4 \( F \models G \) if and only if \( \models (F \leftrightarrow G) \).

**Proof.** Analogously to Prop. 2.3. \( \square \)

Entailment is extended to sets of formulas \( N \) in the “natural way”:

\[ N \models F \] if for all \( \Pi \)-valuations \( \mathcal{A} \):

if \( \mathcal{A} \models G \) for all \( G \in N \), then \( \mathcal{A} \models F \).

Note: Formulas are always finite objects; but sets of formulas may be infinite. Therefore, it is in general not possible to replace a set of formulas by the conjunction of its elements.

**Validity vs. Unsatisfiability**

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 2.5 \( F \) is valid if and only if \( \neg F \) is unsatisfiable.

**Proof.** \((\Rightarrow)\) If \( F \) is valid, then \( \mathcal{A}(F) = 1 \) for every valuation \( \mathcal{A} \). Hence \( \mathcal{A}(\neg F) = 1 - \mathcal{A}(F) = 0 \) for every valuation \( \mathcal{A} \), so \( \neg F \) is unsatisfiable.

\((\Leftarrow)\) Analogously. \( \square \)

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

In a similar way, entailment can be reduced to unsatisfiability and vice versa:

Proposition 2.6 \( N \models F \) if and only if \( N \cup \{\neg F\} \) is unsatisfiable.

Proposition 2.7 \( N \models \bot \) if and only if \( N \) is unsatisfiable.

**Checking Unsatisfiability**

Every formula \( F \) contains only finitely many propositional variables. Obviously, \( \mathcal{A}(F) \) depends only on the values of those finitely many variables in \( F \) under \( \mathcal{A} \).

If \( F \) contains \( n \) distinct propositional variables, then it is sufficient to check \( 2^n \) valuations to see whether \( F \) is satisfiable or not \( \Rightarrow \) truth table.

So the satisfiability problem is clearly decidable (but, by Cook’s Theorem, NP-complete).

Nevertheless, in practice, there are (much) better methods than truth tables to check the satisfiability of a formula. (later more)
**Replacement Theorem**

**Proposition 2.8** Let $\mathcal{A}$ be a valuation, let $F$ and $G$ be formulas, and let $H = H[F]_p$ be a formula in which $F$ occurs as a subformula at position $p$.

If $\mathcal{A}(F) = \mathcal{A}(G)$, then $\mathcal{A}(H[F]_p) = \mathcal{A}(H[G]_p)$.

**Proof.** The proof proceeds by induction over the length of $p$.

If $p = \varepsilon$, then $H[F]_\varepsilon = F$ and $H[G]_\varepsilon = G$, so $\mathcal{A}(H[F]_p) = \mathcal{A}(F) = \mathcal{A}(G) = \mathcal{A}(H[G]_p)$ by assumption.

If $p = 1q$ or $p = 2q$, then $H = \neg H_1$ or $H = H_1 \circ H_2$ for $\circ \in \{\land, \lor, \to, \leftrightarrow\}$. Assume that $p = 1q$ and that $H = H_1 \land H_2$, hence $H[F]_p = H[F]_{1q} = H_1[F]_q \land H_2$. By the induction hypothesis, $\mathcal{A}(H_1[F]_q) = \mathcal{A}(H_1[G]_q)$. Hence $\mathcal{A}(H[F]_{1q}) = \mathcal{A}(H_1[F]_q \land H_2) = \min(\mathcal{A}(H_1[F]_q), \mathcal{A}(H_2)) = \min(\mathcal{A}(H_1[G]_q), \mathcal{A}(H_2)) = \mathcal{A}(H_1[G]_q \land H_2) = \mathcal{A}(H[G]_{1q})$.

The case $p = 2q$ and the other boolean connectives are handled analogously. $\square$

**Theorem 2.9** Let $F$ and $G$ be equivalent formulas, let $H = H[F]_p$ be a formula in which $F$ occurs as a subformula at position $p$.

Then $H[F]_p$ is equivalent to $H[G]_p$.

**Proof.** We have to show that $\mathcal{A}(H[F]_p) = \mathcal{A}(H[G]_p)$ for every $\Pi$-valuation $\mathcal{A}$.

Choose $\mathcal{A}$ arbitrarily. Since $F$ and $G$ are equivalent, we know that $\mathcal{A}(F) = \mathcal{A}(G)$.

Hence, by the previous proposition, $\mathcal{A}(H[F]_p) = \mathcal{A}(H[G]_p)$. $\square$
Some Important Equivalences

**Proposition 2.10** The following equivalences hold for all formulas $F, G, H$:

- $(F \land F) \models F$ (Idempotency)
- $(F \lor F) \models F$
- $(F \land G) \models (G \land F)$ (Commutativity)
- $(F \lor G) \models (G \lor F)$ (Associativity)
- $(F \land (G \land H)) \models ((F \land G) \land H)$ (Distributivity)
- $(F \lor (G \land H)) \models ((F \lor G) \lor (F \land H))$
- $(F \land (F \lor G)) \models F$ (Absorption)
- $(F \lor (F \land G)) \models F$ (Absorption)
- $(\neg \neg F) \models F$ (Double Negation)
- $\neg (F \land G) \models (\neg F \lor \neg G)$ (De Morgan’s Laws)
- $\neg (F \lor G) \models (\neg F \land \neg G)$
- $(F \land G) \models F$, if $G$ is a tautology
- $(F \lor G) \models \top$, if $G$ is a tautology
- $(F \land G) \models \bot$, if $G$ is unsatisfiable
- $(F \lor G) \models F$, if $G$ is unsatisfiable (Tautology Laws)
- $(F \leftrightarrow G) \models ((F \rightarrow G) \land (G \rightarrow F))$
- $(F \leftrightarrow G) \models ((F \land G) \lor (\neg F \land \neg G))$ (Equivalence)
- $(F \rightarrow G) \models (\neg F \lor G)$ (Implication)

An Important Entailment

**Proposition 2.11** The following entailment holds for all formulas $F, G, H$:

$$(F \lor H) \land (G \lor \neg H) \models F \lor G$$ (Generalized Resolution)
2.4 Normal Forms

We define *conjunctions* of formulas as follows:

\[ \bigwedge_{i=1}^{0} F_i = \top. \]
\[ \bigwedge_{i=1}^{1} F_i = F_1. \]
\[ \bigwedge_{i=1}^{n+1} F_i = \bigwedge_{i=1}^{n} F_i \land F_{n+1}. \]

and analogously *disjunctions*:

\[ \bigvee_{i=1}^{0} F_i = \bot. \]
\[ \bigvee_{i=1}^{1} F_i = F_1. \]
\[ \bigvee_{i=1}^{n+1} F_i = \bigvee_{i=1}^{n} F_i \lor F_{n+1}. \]

**Literals and Clauses**

A *literal* is either a propositional variable \( P \) or a negated propositional variable \( \neg P \).

A *clause* is a (possibly empty) disjunction of literals.

**CNF and DNF**

A formula is in *conjunctive normal form* (CNF, clause normal form), if it is a conjunction of disjunctions of literals (or in other words, a conjunction of clauses).

A formula is in *disjunctive normal form* (DNF), if it is a disjunction of conjunctions of literals.

Warning: definitions in the literature differ:

- are complementary literals permitted?
- are duplicated literals permitted?
- are empty disjunctions/conjunctions permitted?

Checking the validity of CNF formulas or the unsatisfiability of DNF formulas is easy:

A formula in CNF is valid, if and only if each of its disjunctions contains a pair of complementary literals \( P \) and \( \neg P \).

Conversely, a formula in DNF is unsatisfiable, if and only if each of its conjunctions contains a pair of complementary literals \( P \) and \( \neg P \).

On the other hand, checking the unsatisfiability of CNF formulas or the validity of DNF formulas is known to be coNP-complete.
Conversion to CNF/DNF

**Proposition 2.12** For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).

**Proof.** We describe a (naive) algorithm to convert a formula to CNF.

Apply the following rules as long as possible (modulo commutativity of $\land$ and $\lor$):

Step 1: Eliminate equivalences:

$$ H[F \leftrightarrow G]_p \Rightarrow_{\text{CNF}} H[(F \rightarrow G) \land (G \rightarrow F)]_p $$

Step 2: Eliminate implications:

$$ H[F \rightarrow G]_p \Rightarrow_{\text{CNF}} H[\neg F \lor G]_p $$

Step 3: Push negations downward:

$$ H[\neg(F \lor G)]_p \Rightarrow_{\text{CNF}} H[\neg F \land \neg G]_p $$

$$ H[\neg(F \land G)]_p \Rightarrow_{\text{CNF}} H[\neg F \lor \neg G]_p $$

Step 4: Eliminate multiple negations:

$$ H[\neg\neg F]_p \Rightarrow_{\text{CNF}} H[F]_p $$

Step 5: Push disjunctions downward:

$$ H[(F \land F') \lor G]_p \Rightarrow_{\text{CNF}} H[(F \lor G) \land (F' \lor G)]_p $$

Step 6: Eliminate $\top$ and $\bot$:

$$ H[F \land \top]_p \Rightarrow_{\text{CNF}} H[F]_p $$

$$ H[F \land \bot]_p \Rightarrow_{\text{CNF}} H[\bot]_p $$

$$ H[F \lor \top]_p \Rightarrow_{\text{CNF}} H[\top]_p $$

$$ H[F \lor \bot]_p \Rightarrow_{\text{CNF}} H[F]_p $$

$$ H[\neg \bot]_p \Rightarrow_{\text{CNF}} H[\top]_p $$

$$ H[\neg \top]_p \Rightarrow_{\text{CNF}} H[\bot]_p $$
Proving termination is easy for steps 2, 4, and 6; steps 1, 3, and 5 are a bit more complicated.

For step 1, we can prove termination in the following way: We define a function $\mu_1$ from formulas to positive integers such that $\mu_1(\bot) = \mu_1(\top) = \mu_1(P) = 1$, $\mu_1(\neg F) = \mu_1(F)$, $\mu_1(F \land G) = \mu_1(F \lor G) = \mu_1(F \to G) = \mu_1(F) + \mu_1(G)$, and $\mu_1(F \leftrightarrow G) = 2\mu_1(F) + 2\mu_1(G) + 1$. Observe that $\mu_1$ is constructed in such a way that $\mu_1(F) > \mu_1(G)$ implies $\mu_1(H[F]) > \mu_1(H[G])$ for all formulas $F$, $G$, and $H$. Furthermore, $\mu_1$ has the property that swapping the arguments of some $\land$ or $\lor$ in a formula $F$ does not change the value of $\mu_1(F)$. (This is important since the transformation rules can be applied modulo commutativity of $\land$ and $\lor$.) Using these properties, we can show that whenever a formula $H'$ is the result of applying the rule of step 1 to a formula $H$, then $\mu_1(H) > \mu_1(H')$. Since $\mu_1$ takes only positive integer values, step 1 must terminate.

Termination of steps 3 and 5 is proved similarly. For step 3, we use function $\mu_2$ from formulas to positive integers such that $\mu_2(\bot) = \mu_2(\top) = \mu_2(P) = 1$, $\mu_2(\neg F) = 2\mu_2(F)$, $\mu_2(F \land G) = \mu_2(F \lor G) = \mu_2(F \to G) = \mu_2(F \leftrightarrow G) = \mu_2(F) + \mu_2(G) + 1$. Whenever a formula $H'$ is the result of applying a rule of step 3 to a formula $H$, then $\mu_2(H) > \mu_2(H')$. Since $\mu_2$ takes only positive integer values, step 3 must terminate.

For step 5, we use a function $\mu_3$ from formulas to positive integers such that $\mu_3(\bot) = \mu_3(\top) = \mu_3(P) = 1$, $\mu_3(\neg F) = \mu_3(F) + 1$, $\mu_3(F \land G) = \mu_3(F \to G) = \mu_3(F \leftrightarrow G) = \mu_3(F) + \mu_3(G) + 1$, and $\mu_3(F \lor G) = 2\mu_3(F)\mu_3(G)$. Again, if a formula $H'$ is the result of applying a rule of step 5 to a formula $H$, then $\mu_3(H) > \mu_3(H')$. Since $\mu_3$ takes only positive integer values, step 5 terminates, too.

The resulting formula is equivalent to the original one and in CNF.

The conversion of a formula to DNF works in the same way, except that conjunctions have to be pushed downward in step 5.

\[\square\]

**Negation Normal Form (NNF)**

The formula after application of Step 4 is said to be in Negation Normal Form, i.e., it contains neither $\to$ nor $\leftrightarrow$ and negation symbols only occur in front of propositional variables (atoms).

**Complexity**

Conversion to CNF (or DNF) may produce a formula whose size is exponential in the size of the original one.
2.5 Improving the CNF Transformation

The goal

“Given a formula $F$, find an equivalent formula $G$ in CNF”

is unpractical.

But if we relax the requirement to

“Given a formula $F$, find an equisatisfiable formula $G$ in CNF”

we can get an efficient transformation.

Tseitin Transformation

**Proposition 2.13** A formula $H[F]_p$ is satisfiable if and only if $H[Q]_p \land (Q \leftrightarrow F)$ is satisfiable, where $Q$ is a new propositional variable that works as an abbreviation for $F$.

Satisfiability-preserving CNF transformation (Tseitin 1970):

Use the rule above recursively for all subformulas in the original formula (this introduces a linear number of new propositional variables $Q$ and definitions $Q \leftrightarrow F$).

Convert of the resulting conjunction to CNF (this increases the size only by an additional factor, since each formula $Q \leftrightarrow F$ yields at most four clauses in the CNF).

Polarity-based CNF Transformation

A further improvement is possible by taking the polarity of the subformula $F$ into account.

**Proposition 2.14** Let $A$ be a valuation, let $F$ and $G$ be formulas, and let $H = H[F]_p$ be a formula in which $F$ occurs as a subformula at position $p$.

If $\text{pol}(H, p) = 1$ and $A(F) \leq A(G)$, then $A(H[F]_p) \leq A(H[G]_p)$.

If $\text{pol}(H, p) = -1$ and $A(F) \geq A(G)$, then $A(H[F]_p) \leq A(H[G]_p)$.

**Proof.** Exercise. □
Let $Q$ be a propositional variable not occurring in $H[F]_p$.

Define the formula $\text{def}(H, p, Q, F)$ by

- $(Q \rightarrow F)$, if $\text{pol}(H, p) = 1$,
- $(F \rightarrow Q)$, if $\text{pol}(H, p) = -1$,
- $(Q \leftrightarrow F)$, if $\text{pol}(H, p) = 0$.

**Proposition 2.15** Let $Q$ be a propositional variable not occurring in $H[F]_p$. Then $H[F]_p$ is satisfiable if and only if $H[Q]_p \land \text{def}(H, p, Q, F)$ is satisfiable.

**Proof.** $(\Rightarrow)$ Since $H[F]_p$ is satisfiable, there exists a $\Pi$-valuation $\mathcal{A}$ such that $\mathcal{A} \models H[F]_p$. Let $\Pi' = \Pi \cup \{Q\}$ and define the $\Pi'$-valuation $\mathcal{A}'$ by $\mathcal{A}'(P) = \mathcal{A}(P)$ for $P \in \Pi$ and $\mathcal{A}'(Q) = \mathcal{A}(F)$. Obviously $\mathcal{A}'(\text{def}(H, p, Q, F)) = 1$; moreover $\mathcal{A}'(H[Q]_p) = \mathcal{A}'(H[F]_p) = \mathcal{A}(H[F]_p) = 1$ by Prop. 2.8, so $H[Q]_p \land \text{def}(H, p, Q, F)$ is satisfiable.

$(\Leftarrow)$ Let $\mathcal{A}$ be a valuation such that $\mathcal{A} \models H[Q]_p \land \text{def}(H, p, Q, F)$. So $\mathcal{A}(H[Q]_p) = 1$ and $\mathcal{A}(\text{def}(H, p, Q, F)) = 1$. We will show that $\mathcal{A} \models H[F]_p$.

If $\text{pol}(H, p) = 0$, then $\text{def}(H, p, Q, F) = (Q \leftrightarrow F)$, so $\mathcal{A}(Q) = \mathcal{A}(F)$, hence $\mathcal{A}(H[F]_p) = \mathcal{A}(H[Q]_p) = 1$ by Prop. 2.8.

If $\text{pol}(H, p) = 1$, then $\text{def}(H, p, Q, F) = (Q \rightarrow F)$, so $\mathcal{A}(Q) \leq \mathcal{A}(F)$. By Prop. 2.14, $\mathcal{A}(H[F]_p) \geq \mathcal{A}(H[Q]_p) = 1$, so $\mathcal{A}(H[F]_p) = 1$.

If $\text{pol}(H, p) = -1$, then $\text{def}(H, p, Q, F) = (F \rightarrow Q)$, so $\mathcal{A}(F) \leq \mathcal{A}(Q)$. By Prop. 2.14, $\mathcal{A}(H[F]_p) \geq \mathcal{A}(H[Q]_p) = 1$, so $\mathcal{A}(H[F]_p) = 1$. $\square$

**Optimized CNF**

Not every introduction of a definition for a subformula leads to a smaller CNF.

The number of potentially generated clauses is a good indicator for useful CNF transformations.
The functions $\nu(F)$ and $\bar{\nu}(F)$ give us upper bounds for the number of clauses in $\text{cnf}(F)$ and $\text{cnf}(\neg F)$ using a naive CNF transformation.

<table>
<thead>
<tr>
<th>$G$</th>
<th>$\nu(G)$</th>
<th>$\bar{\nu}(G)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P, \top, \bot$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>$F_1 \land F_2$</td>
<td>$\nu(F_1) + \nu(F_2)$</td>
<td>$\bar{\nu}(F_1) \bar{\nu}(F_2)$</td>
</tr>
<tr>
<td>$F_1 \lor F_2$</td>
<td>$\nu(F_1) \nu(F_2)$</td>
<td>$\bar{\nu}(F_1) + \bar{\nu}(F_2)$</td>
</tr>
<tr>
<td>$\neg F_1$</td>
<td>$\bar{\nu}(F_1)$</td>
<td>$\nu(F_1)$</td>
</tr>
<tr>
<td>$F_1 \leftrightarrow F_2$</td>
<td>$\nu(F_1) \bar{\nu}(F_2) + \bar{\nu}(F_1) \nu(F_2)$</td>
<td>$\nu(F_1) \nu(F_2) + \bar{\nu}(F_1) \bar{\nu}(F_2)$</td>
</tr>
</tbody>
</table>

A better CNF transformation:

Step 1: Exhaustively apply modulo commutativity of $\leftrightarrow$ and associativity/commutativity of $\land$, $\lor$:

\begin{align*}
H[(F \land \top)]_p & \Rightarrow \text{OCNF} \quad H[F]_p \\
H[(F \lor \bot)]_p & \Rightarrow \text{OCNF} \quad H[F]_p \\
H[(F \leftrightarrow \bot)]_p & \Rightarrow \text{OCNF} \quad H[\neg F]_p \\
H[(F \leftrightarrow \top)]_p & \Rightarrow \text{OCNF} \quad H[F]_p \\
H[(F \lor \top)]_p & \Rightarrow \text{OCNF} \quad H[\top]_p \\
H[(F \land \bot)]_p & \Rightarrow \text{OCNF} \quad H[\bot]_p \\
H[(F \land (F \lor G))]_p & \Rightarrow \text{OCNF} \quad H[F]_p \\
H[(F \lor (F \land G))]_p & \Rightarrow \text{OCNF} \quad H[F]_p \\
H[(F \land \neg F)]_p & \Rightarrow \text{OCNF} \quad H[\bot]_p \\
H[(F \lor \neg F)]_p & \Rightarrow \text{OCNF} \quad H[\top]_p \\
H[(\neg \top)]_p & \Rightarrow \text{OCNF} \quad H[\bot]_p \\
H[(\neg \bot)]_p & \Rightarrow \text{OCNF} \quad H[\top]_p \\
H[(F \rightarrow \bot)]_p & \Rightarrow \text{OCNF} \quad H[\neg F]_p \\
H[(F \rightarrow \top)]_p & \Rightarrow \text{OCNF} \quad H[\top]_p \\
H[(\bot \rightarrow F)]_p & \Rightarrow \text{OCNF} \quad H[F]_p \\
H[(\top \rightarrow F)]_p & \Rightarrow \text{OCNF} \quad H[F]_p
\end{align*}
Note: Applying the absorption laws exhaustively modulo associativity/commutativity of $\land$ and $\lor$ is expensive. In practice, it is sufficient to apply them only in those cases that are easy to detect.

Step 2: Introduce top-down fresh variables for beneficial subformulas:

$$H[F]_p \Rightarrow_{OCNF} H[Q]_p \land \text{def}(H, p, Q, F)$$

where $Q$ is new to $H[F]_p$ and $\nu(H[F]_p) > \nu(H[Q]_p \land \text{def}(H, p, Q, F))$.

Remark: Although computing $\nu$ is not practical in general, the test $\nu(H[F]_p) > \nu(H[Q]_p \land \text{def}(H, p, Q, F))$ can be computed in constant time.

Step 3: Eliminate equivalences dependent on their polarity:

$$H[F \leftrightarrow G]_p \Rightarrow_{OCNF} H[(F \rightarrow G) \land (G \rightarrow F)]_p$$

if $\text{pol}(F, p) = 1$ or $\text{pol}(F, p) = 0$.

$$H[F \leftrightarrow G]_p \Rightarrow_{OCNF} H[(F \land G) \lor (\neg F \land \neg G)]_p$$

if $\text{pol}(F, p) = -1$.

Step 4: Apply steps 2, 3, 4, 5 of $\Rightarrow_{CNF}$

Remark: The $\Rightarrow_{OCNF}$ algorithm is already close to a state of the art algorithm, but some additional redundancy tests and simplification mechanisms are missing.
2.6 The DPLL Procedure

Goal:
Given a propositional formula in CNF (or alternatively, a finite set $N$ of clauses), check whether it is satisfiable (and optionally: output one solution, if it is satisfiable).

Preliminaries

Recall:
- $A \models N$ if and only if $A \models C$ for all clauses $C$ in $N$.
- $A \models C$ if and only if $A \models L$ for some literal $L \in C$.

Assumptions:
- Clauses contain neither duplicated literals nor complementary literals.
- The order of literals in a clause is irrelevant.
- $\Rightarrow$ Clauses behave like sets of literals.

Notation:
- We use the notation $C \lor L$ to denote a clause with some literal $L$ and a clause rest $C$. Here $L$ need not be the last literal of the clause and $C$ may be empty.
- $\overline{L}$ is the complementary literal of $L$, i.e., $\overline{P} = \neg P$ and $\overline{\neg P} = P$.

Partial Valuations

Since we will construct satisfying valuations incrementally, we consider partial valuations (that is, partial mappings $\mathcal{A} : \Pi \rightarrow \{0, 1\}$).

Every partial valuation $\mathcal{A}$ corresponds to a set $M$ of literals that does not contain complementary literals, and vice versa:
- $\mathcal{A}(L)$ is true, if $L \in M$.
- $\mathcal{A}(L)$ is false, if $\overline{L} \in M$.
- $\mathcal{A}(L)$ is undefined, if neither $L \in M$ nor $\overline{L} \in M$.

We will use $\mathcal{A}$ and $M$ interchangeably.

A clause is true under a partial valuation $\mathcal{A}$ (or under a set $M$ of literals) if one of its literals is true; it is false (or “conflicting”) if all its literals are false; otherwise it is undefined (or “unresolved”).
Unit Clauses

Observation:
Let $A$ be a partial valuation. If the set $N$ contains a clause $C$, such that all literals but one in $C$ are false under $A$, then the following properties are equivalent:

- there is a valuation that is a model of $N$ and extends $A$.
- there is a valuation that is a model of $N$ and extends $A$ and makes the remaining literal $L$ of $C$ true.

$C$ is called a unit clause; $L$ is called a unit literal.

Pure Literals

One more observation:
Let $A$ be a partial valuation and $P$ a variable that is undefined under $A$. If $P$ occurs only positively (or only negatively) in the unresolved clauses in $N$, then the following properties are equivalent:

- there is a valuation that is a model of $N$ and extends $A$.
- there is a valuation that is a model of $N$ and extends $A$ and assigns 1 (0) to $P$.

$P$ is called a pure literal.

The Davis-Putnam-Logemann-Loveland Procedure

```java
boolean DPLL(literal set $M$, clause set $N$) {
    if (all clauses in $N$ are true under $M$) return true;
    elsif (some clause in $N$ is false under $M$) return false;
    elsif ($N$ contains unit literal $P$) return DPLL($M \cup \{P\}$, $N$);
    elsif ($N$ contains unit literal $\neg P$) return DPLL($M \cup \{\neg P\}$, $N$);
    elsif ($N$ contains pure literal $P$) return DPLL($M \cup \{P\}$, $N$);
    elsif ($N$ contains pure literal $\neg P$) return DPLL($M \cup \{\neg P\}$, $N$);
    else {
        let $P$ be some undefined variable in $N$;
        if (DPLL($M \cup \{P\}$, $N$)) return true;
        else return DPLL($M \cup \{\neg P\}$, $N$);
    }
}
```

Initially, DPLL is called with an empty literal set and the clause set $N$. 

28
2.7 From DPLL to CDCL

In practice, there are several changes to the procedure:

The pure literal check is only done while preprocessing (otherwise is too expensive).

The algorithm is implemented iteratively ⇒ the backtrack stack is managed explicitly (it may be possible and useful to backtrack more than one level).

Information is reused by conflict analysis and learning.

The branching variable is not chosen randomly.

Under certain circumstances, the procedure is restarted.

Conflict Analysis and Learning

Conflict analysis serves a dual purpose:

\textit{Backjumping (non-chronological backtracking):} If we detect that the conflict is independent of some earlier branch, we can skip over that backtrack level.

\textit{Learning:} By deriving a new clause from the conflict that is added to the current set of clauses, we can reuse information that is obtained in one branch in further branches. (Note: This may produce a large number of new clauses; therefore it may become necessary to delete some of them afterwards to save space.)

These ideas are implemented in all modern SAT solvers.

Because of the importance of clause learning the algorithm is now called CDCL: Conflict Driven Clause Learning.

Formalizing CDCL

We model the improved DPLL procedure by a transition relation \( \Rightarrow_{\text{CDCL}} \) on a set of states.

States:

- \textit{fail}
- \( M \parallel N \),

where \( M \) is a list of annotated literals ("trail") and \( N \) is a set of clauses.

Annotated literal:

- \( L \): deduced literal, due to unit propagation.
- \( L^d \): decision literal (guessed literal).
Unit Propagate:
\[ M \parallel N \cup \{C \lor L\} \Rightarrow_{\text{CDCL}} M L \parallel N \cup \{C \lor L\} \]
if \( C \) is false under \( M \) and \( L \) is undefined under \( M \).

Decide:
\[ M \parallel N \Rightarrow_{\text{CDCL}} M L^d \parallel N \]
if \( L \) is undefined under \( M \) and contained in \( N \).

Fail:
\[ M \parallel N \cup \{C\} \Rightarrow_{\text{CDCL}} \text{fail} \]
if \( C \) is false under \( M \) and \( M \) contains no decision literals.

Backjump:
\[ M' L^d M'' \parallel N \Rightarrow_{\text{CDCL}} M' L' \parallel N \]
if there is some “backjump clause” \( C \lor L' \) such that
\( N \models C \lor L' \),
\( C \) is false under \( M' \), and
\( L' \) is undefined under \( M' \).

We will see later that the Backjump rule is always applicable, if the list of literals \( M \)
contains at least one decision literal and some clause in \( N \) is false under \( M \).

There are many possible backjump clauses. One candidate: \( L_1 \lor \ldots \lor L_n \), where the \( L_i \)
are all the decision literals in \( M' L^d M'' \). (But usually there are better choices.)

**Lemma 2.16** If we reach a state \( M \parallel N \) starting from \( \epsilon \parallel N \), then:

1. \( M \) does not contain complementary literals.
2. Every deduced literal \( L \) in \( M \) follows from \( N \) and decision literals occurring before \( L \) in \( M \).

**Proof.** By induction on the length of the derivation. \( \square \)
Lemma 2.17 Every derivation starting from $\varepsilon \parallel N$ terminates.

Proof. Let $M \parallel N$ and $M' \parallel N'$ be two CDCL states, such that $M = M_0 L_1^d M_1 \ldots L_k^d M_k$ and $M' = M_0' L_1^d M_1' \ldots L_k^d M_k'$. We define a relation $\succ$ on lists of annotated literals by

\begin{itemize}
  \item[(i)] there is some $j$ such that $0 \leq j \leq \min(k, k')$, $|M_i| = |M_i'|$ for all $0 \leq i < j$, and $|M_j| < |M_j'|$, or
  \item[(ii)] $|M_i| = |M_i'|$ for all $0 < i < k < k'$ and $|M| < |M'|$.
\end{itemize}

It is routine to check that $\succ$ is irreflexive and transitive, hence a strict partial ordering, and that for every CDCL step $M \parallel N \Rightarrow_{\text{CDCL}} M' \parallel N'$ we have $M \succ M'$. Moreover, the set of propositional variables in $N$ is finite, and each of these variables can occur at most once in a literal list (positively or negatively, with or without a d-superscript). So there are only finitely many literal lists that can occur in a CDCL derivation. Consequently, if there were an infinite CDCL derivation, there would be some cycle $M \parallel N \Rightarrow_{\text{CDCL}}^+ M'$, so by transitivity $M \succ M'$, but that would contradict the irreflexivity of $\succ$.

Lemma 2.18 Suppose that we reach a state $M \parallel N$ starting from $\varepsilon \parallel N$ such that some clause $D \in N$ is false under $M$. Then:

(1) If $M$ does not contain any decision literal, then “Fail” is applicable.

(2) Otherwise, “Backjump” is applicable.

Proof. (1) Obvious.

(2) Let $L_1, \ldots, L_n$ be the decision literals occurring in $M$ (in this order). Since $M \models \neg D$, we obtain, by Lemma 2.16, $N \cup \{L_1, \ldots, L_n\} \models \neg D$. Since $D \in N$, this is a contradiction, so $N \cup \{L_1, \ldots, L_n\}$ is unsatisfiable. Consequently, $N \models \neg L_1 \lor \cdots \lor \neg L_n$. Now let $C = \neg L_1 \lor \cdots \lor \neg L_{n-1}$, $L' = \neg L_n$, $L = L_n$, and let $M'$ be the list of all literals of $M$ occurring before $L_n$, then the condition of “Backjump” is satisfied. \qed

Theorem 2.19 Suppose that we reach a final state starting from $\varepsilon \parallel N$.

(1) If the final state is $M \parallel N$, then $N$ is satisfiable and $M$ is a model of $N$.

(2) If the final state is fail, then $N$ is unsatisfiable.

Proof. (1) Observe that the “Decide” rule is applicable as long as literals in $N$ are undefined under $M$. Hence, in a final state, all literals must be defined. Furthermore, in a final state, no clause in $N$ can be false under $M$, otherwise “Fail” or “Backjump” would be applicable. Hence $M$ is a model of every clause in $N$.

(2) If we reach fail, then in the previous step we must have reached a state $M \parallel N$ such that some $C \in N$ is false under $M$ and $M$ contains no decision literals. By part (2) of Lemma 2.16, every literal in $M$ follows from $N$. On the other hand, $C \in N$, so $N$ must be unsatisfiable. \qed
Getting Better Backjump Clauses

Suppose that we have reached a state \( M \parallel N \) such that some clause \( C \in N \) (or entailed by \( N \)) is false under \( M \).

Consequently, every literal of \( C \) is the complement of some literal in \( M \).

1. If every literal in \( C \) is the complement of a decision literal of \( M \), then \( C \) is either a backjump clause or \( \bot \).

2. Otherwise, \( C = C' \lor L \), such that \( L \) is a deduced literal.

   For every deduced literal \( L \), there is a (unit or backjump) clause \( D \lor L \), such that \( N \models D \lor L \) and \( D \) is false under \( M \).

   Then \( N \models D \lor C' \) and \( D \lor C' \) is also false under \( M \). (\( D \lor C' \) is a resolvent of \( C' \lor \overline{L} \) and \( D \lor L \).)

As long as we are in case (2), we can repeat this transformation with the new clause \( D \lor C' \).

This process must terminate:

Define an ordering \( \succ \) on literals so that \( \overline{L_2} \succ \overline{L_1} \) if \( L_2 \) occurs right of \( L_1 \) on the trail.

The trail is finite, so \( \succ \) is well-founded, and therefore \( \succ_{\text{mul}} \) is well-founded.

Then the multiset of literals in \( C' \lor \overline{L} \) is larger than the multiset of literals in \( D \lor C' \) with respect to \( \succ_{\text{mul}} \).

So we must eventually reach case (1): We obtain a backjump clause or the empty clause.

Usually, one resolves the literals in the reverse order in which they were added to \( M \) and stops as soon as one obtains a clause in which all but one literal are complements of literals occurring in \( M \) before the last decision literal.

\( \Rightarrow \) 1UIP (first unique implication point) strategy.
Learning Clauses

Backjump clauses are good candidates for learning.

To model learning, the CDCL system is extended by the following two rules:

Learn:

\[ M \parallel N \Rightarrow_{\text{CDCL}} M \parallel N \cup \{C\} \]

if \( N \models C \).

Forget:

\[ M \parallel N \cup \{C\} \Rightarrow_{\text{CDCL}} M \parallel N \]

if \( N \models C \).

If we ensure that no clause is learned infinitely often, then termination is guaranteed.

The other properties of the basic CDCL system hold also for the extended system.

Restart

Runtimes of CDCL-style procedures depend extremely on the choice of branching variables.

If no solution is found within a certain time limit, it can be useful to restart from scratch with an adapted variable selection heuristics. Learned clauses, however, are kept.

In addition, it is useful to restart after a unit clause has been learned.

The restart rule is typically applied after a certain number of clauses have been learned or a unit is derived:

Restart:

\[ M \parallel N \Rightarrow_{\text{CDCL}} \emptyset \parallel N \]

If Restart is only applied finitely often, termination is guaranteed.
2.8 Implementing CDCL

The formalization of CDCL that we have seen so far leaves many aspects unspecified. To get a fast solver, we must use good heuristics, for instance to choose the next undefined variable, and we must implement basic operations efficiently.

**Variable Order Heuristic**

Choosing the right undefined variable to branch is important for efficiency, but the branching heuristics may be expensive itself.

State of the art: Use branching heuristics that need not be recomputed too frequently.

In general: Choose variables that occur frequently; after a restart prefer variables from recent conflicts.

The VSIDS (Variable State Independent Decaying Sum) heuristic:

- We associate a positive score to every propositional variable $P_i$. At the start, $k_i$ is the number of occurrences of $P_i$ in $N$.
- The variable order is then the descending ordering of the $P_i$ according to the $k_i$.

The scores $k_i$ are adjusted during a CDCL run.

- Every time a learned clause is computed after a conflict, the propositional variables in the learned clause obtain a bonus $b$, i.e., $k_i := k_i + b$.
- Periodically, the scores are leveled: $k_i := k_i/l$ for some $l$.
- After each restart, the variable order is recomputed, using the new scores.

The purpose of these mechanisms is to keep the search focused. The parameter $b$ directs the search around the conflict.

Further refinements:

- Add the bonus to all literals in the clauses that occur in the resolution steps to generate a backjump clause.
- If the score of a variable reaches a certain limit, all scores are rescaled by a constant.
- Occasionally (with low probability) choose a variable at random, otherwise choose the undefined variable with the highest score.
Implementing Unit Propagation Efficiently

For applying the unit rule, we need to know the number of literals in a clause that are not false.

Maintaining this number is expensive, however.

Better approach: “Two watched literals”:

In each clause, select two (currently undefined) “watched” literals.

For each variable $P$, keep a list of all clauses in which $P$ is watched and a list of all clauses in which $\neg P$ is watched.

If an undefined variable is set to 0 (or to 1), check all clauses in which $P$ (or $\neg P$) is watched and watch another literal (that is true or undefined) in this clause if possible.

Watched literal information need not be restored upon backtracking.

Preprocessing

Some operations are only needed once at the beginning of the CDCL run.

(i) Deletion of tautologies

(ii) Deletion of duplicated literals

Inprocessing

Some operations are useful, but expensive. They are performed only initially and after restarts (before computation of the variable order heuristics), possibly with time limits.

Note: Some of these operations are only satisfiability-preserving; they do not yield equivalent clause sets.

Examples:

(i) Subsumption

$$N \cup \{C\} \cup \{D\} \Rightarrow N \cup \{C\}$$

if $C \subseteq D$ considering $C$, $D$ as multisets of literals.

(ii) Purity deletion

Delete all clauses containing a literal $L$ where $\overline{L}$ does not occur in the clause set.
(iii) Merging replacement resolution

\[ N \cup \{C \lor L\} \cup \{D \lor \overline{L}\} \Rightarrow N \cup \{C \lor L\} \cup \{D\} \]

if \(C \subseteq D\) considering \(C, D\) as multisets of literals.

(iv) Bounded variable elimination

Compute all possible resolution steps

\[
\frac{C \lor L}{C \lor D} \quad \frac{D \lor \overline{L}}{C \lor D}
\]

on a literal \(L\) with premises in \(N\); add all non-tautological conclusions to \(N\); then throw away all clauses containing \(L\) or \(\overline{L}\); repeat this as long as \(|N|\) does not grow.

(v) RAT ("Resolution asymmetric tautologies")

\(C\) is called an asymmetric tautology w.r.t. \(N\), if its negation can be refuted by unit propagation using clauses in \(N\).

\(C\) has the RAT property w.r.t. \(N\), if it is an asymmetric tautology w.r.t. \(N\), or if there is a literal \(L\) in \(C\) such that \(C = C' \lor L\) and all clauses \(D' \lor C'\) for \(D' \lor \overline{L} \in N\) are asymmetric tautologies w.r.t. \(N\).

RAT elimination:

\[ N \cup \{C\} \Rightarrow N \]

if \(C\) has the RAT property w.r.t. \(N\).

Literature


Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh (eds.): Handbook of Satisfiability; IOS Press, 2009

Daniel Le Berre’s slides at VTSA’09: http://www.mpi-inf.mpg.de/vtsa09/.
2.9 OBDDs

Goal:

Efficient manipulation of (equivalence classes of) propositional formulas.

Method: Minimized graph representation of decision trees, based on a fixed ordering on propositional variables.

⇒ Canonical representation of formulas.

⇒ Satisfiability checking as a side effect.

BDDs

BDD (Binary decision diagram):

Labelled DAG (directed acyclic graph).

Leaf nodes:

labelled with a truth value (0 or 1).

Non-leaf nodes (interior nodes):

labelled with a propositional variable,

exactly two outgoing edges, labelled with 0 (→→) and 1 (→→)

Every BDD node can be interpreted as a mapping from valuations to truth values: Traverse the BDD from the given node to a leaf node; for any node labelled with \( P \) take the 0-edge or 1-edge depending on whether \( \mathcal{A}(P) \) is 0 or 1.

⇒ Compact representation of truth tables.
OBDDs

OBDD (Ordered BDD):

Let $<$ be a total ordering of the propositional variables.

An OBDD w.r.t. $<$ is a BDD where every edge from a non-leaf node leads either to a leaf node or to a non-leaf node with a strictly larger label w.r.t. $<$. 

OBDDs and formulas:

A leaf node $\square 0$ represents $\bot$ (or any unsatisfiable formula).

A leaf node $\square 1$ represents $\top$ (or any valid formula).

If a non-leaf node $v$ has the label $P$, and its 0-edge leads to a node representing the formula $F_0$, and its 1-edge leads to a node representing the formula $F_1$, then $v$ represents the formula

$$F \models \text{if } P \text{ then } F_1 \text{ else } F_0$$
$$= (P \wedge F_1) \lor (\neg P \wedge F_0)$$

Conversely:

Define $F\{P \mapsto H\}$ as the formula obtained from $F$ by replacing every occurrence of $P$ in $F$ by $H$.

For every formula $F$ and propositional variable $P$:

$$F \models (P \wedge F\{P \mapsto \top\}) \lor (\neg P \wedge F\{P \mapsto \bot\})$$

(Shannon expansion of $F$, originally due to Boole).

Consequence: Every formula $F$ can be represented by an OBDD.
Reduced OBDDs

An OBDD is called reduced, if it has

- no duplicated leaf nodes

- no duplicated interior nodes

- no redundant tests

Theorem 2.20 (Bryant 1986) Every OBDD can be converted into an equivalent reduced OBDD.

Assumptions from now on:
- One fixed ordering $>$. 
- We consider only reduced OBDDs.
- All OBDDs are sub-OBDDs of a single OBDD.

Implementation:
- Bottom-up construction of reduced OBDDs is possible using a hash table.
- Keys and values are triples $(\text{PropVar}, \text{Ptr}_0, \text{Ptr}_1)$,
- where $\text{Ptr}_0$ and $\text{Ptr}_1$ are pointers to the 0-successor and 1-successor hash table entry.
Theorem 2.21 (Bryant 1986) If $v$ and $v'$ are two different nodes in a reduced OBDD, then they represent non-equivalent formulas.

Proof. We use induction over the maximum of the numbers of nodes reachable from $v$ and $v'$, respectively. Let $F$ and $F'$ be the formulas represented by $v$ and $v'$.

Case 1: $v$ and $v'$ are non-leaf nodes labelled by different propositional variables $P$ and $P'$. Without loss of generality, $P < P'$.

Let $v_0$ and $v_1$ be the 0-successor and the 1-successor of $v$, and let $F_0$ and $F_1$ be formulas represented by $v_0$ and $v_1$. We may assume without loss of generality that all propositional variables occurring in $F_0$, $F_1$, and $F'$ are larger than $P$. By reducedness, $v_0 \neq v_1$, so by induction, $F_0 \not\models F_1$. Hence there must be a valuation $\mathcal{A}$ such that $\mathcal{A}(F_0) \neq \mathcal{A}(F_1)$.

Define valuations $\mathcal{A}_0$ and $\mathcal{A}_1$ by

\[
\begin{align*}
\mathcal{A}_0(P) &= 0 \\
\mathcal{A}_0(Q) &= \mathcal{A}(Q) \\
\mathcal{A}_1(P) &= 1 \\
\mathcal{A}_1(Q) &= \mathcal{A}(Q)
\end{align*}

\]

for all $Q \neq P$.

We know that the node $v$ represents $F \models (P \land F_1) \lor (\neg P \land F_0)$, so $\mathcal{A}_0(F) = \mathcal{A}_0(F_0) = \mathcal{A}(F_0)$ and $\mathcal{A}_1(F) = \mathcal{A}_1(F_1) = \mathcal{A}(F_1)$, and therefore $\mathcal{A}_0(F) \neq \mathcal{A}_1(F)$. On the other hand, $P$ does not occur in $F'$, therefore $\mathcal{A}_0(F') = \mathcal{A}_1(F')$. So we must have $\mathcal{A}_0(F) \neq \mathcal{A}_0(F')$ or $\mathcal{A}_1(F) \neq \mathcal{A}_1(F')$, which implies $F \not\models F'$.

Case 2: $v$ and $v'$ are non-leaf nodes labelled by the same propositional variable.

Case 3: $v$ is a non-leaf node, $v'$ is a non-leaf node, or vice versa.

Case 4: $v$ and $v'$ are different leaf nodes.

Analogously. \hfill \Box

Corollary 2.22 $F$ is valid, if and only if it is represented by $\boxed{1}$ $F$ is unsatisfiable, if and only if it is represented by $\boxed{0}$. 

40
Operations on OBDDs

Example:

Let \( \circ \) be a binary connective.
Let \( P \) be the smallest propositional variable that occurs in \( F \) or \( G \) or both.

\[
F \circ G \models (P \land (F \circ G)\{P \mapsto \top\}) \lor (\neg P \land (F \circ G)\{P \mapsto \bot\})
\]
\[
\models (P \land (F\{P \mapsto \top\} \circ G\{P \mapsto \top\})
\lor (\neg P \land (F\{P \mapsto \bot\} \circ G\{P \mapsto \bot\}))
\]

Note: \( F\{P \mapsto \top\} \) is either represented by the same node as \( F \) (if \( P \) does not occur in \( F \)), or by its 1-successor (otherwise).

\[\Rightarrow\] Obvious recursive function on OBDD nodes
(needs memoizing for efficient implementation).

OBDD operations are not restricted to the connectives of propositional logic.

We can also compute operations of quantified boolean formulas

\[
\forall P. F \models (F\{P \mapsto \top\}) \land (F\{P \mapsto \bot\})
\]
\[
\exists P. F \models (F\{P \mapsto \top\}) \lor (F\{P \mapsto \bot\})
\]

and images or preimages of propositional formulas w.r.t. boolean relations (needed for typical verification tasks).

The size of the OBDD for \( F \circ G \) is bounded by \( mn \), where \( F \) has size \( m \) and \( G \) has size \( n \). (Size = number of nodes)

With memoization, the time for computing \( F \circ G \) is also at most \( O(mn) \).

The size of an OBDD for a given formula depends crucially on the chosen ordering of the propositional variables:

Let \( F = (P_1 \land P_2) \lor (P_3 \land P_4) \lor \cdots \lor (P_{2n-1} \land P_{2n}). \)

\begin{align*}
P_1 &< P_2 < P_3 < P_4 < \cdots < P_{2n-1} < P_{2n}: \quad 2n + 2 \text{ nodes.} \\
P_1 &< P_3 < \cdots < P_{2n-1} < P_2 < P_4 < \cdots < P_{2n}: \quad 2^{n+1} \text{ nodes.}
\end{align*}

Even worse: There are (practically relevant!) formulas for which the OBDD has exponential size for every ordering of the propositional variables.

Example: middle bit of binary multiplication.
Literature


2.10 FRAIGs

Goal:
Efficient manipulation of (equivalence classes of) propositional formulas.
Smaller representation than OBDDs.
Method: Minimized graph representation of boolean circuits.

FRAIG (Functionally Reduced And-Inverter Graph):
Labelled DAG (directed acyclic graph).
Leaf nodes:
labelled with propositional variables.
Non-leaf nodes (interior nodes):
labelled with $\land$ (two outgoing edges) or $\neg$ (one outgoing edge).

Reducedness (i.e., no two different nodes represent equivalent formulas) must be established explicitly, using
 structural hashing,
 simulation vectors,
 CDCL,
 OBDDs.

$\Rightarrow$ Semi-canonical representation of formulas.

Literature


2.11 Other Calculi

Ordered resolution
Tableau calculus
Hilbert calculus
Sequent calculus
Natural deduction

see next chapter
3 First-Order Logic

First-order logic

• formalizes fundamental mathematical concepts
• is expressive (Turing-complete)
• is not too expressive (e.g. not axiomatizable: natural numbers, uncountable sets)
• has a rich structure of decidable fragments
• has a rich model and proof theory

First-order logic is also called (first-order) predicate logic.

3.1 Syntax

Syntax:

• non-logical symbols (domain-specific)
  ⇒ terms, atomic formulas
• logical connectives (domain-independent)
  ⇒ Boolean combinations, quantifiers

Signatures

A signature Σ = (Ω, Π) fixes an alphabet of non-logical symbols, where

• Ω is a set of function symbols f with arity \( n \geq 0 \), written \( \text{arity}(f) = n \),
• Π is a set of predicate symbols P with arity \( m \geq 0 \), written \( \text{arity}(P) = m \).

Function symbols are also called operator symbols.
If \( n = 0 \) then f is also called a constant (symbol).
If \( m = 0 \) then P is also called a propositional variable.

We will usually use

\( b, c, d \) for constant symbols,
\( f, g, h \) for non-constant function symbols,
\( P, Q, R, S \) for predicate symbols.
Convention: We will usually write $f/n \in \Omega$ instead of $f \in \Omega$, $\text{arity}(f) = n$ (analogously for predicate symbols).

Refined concept for practical applications: many-sorted signatures (corresponds to simple type systems in programming languages); no big change from a logical point of view.

Variables

Predicate logic admits the formulation of abstract, schematic assertions. (Object) variables are the technical tool for schematization.

We assume that $X$ is a given countably infinite set of symbols which we use to denote variables.

Terms

Terms over $\Sigma$ and $X$ ($\Sigma$-terms) are formed according to these syntactic rules:

\[
 s, t, u, v ::= x, \quad x \in X \quad \text{(variable)} \\
     | f(s_1, \ldots, s_n), \quad f/n \in \Omega \quad \text{(functional term)}
\]

By $T_{\Sigma}(X)$ we denote the set of $\Sigma$-terms (over $X$). A term not containing any variable is called a ground term. By $T_{\Sigma}$ we denote the set of $\Sigma$-ground terms.

In other words, terms are formal expressions with well-balanced parentheses which we may also view as marked, ordered trees. The markings are function symbols or variables. The nodes correspond to the subterms of the term. A node $v$ that is marked with a function symbol $f$ of arity $n$ has exactly $n$ subtrees representing the $n$ immediate subterms of $v$.

Atoms

Atoms (also called atomic formulas) over $\Sigma$ are formed according to this syntax:

\[
 A, B ::= P(s_1, \ldots, s_m), \quad P/m \in \Pi \quad \text{(non-equational atom)} \\
     \left[ \quad (s \approx t) \quad \text{(equation)} \right]
\]

Whenever we admit equations as atomic formulas we are in the realm of first-order logic with equality. Admitting equality does not really increase the expressiveness of first-order logic (see next chapter). But deductive systems where equality is treated specifically are much more efficient.
Literals

\[ L ::= A \text{ (positive literal)} \]
| \( \neg A \text{ (negative literal)} \)

Clauses

\[ C, D ::= \bot \text{ (empty clause)} \]
| \( L_1 \lor \ldots \lor L_k, \ k \geq 1 \text{ (non-empty clause)} \)

General First-Order Formulas

\( F_{\Sigma}(X) \) is the set of first-order formulas over \( \Sigma \) defined as follows:

\[ F, G, H ::= \bot \text{ (falsum)} \]
| \( \top \text{ (verum)} \)
| \( A \text{ (atomic formula)} \)
| \( \neg F \text{ (negation)} \)
| \( (F \land G) \text{ (conjunction)} \)
| \( (F \lor G) \text{ (disjunction)} \)
| \( (F \rightarrow G) \text{ (implication)} \)
| \( (F \leftrightarrow G) \text{ (equivalence)} \)
| \( \forall x \ F \text{ (universal quantification)} \)
| \( \exists x \ F \text{ (existential quantification)} \)

Notational Conventions

We omit parentheses according to the conventions for propositional logic.

\( \forall x_1, \ldots, x_n \ F \) and \( \exists x_1, \ldots, x_n \ F \) abbreviate \( \forall x_1 \ldots \forall x_n \ F \) and \( \exists x_1 \ldots \exists x_n \ F \).

We use infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences.

Examples:

\[ s + t \ast u \text{ for } +(s, *(t, u)) \]
\[ s \ast u \leq t + v \text{ for } \leq (*s, u), +(t, v)) \]
\[ -s \text{ for } -(s) \]
\[ s! \text{ for } !(s) \]
\[ |s| \text{ for } |](s) \]
\[ 0 \text{ for } 0() \]
Example: Peano Arithmetic

\[ \Sigma_{\text{PA}} = (\Omega_{\text{PA}}, \Pi_{\text{PA}}) \]
\[ \Omega_{\text{PA}} = \{0/0, +/2, */2, s/1\} \]
\[ \Pi_{\text{PA}} = \{</2\} \]

Examples of formulas over this signature are:

\[ \forall x, y ((x < y \lor x \approx y) \leftrightarrow \exists z (x + z \approx y)) \]
\[ \exists x \forall y (x + y \approx y) \]
\[ \forall x, y (x \ast s(y) \approx x \ast y + x) \]
\[ \forall x, y (s(x) \approx s(y) \rightarrow x \approx y) \]
\[ \forall x \exists y (x < y \land \neg \exists z (x < z \land z < y)) \]

Positions in Terms and Formulas

The set of positions is extended from propositional logic to first-order logic:

The positions of a term \( s \) (formula \( F \)):

\[ \text{pos}(x) = \{\varepsilon\}, \]
\[ \text{pos}(f(s_1, \ldots, s_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{ ip \mid p \in \text{pos}(s_i) \}, \]
\[ \text{pos}(P(t_1, \ldots, t_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{ ip \mid p \in \text{pos}(t_i) \}, \]
\[ \text{pos}(\forall x F) = \{\varepsilon\} \cup \{ 1p \mid p \in \text{pos}(F) \}, \]
\[ \text{pos}(\exists x F) = \{\varepsilon\} \cup \{ 1p \mid p \in \text{pos}(F) \}. \]

The prefix order \( \leq \), the subformula (subterm) operator, the formula (term) replacement operator and the size operator are extended accordingly. See the definitions in Sect. 2.

Variables

The set of variables occurring in a term \( t \) is denoted by \( \text{var}(t) \) (and analogously for atoms, literals, clauses, and formulas).
Bound and Free Variables

In $Qx F$, $Q \in \{\exists, \forall\}$, we call $F$ the scope of the quantifier $Qx$. An occurrence of a variable $x$ is called bound, if it is inside the scope of a quantifier $Qx$. Any other occurrence of a variable is called free.

Formulas without free variables are also called closed formulas or sentential forms.

Formulas without variables are called ground.

Example:

$$\forall y \ ( (\forall x \ P(x)) \rightarrow R(x, y) )$$

The occurrence of $y$ is bound, as is the first occurrence of $x$. The second occurrence of $x$ is a free occurrence.

Substitutions

Substitution is a fundamental operation on terms and formulas that occurs in all inference systems for first-order logic.

Substitutions are mappings

$$\sigma : X \rightarrow T_\Sigma(X)$$

such that the domain of $\sigma$, that is, the set

$$\text{dom}(\sigma) = \{ x \in X \mid \sigma(x) \neq x \},$$

is finite. The set of variables introduced by $\sigma$, that is, the set of variables occurring in one of the terms $\sigma(x)$, with $x \in \text{dom}(\sigma)$, is denoted by $\text{codom}(\sigma)$.

Substitutions are often written as $\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$, with $x_i$ pairwise distinct, and then denote the mapping

$$\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}(y) = \begin{cases} s_i, & \text{if } y = x_i \\ y, & \text{otherwise} \end{cases}$$

We also write $x\sigma$ for $\sigma(x)$.

The modification of a substitution $\sigma$ at $x$ is defined as follows:

$$\sigma[x \mapsto t](y) = \begin{cases} t, & \text{if } y = x \\ \sigma(y), & \text{otherwise} \end{cases}$$
Why Substitution is Complicated

We define the application of a substitution $\sigma$ to a term $t$ or formula $F$ by structural induction over the syntactic structure of $t$ or $F$ by the equations depicted on the next page.

In the presence of quantification it is surprisingly complex: We need to make sure that the (free) variables in the codomain of $\sigma$ are not captured upon placing them into the scope of a quantifier $Qy$, hence the bound variable must be renamed into a “fresh”, that is, previously unused, variable $z$.

Application of a Substitution

“Homomorphic” extension of $\sigma$ to terms and formulas:

\[
\begin{align*}
  f(s_1, \ldots, s_n)\sigma &= f(s_1\sigma, \ldots, s_n\sigma) \\
  \bot\sigma &= \bot \\
  \top\sigma &= \top \\
  P(s_1, \ldots, s_n)\sigma &= P(s_1\sigma, \ldots, s_n\sigma) \\
  (u \approx v)\sigma &= (u\sigma \approx v\sigma) \\
  \neg F\sigma &= \neg(F\sigma) \\
  (F \circ G)\sigma &= (F\sigma \circ G\sigma) \text{ for each binary connective } \circ \\
  (Qx F)\sigma &= Qz (F\sigma[x \mapsto z]) \text{ with } z \text{ a fresh variable}
\end{align*}
\]

If $s = t\sigma$ for some substitution $\sigma$, we call the term $s$ an instance of the term $t$, and we call $t$ a generalization of $s$ (analogously for formulas).
3.2 Semantics

To give semantics to a logical system means to define a notion of truth for the formulas. The concept of truth that we will now define for first-order logic goes back to Tarski.

As in the propositional case, we use a two-valued logic with truth values “true” and “false” denoted by 1 and 0, respectively.

Algebras

A $\Sigma$-algebra (also called $\Sigma$-interpretation or $\Sigma$-structure) is a triple $A = (U_A, (f_A : U^n_A \to U_A)_{f/n \in \Omega}, (P_A \subseteq U^m_A)_{P/m \in \Pi})$ where $U_A \neq \emptyset$ is a set, called the universe of $A$.

By $\Sigma$-Alg we denote the class of all $\Sigma$-algebras. $\Sigma$-algebras generalize the valuations from propositional logic.

Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A (variable) assignment (over a given $\Sigma$-algebra $A$), is a map $\beta : X \to U_A$.

Variable assignments are the semantic counterparts of substitutions.

Value of a Term in $A$ with Respect to $\beta$

By structural induction we define

$A(\beta) : T_\Sigma(X) \to U_A$

as follows:

$A(\beta)(x) = \beta(x), \quad x \in X$

$A(\beta)(f(s_1, \ldots, s_n)) = f_A(A(\beta)(s_1), \ldots, A(\beta)(s_n)), \quad f/n \in \Omega$

In the scope of a quantifier we need to evaluate terms with respect to modified assignments. To that end, let $\beta[x \mapsto a] : X \to U_A$, for $x \in X$ and $a \in U_A$, denote the assignment

$\beta[x \mapsto a](y) = \begin{cases} 
  a & \text{if } x = y \\
  \beta(y) & \text{otherwise} 
\end{cases}$
Truth Value of a Formula in $A$ with Respect to $\beta$

$A(\beta) : F_\Sigma(X) \rightarrow \{0,1\}$ is defined inductively as follows:

\[
A(\beta)(\bot) = 0 \\
A(\beta)(\top) = 1 \\
A(\beta)(P(s_1, \ldots, s_n)) = \text{if } (A(\beta)(s_1), \ldots, A(\beta)(s_n)) \in P_A \text{ then } 1 \text{ else } 0 \\
A(\beta)(s \approx t) = \text{if } A(\beta)(s) = A(\beta)(t) \text{ then } 1 \text{ else } 0 \\
A(\beta)(\neg F) = 1 - A(\beta)(F) \\
A(\beta)(F \land G) = \min(A(\beta)(F), A(\beta)(G)) \\
A(\beta)(F \lor G) = \max(A(\beta)(F), A(\beta)(G)) \\
A(\beta)(F \rightarrow G) = \max(1 - A(\beta)(F), A(\beta)(G)) \\
A(\beta)(\forall x F) = \min\{A(\beta[x \mapsto a])(F)\} \\
A(\beta)(\exists x F) = \max\{A(\beta[x \mapsto a])(F)\}
\]

Example

The “Standard” Interpretation for Peano Arithmetic:

$U_\mathbb{N} = \{0, 1, 2, \ldots\}$

$0_\mathbb{N} = 0$

$s_\mathbb{N} : n \mapsto n + 1$

$+_\mathbb{N} : (n, m) \mapsto n + m$

$*_\mathbb{N} : (n, m) \mapsto n * m$

$<_\mathbb{N} = \{ (n, m) \mid n \text{ less than } m \}$

Note that $\mathbb{N}$ is just one out of many possible $\Sigma_{PA}$-interpretations.

Values over $\mathbb{N}$ for sample terms and formulas:

Under the assignment $\beta : x \mapsto 1, y \mapsto 3$ we obtain

$\mathbb{N}(\beta)(s(x) + s(0)) = 3$

$\mathbb{N}(\beta)(x + y \approx s(y)) = 1$

$\mathbb{N}(\beta)(\forall x, y \ (x + y \approx y + x)) = 1$

$\mathbb{N}(\beta)(\forall z \ (z < y)) = 0$

$\mathbb{N}(\beta)(\forall x \exists y \ (x < y)) = 1$
Ground Terms and Closed Formulas

If $t$ is a ground term, then $A(\beta)(t)$ does not depend on $\beta$:

$$A(\beta)(t) = A(\beta')(t)$$

for every $\beta$ and $\beta'$.

Analogously, if $F$ is a closed formula, then $A(\beta)(F)$ does not depend on $\beta$:

$$A(\beta)(F) = A(\beta')(F)$$

for every $\beta$ and $\beta'$.

An element $a \in U_A$ is called term-generated, if $a = A(\beta)(t)$ for some ground term $t$.

In general, not every element of an algebra is term-generated.

3.3 Models, Validity, and Satisfiability

$F$ is true in $A$ under assignment $\beta$:

$$A, \beta \models F :\iff A(\beta)(F) = 1$$

$F$ is true in $A$ ($A$ is a model of $F$; $F$ is valid in $A$):

$$A \models F :\iff A, \beta \models F \text{ for all } \beta \in X \rightarrow U_A$$

$F$ is valid (or is a tautology):

$$\models F :\iff A \models F \text{ for all } A \in \Sigma\text{-Alg}$$

$F$ is called satisfiable iff there exist $A$ and $\beta$ such that $A, \beta \models F$. Otherwise $F$ is called unsatisfiable.

Entailment and Equivalence

$F$ entails (implies) $G$ (or $G$ is a consequence of $F$), written $F \models G$, if for all $A \in \Sigma\text{-Alg}$ and $\beta \in X \rightarrow U_A$, whenever $A, \beta \models F$, then $A, \beta \models G$.

$F$ and $G$ are called equivalent, written $F \models G$, if for all $A \in \Sigma\text{-Alg}$ and $\beta \in X \rightarrow U_A$ we have $A, \beta \models F \iff A, \beta \models G$.

Proposition 3.1 $F$ entails $G$ iff $(F \rightarrow G)$ is valid

Proposition 3.2 $F$ and $G$ are equivalent iff $(F \leftrightarrow G)$ is valid.

Extension to sets of formulas $N$ in the “natural way”, e.g., $N \models F$

$:\iff$ for all $A \in \Sigma\text{-Alg}$ and $\beta \in X \rightarrow U_A$: if $A, \beta \models G$, for all $G \in N$, then $A, \beta \models F$. 

52
Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

**Proposition 3.3** Let $F$ and $G$ be formulas, let $N$ be a set of formulas. Then

(i) $F$ is valid if and only if $\neg F$ is unsatisfiable.

(ii) $F \models G$ if and only if $F \land \neg G$ is unsatisfiable.

(iii) $N \models G$ if and only if $N \cup \{\neg G\}$ is unsatisfiable.

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

Substitution Lemma

The following propositions, to be proved by structural induction, hold for all $\Sigma$-algebras $A$, assignments $\beta$, and substitutions $\sigma$.

**Lemma 3.4** For any $\Sigma$-term $t$

$$A(\beta)(t\sigma) = A(\beta \circ \sigma)(t),$$

where $\beta \circ \sigma : X \rightarrow U_A$ is the assignment $\beta \circ \sigma(x) = A(\beta)(x\sigma)$.

**Proposition 3.5** For any $\Sigma$-formula $F$, $A(\beta)(F\sigma) = A(\beta \circ \sigma)(F)$.

**Corollary 3.6** $A, \beta \models F\sigma \iff A, \beta \circ \sigma \models F$

These theorems basically express that the syntactic concept of substitution corresponds to the semantic concept of an assignment.

Two Lemmas

**Lemma 3.7** Let $A$ be a $\Sigma$-algebra and let $F$ be a $\Sigma$-formula with free variables $x_1, \ldots, x_n$. Then

$$A \models \forall x_1, \ldots, x_n F \text{ if and only if } A \models F$$

**Lemma 3.8** Let $F$ be a $\Sigma$-formula with free variables $x_1, \ldots, x_n$, let $\sigma$ be a substitution, and let $y_1, \ldots, y_m$ be the free variables of $F\sigma$. Then

$$A \models \forall x_1, \ldots, x_n F \text{ implies } A \models \forall y_1, \ldots, y_m F\sigma$$
3.4 Algorithmic Problems

Validity($F$): $\models F$?
Satisfiability($F$): $F$ satisfiable?
Entailment($F,G$): does $F$ entail $G$?
Model($A,F$): $A \models F$?
Solve($A,F$): find an assignment $\beta$ such that $A, \beta \models F$.
Solve($F$): find a substitution $\sigma$ such that $\models F\sigma$.
Abduce($F$): find $G$ with “certain properties” such that $G \models F$.

Theory of an Algebra

Let $A \in \Sigma$-Alg. The (first-order) theory of $A$ is defined as

$$Th(A) = \{ G \in F_{\Sigma}(X) \mid A \models G \}$$

Problem of axiomatizability:

For which algebras $A$ can one axiomatize $Th(A)$, that is, can one write down a formula $F$ (or a recursively enumerable set $F$ of formulas) such that

$$Th(A) = \{ G \mid F \models G \}?$$

(analogously for classes of algebras).

Two Interesting Theories

Let $\Sigma_{\text{Pres}} = (\{0/0, s/1, +/2\}, \{<\})$ and $\mathbb{N}_+ = (\mathbb{N}, 0, s, +, <)$ its standard interpretation on the natural numbers. $Th(\mathbb{N}_+)$ is called Presburger arithmetic (M. Presburger, 1929). (There is no essential difference when one, instead of $\mathbb{N}$, considers the integer numbers $\mathbb{Z}$ as standard interpretation.)

Presburger arithmetic is decidable in 3EXPTIME (D. Oppen, JCSS, 16(3):323–332, 1978), and in 2EXPSPACE, using automata-theoretic methods (and there is a constant $c \geq 0$ such that $Th(\mathbb{Z}_+) \not\in \text{NTIME}(2^{2^n})$).

However, $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *, <)$, the standard interpretation of $\Sigma_{\text{PA}} = (\{0/0, s/1, +/2, */2\}, \{<\})$, has as theory the so-called Peano arithmetic which is undecidable and not even recursively enumerable.
(Non-)Computability Results

1. For most signatures $\Sigma$, validity is undecidable for $\Sigma$-formulas.
   (One can easily encode Turing machines in most signatures.)

2. Gödel’s completeness theorem:
   For each signature $\Sigma$, the set of valid $\Sigma$-formulas is recursively enumerable.
   (We will prove this by giving complete deduction systems.)

3. Gödel’s incompleteness theorem:
   For $\Sigma = \Sigma_{PA}$ and $\mathbb{N}^* = (\mathbb{N}, 0, s, +, \times, <)$, the theory $Th(\mathbb{N}^*)$ is not recursively enumerable.

These complexity results motivate the study of subclasses of formulas (fragments) of first-order logic

Some Decidable Fragments

Some decidable fragments:

- **Monadic class**: no function symbols, all predicates unary; validity is NEXPTIME-complete.

- Variable-free formulas without equality: satisfiability is NP-complete. (why?)

- Variable-free Horn clauses (clauses with at most one positive atom): entailment is decidable in linear time.

- Finite model checking is decidable in exponential time and PSPACE-complete.

3.5 Normal Forms and Skolemization

Study of normal forms motivated by

- reduction of logical concepts,

- efficient data structures for theorem proving.

The main problem in first-order logic is the treatment of quantifiers. The subsequent normal form transformations are intended to eliminate many of them.
Prenex Normal Form (Traditional)

Prenex formulas have the form

\[ Q_1 x_1 \ldots Q_n x_n F, \]

where \( F \) is quantifier-free and \( Q_i \in \{ \forall, \exists \} \); we call \( Q_1 x_1 \ldots Q_n x_n \) the quantifier prefix and \( F \) the matrix of the formula.

Computing prenex normal form by the reduction system \( \Rightarrow_p \):

\[
\begin{align*}
H[(F \leftrightarrow G)]_p & \Rightarrow_p H[(F \rightarrow G) \land (G \rightarrow F)]_p \\
H[(\neg Qx F)]_p & \Rightarrow_p H[\overline{Q}x \neg F]_p \\
H[((Qx F) \circ G)]_p & \Rightarrow_p H[Qy (F\{x \mapsto y\} \circ G)]_p, \quad \circ \in \{\land, \lor\} \\
H[(F \circ (Qx G))]_p & \Rightarrow_p H[Qy (F \circ G\{x \mapsto y\})]_p, \quad \circ \in \{\land, \lor, \rightarrow\}
\end{align*}
\]

Here \( y \) is always assumed to be some fresh variable and \( \overline{Q} \) denotes the quantifier dual to \( Q \), i.e., \( \overline{\forall} = \exists \) and \( \overline{\exists} = \forall \).

Skolemization

Intuition: replacement of \( \exists y \) by a concrete choice function computing \( y \) from all the arguments \( y \) depends on.

Transformation \( \Rightarrow_S \)

(to be applied outermost, not in subformulas):

\[
\forall x_1, \ldots, x_n \exists y F \Rightarrow_S \forall x_1, \ldots, x_n F\{y \mapsto f(x_1, \ldots, x_n)\}
\]

where \( f/n \) is a new function symbol (Skolem function).

Together: \( F \Rightarrow_p \underbrace{G}_{\text{prenex}} \Rightarrow_S \underbrace{H}_{\text{prenex, no } \exists} \)

**Theorem 3.9** Let \( F, G, \) and \( H \) as defined above and closed. Then

(i) \( F \) and \( G \) are equivalent.

(ii) \( H \models G \) but the converse is not true in general.

(iii) \( G \) satisfiable (w.r.t. \( \Sigma\)-Alg) \( \Leftrightarrow \) \( H \) satisfiable (w.r.t. \( \Sigma'\)-Alg) where \( \Sigma' = (\Omega \cup SKF, \Pi) \) if \( \Sigma = (\Omega, \Pi) \).
The Complete Picture

\[ F \Rightarrow_p Q_1y_1 \ldots Q_ny_n G \quad (G \text{ quantifier-free}) \]
\[ \Rightarrow_S \forall x_1, \ldots, x_m H \quad (m \leq n, H \text{ quantifier-free}) \]
\[ \Rightarrow_{\text{CNF}}^* \forall x_1, \ldots, x_m \bigwedge_{i=1}^k \bigvee_{j=1}^{n_i} L_{ij} \]

leave out clauses \( C_i \)

\[ F' \]

\( N = \{C_1, \ldots, C_k\} \) is called the clausal (normal) form (CNF) of \( F \).

Note: The variables in the clauses are implicitly universally quantified.

Theorem 3.10 Let \( F \) be closed. Then \( F' \models F \). (The converse is not true in general.)

Theorem 3.11 Let \( F \) be closed. Then \( F \) is satisfiable iff \( F' \) is satisfiable iff \( N \) is satisfiable

Optimization

The normal form algorithm described so far leaves lots of room for optimization. Note that we only can preserve satisfiability anyway due to Skolemization.

- the size of the CNF is exponential when done naively; the transformations we introduced already for propositional logic avoid this exponential growth;
- we want to preserve the original formula structure;
- we want small arity of Skolem functions (see next section).
3.6 Getting Skolem Functions with Small Arity

A clause set that is better suited for automated theorem proving can be obtained using the following steps:

- eliminate trivial subformulas
- replace beneficial subformulas
- produce a negation normal form (NNF)
- apply miniscoping
- rename all variables
- Skolemize
- push quantifiers upward
- apply distributivity

We start with a closed formula.

**Elimination of Trivial Subformulas**

Eliminate subformulas $\top$ and $\bot$ essentially as in the propositional case modulo associativity/commutativity of $\wedge$, $\vee$:

\[
\begin{align*}
H[(F \land \top)]_p & \Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \lor \bot)]_p & \Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \iff \bot)]_p & \Rightarrow_{\text{OCNF}} H[\neg F]_p \\
H[(F \iff \top)]_p & \Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \lor \top)]_p & \Rightarrow_{\text{OCNF}} H[\top]_p \\
H[(F \land \bot)]_p & \Rightarrow_{\text{OCNF}} H[\bot]_p \\
H[\neg \top]_p & \Rightarrow_{\text{OCNF}} H[\bot]_p \\
H[\neg \bot]_p & \Rightarrow_{\text{OCNF}} H[\top]_p \\
H[(F \rightarrow \bot)]_p & \Rightarrow_{\text{OCNF}} H[\neg F]_p \\
H[(F \rightarrow \top)]_p & \Rightarrow_{\text{OCNF}} H[\top]_p \\
H[(\bot \rightarrow F)]_p & \Rightarrow_{\text{OCNF}} H[\top]_p \\
H[(\top \rightarrow F)]_p & \Rightarrow_{\text{OCNF}} H[F]_p \\
H[Qx \top]_p & \Rightarrow_{\text{OCNF}} H[\top]_p \\
H[Qx \bot]_p & \Rightarrow_{\text{OCNF}} H[\bot]_p
\end{align*}
\]
Replacement of Beneficial Subformulas

The functions $\nu$ and $\bar{\nu}$ that give us an overapproximation for the number of clauses generated by a formula are extended to quantified formulas by

$$\nu(\forall x \ F) = \nu(\exists x \ F) = \nu(F),$$
$$\bar{\nu}(\forall x \ F) = \bar{\nu}(\exists x \ F) = \bar{\nu}(F).$$

The other cases are defined as for propositional formulas.

Introduce top-down fresh predicates for beneficial subformulas:

$$H[F]_p \Rightarrow_{OCNF} H[P(x_1, \ldots, x_n)]_p \land \text{def}(H, p, P, F)$$

if $\nu(H[F]_p) > \nu(H[P(\ldots)]_p \land \text{def}(H, p, P, F))$,

where $\{x_1, \ldots, x_n\}$ are the free variables in $F$, $P/n$ is a predicate new to $H[F]_p$, and $\text{def}(H, p, P, F)$ is defined by

- $\forall x_1, \ldots, x_n (P(x_1, \ldots, x_n) \rightarrow F)$, if $\text{pol}(H, p) = 1$,
- $\forall x_1, \ldots, x_n (F \rightarrow P(x_1, \ldots, x_n))$, if $\text{pol}(H, p) = -1$,
- $\forall x_1, \ldots, x_n (P(x_1, \ldots, x_n) \leftrightarrow F)$, if $\text{pol}(H, p) = 0$.

As in the propositional case, one can test $\nu(H[F]_p) > \nu(H[P]_p \land \text{def}(H, p, P, F))$ in constant time without actually computing $\nu$.

Negation Normal Form (NNF)

Apply the reduction system $\Rightarrow_{\text{NNF}}$:

$$H[F \leftrightarrow G]_p \Rightarrow_{\text{NNF}} H[(F \rightarrow G) \land (G \rightarrow F)]_p$$

if $\text{pol}(H, p) = 1$ or $\text{pol}(H, p) = 0$.

$$H[F \leftrightarrow G]_p \Rightarrow_{\text{NNF}} H[(F \land G) \lor (\neg G \land \neg F)]_p$$

if $\text{pol}(H, p) = -1$.

$$H[F \rightarrow G]_p \Rightarrow_{\text{NNF}} H[\neg F \lor G]_p$$
$$H[\neg F]_p \Rightarrow_{\text{NNF}} H[F]_p$$
$$H[\neg(F \lor G)]_p \Rightarrow_{\text{NNF}} H[\neg F \land \neg G]_p$$
$$H[\neg(F \land G)]_p \Rightarrow_{\text{NNF}} H[\neg F \lor \neg G]_p$$
$$H[\neg Qx \ F]_p \Rightarrow_{\text{NNF}} H[\neg Qx \ F]_p$$
Miniscoping

Apply the reduction system $\Rightarrow_{\text{MS}}$ modulo associativity and commutativity of $\land$, $\lor$. For the rules below we assume that $x$ occurs freely in $F$, $F'$, but $x$ does not occur freely in $G$:

$$H[Qx (F \land G)]_p \Rightarrow_{\text{MS}} H[(Qx F) \land G]_p$$

$$H[Qx (F \lor G)]_p \Rightarrow_{\text{MS}} H[(Qx F) \lor G]_p$$

$$H[\forall x (F \land F')]_p \Rightarrow_{\text{MS}} H[(\forall x F) \land (\forall x F')]_p$$

$$H[\exists x (F \lor F')]_p \Rightarrow_{\text{MS}} H[(\exists x F) \lor (\exists x F')]_p$$

$$H[Qx G]_p \Rightarrow_{\text{MS}} H[G]_p$$

Variable Renaming

Rename all variables in $H$ such that there are no two different positions $p, q$ with $H|_p = Qx F$ and $H|_q = Q'x G$.

Standard Skolemization

Apply the reduction system:

$$H[\exists x F]_p \Rightarrow_{\text{SK}} H[F\{x \mapsto f(y_1, \ldots, y_n)\}]_p$$

where $p$ has minimal length,

$\{y_1, \ldots, y_n\}$ are the free variables in $\exists x F$,

and $f/n$ is a new function symbol to $H$.

Final Steps

Apply the reduction system modulo commutativity of $\land$, $\lor$ to push $\forall$ upward:

$$H[\forall x F \land G]_p \Rightarrow_{\text{OCNF}} H[\forall x (F \land G)]_p$$

$$H[\forall x F \lor G]_p \Rightarrow_{\text{OCNF}} H[\forall x (F \lor G)]_p$$

Note that variable renaming ensures that $x$ does not occur in $G$.

Apply the reduction system modulo commutativity of $\land$, $\lor$ to push disjunctions downward:

$$H[(F \land F') \lor G]_p \Rightarrow_{\text{CNF}} H[(F \lor G) \land (F' \lor G)]_p$$
3.7 Herbrand Interpretations

From now on we shall consider FOL without equality. We assume that Ω contains at least one constant symbol.

A Herbrand interpretation (over Σ) is a Σ-algebra A such that

- \( U_A = T_Σ \) (= the set of ground terms over Σ)
- \( f_A : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n), f/n \in \Omega \)

In other words, values are fixed to be ground terms and functions are fixed to be the term constructors. Only predicate symbols \( P/m \in Π \) may be freely interpreted as relations \( P_A \subseteq T_Σ^m \).

**Proposition 3.12** Every set of ground atoms I uniquely determines a Herbrand interpretation A via

\[(s_1, \ldots, s_n) \in P_A \iff P(s_1, \ldots, s_n) \in I\]

Thus we shall identify Herbrand interpretations (over Σ) with sets of Σ-ground atoms.

**Existence of Herbrand Models**

A Herbrand interpretation I is called a Herbrand model of F, if \( I \models F \).

**Theorem 3.13 (Herbrand)** Let N be a set of (universally quantified) Σ-clauses.

\[N \text{ satisfiable} \iff N \text{ has a Herbrand model (over Σ)} \iff G_Σ(N) \text{ has a Herbrand model (over Σ)}\]

where \( G_Σ(N) = \{ Cσ \text{ ground clause } | (\forall x C) \in N, \sigma : X \to T_Σ \} \) is the set of ground instances of N.

[The proof will be given below in the context of the completeness proof for general resolution.]
3.8 Inference Systems and Proofs

Inference systems $\Gamma$ (proof calculi) are sets of tuples

$$(F_1, \ldots, F_n, F_{n+1}), \ n \geq 0,$$

called inferences, and written

$$\begin{array}{c}
\text{premises} \\
\hline
F_1 \ldots F_n \\
\hline
F_{n+1} \\
\hline
\text{conclusion}
\end{array}$$

Clausal inference system: premises and conclusions are clauses. One also considers inference systems over other data structures.

Inference Systems

Inference systems $\Gamma$ are shorthands for reduction systems over sets of formulas. If $N$ is a set of formulas, then

$$\begin{array}{c}
\text{premises} \\
\hline
F_1 \ldots F_n \\
\hline
F_{n+1} \\
\hline
\text{side condition}
\end{array}$$

is a shorthand for

$$N \cup \{F_1, \ldots, F_n\} \Rightarrow_{\Gamma} N \cup \{F_1, \ldots, F_n\} \cup \{F_{n+1}\}$$

if side condition

Proofs

A proof in $\Gamma$ of a formula $F$ from a set of formulas $N$ (called assumptions) is a sequence $F_1, \ldots, F_k$ of formulas where

(i) $F_k = F$,

(ii) for all $1 \leq i \leq k$: $F_i \in N$ or there exists an inference

$$\begin{array}{c}
F_{m_1} \ldots F_{m_n} \\
\hline
F_i \\
\hline
\end{array}$$

in $\Gamma$, such that $0 \leq m_j < i$, for $1 \leq j \leq n$.
Soundness and Completeness

\textit{Provability} \vdash \Gamma \text{ of } F \text{ from } N \text{ in } \Gamma:

\[ N \vdash \Gamma F \text{ if there exists a proof in } \Gamma \text{ of } F \text{ from } N. \]

\(\Gamma\) is called \textit{sound}, if

\[ \frac{F_1 \ldots F_n}{F} \in \Gamma \text{ implies } F_1, \ldots, F_n \models F \]

\(\Gamma\) is called \textit{complete}, if

\[ N \models F \text{ implies } N \vdash \Gamma F \]

\(\Gamma\) is called \textit{refutationally complete}, if

\[ N \models \bot \text{ implies } N \vdash \Gamma \bot \]

\textbf{Proposition 3.14}

(i) Let \(\Gamma\) be sound. Then \(N \vdash \Gamma F \Rightarrow N \models F\)

(ii) If \(N \vdash \Gamma F\) then there exist finitely many \(F_1, \ldots, F_n \in N\) such that \(F_1, \ldots, F_n \vdash \Gamma F\)

\textbf{Reduced Proofs}

The definition of a proof of \(F\) given above admits sequences \(F_1, \ldots, F_k\) of formulas where some \(F_i\) are not ancestors of \(F_k = F\) (i.e., some \(F_i\) are not actually used to derive \(F\)).

A proof is called \textit{reduced}, if every \(F_i\) with \(i < k\) is an ancestor of \(F_k\).

We obtain a reduced proof from a proof by marking first \(F_k\) and then recursively all the premises used to derive a marked conclusion, and by deleting all non-marked formulas in the end.
Reduced Proofs as Trees

markings  ≜  formulas
leaves   ≜  assumptions and axioms
other nodes  ≜  inferences: conclusion  ≜  parent node
            premises  ≜  child nodes

\[
\begin{align*}
P(f(c)) \lor Q(b) & \quad \neg P(f(c)) \lor \neg P(f(c)) \lor Q(b) \\
\hline
\neg P(f(c)) \lor Q(b) & \\
\hline
P(f(c)) \lor Q(b) & \quad \neg P(f(c)) \lor Q(b) \\
\hline
Q(b) \lor Q(b) & \\
\hline
Q(b) & \\
\hline
\neg P(f(c)) & \\
\hline
\bot & \neg P(f(c))
\end{align*}
\]

3.9 Ground (or propositional) Resolution

We observe that propositional clauses and ground clauses are essentially the same, as long as we do not consider equational atoms.

In this section we only deal with ground clauses.

Unlike in Section 2 we admit duplicated literals in clauses, i.e., we treat clauses like multisets of literals, not like sets.

The Resolution Calculus \(\text{Res}\)

Resolution inference rule:

\[
D \lor A \quad C \lor \neg A \\
\hline
D \lor C
\]

Terminology: \(D \lor C\): resolvent; \(A\): resolved atom

(Positive) factorization inference rule:

\[
C \lor A \lor A \\
\hline
C \lor A
\]

These are schematic inference rules; for each substitution of the schematic variables \(C\), \(D\), and \(A\), by ground clauses and ground atoms, respectively, we obtain an inference.
We treat “∨” as associative and commutative, hence $A$ and $\neg A$ can occur anywhere in the clauses; moreover, when we write $C \lor A$, etc., this includes unit clauses, that is, $C = \bot$.

**Sample Refutation**

1. $\neg P(f(c)) \lor \neg P(f(c)) \lor Q(b)$ (given)
2. $P(f(c)) \lor Q(b)$ (given)
3. $\neg P(g(b, c)) \lor \neg Q(b)$ (given)
4. $P(g(b, c))$ (given)
5. $\neg P(f(c)) \lor Q(b) \lor Q(b)$ (Res. 2. into 1.)
6. $\neg P(f(c)) \lor Q(b)$ (Fact. 5.)
7. $Q(b) \lor Q(b)$ (Res. 2. into 6.)
8. $Q(b)$ (Fact. 7.)
9. $\neg P(g(b, c))$ (Res. 8. into 3.)
10. $\bot$ (Res. 4. into 9.)

**Soundness of Resolution**

**Theorem 3.15** Propositional resolution is sound.

**Proof.** Let $B \in \Sigma$-Alg. We have to show:

1. for resolution: $B \models D \lor A$, $B \models C \lor \neg A \Rightarrow B \models D \lor C$
2. for factorization: $B \models C \lor A \lor A \Rightarrow B \models C \lor A$

(i): Assume premises are valid in $B$. Two cases need to be considered:

   - If $B \models A$, then $B \models C$, hence $B \models D \lor C$.
   - Otherwise, $B \models \neg A$, then $B \models D$, and again $B \models D \lor C$.

(ii): Obvious. \qed

Note: In ground first-order logic we have (like in propositional logic):

1. $B \models L_1 \lor \ldots \lor L_n$ if and only if there exists $i$: $B \models L_i$.
2. $B \models A$ or $B \models \neg A$.

This does not hold for formulas with variables!
3.10 Refutational Completeness of Resolution

How to show refutational completeness of ground resolution:

- We have to show: \( N \models \bot \Rightarrow N \not\models_{Res} \bot \), or equivalently: If \( N \not\models_{Res} \bot \), then \( N \) has a model.

- Idea: Suppose that we have computed sufficiently many inferences (and not derived \( \bot \)).

- Now order the clauses in \( N \) according to some appropriate ordering, inspect the clauses in ascending order, and construct a series of Herbrand interpretations.

- The limit interpretation can be shown to be a model of \( N \).

Clause Orderings

1. We assume that \( \succ \) is any fixed ordering on ground atoms that is total and well-founded. (There exist many such orderings, e.g., the length-based ordering on atoms when these are viewed as words over a suitable alphabet.)

2. Extend \( \succ \) to an ordering \( \succ_L \) on ground literals:
   \[
   \neg A \succ_L \neg B \quad \text{if} \ A \succ B \\
   \neg A \succ_L A
   \]

3. Extend \( \succ_L \) to an ordering \( \succ_C \) on ground clauses:
   \( \succ_C = (\succ_L)_{mul} \), the multiset extension of \( \succ_L \).

   Notation: \( \succ \) also for \( \succ_L \) and \( \succ_C \).

Example

Suppose \( A_5 \succ A_4 \succ A_3 \succ A_2 \succ A_1 \succ A_0 \). Then:

\[
\begin{align*}
A_1 \lor \neg A_5 \\
\succ & \quad A_3 \lor \neg A_4 \\
\succ & \quad \neg A_1 \lor A_3 \lor A_4 \\
\succ & \quad A_1 \lor \neg A_2 \\
\succ & \quad \neg A_1 \lor A_2 \\
\succ & \quad A_1 \lor A_1 \lor A_2 \\
\succ & \quad A_0 \lor A_1
\end{align*}
\]
Properties of the Clause Ordering

Proposition 3.16

1. The orderings on literals and clauses are total and well-founded.

2. Let $C$ and $D$ be clauses with $A = \text{maxatom}(C)$, $B = \text{maxatom}(D)$, where $\text{maxatom}(C)$ denotes the maximal atom in $C$.
   
   (i) If $A \succ B$ then $C \succ D$.
   
   (ii) If $A = B$, $A$ occurs negatively in $C$ but only positively in $D$, then $C \succ D$.

Stratified Structure of Clause Sets

Let $A \succ B$. Clause sets are then stratified in this form:

$$
\begin{array}{l}
A \bigg\{ \begin{array}{l}
   \neg A \lor \ldots \\
   \ldots \lor A \lor A \\
   \ldots \lor A \\
\end{array} \\
\bigg\}
\begin{array}{l}
   \lor A \\
   \ldots \\
   \lor A \\
\end{array}
\end{array}
\begin{array}{l}
\forall \\
\ldots \\
\end{array}
\begin{array}{l}
B \bigg\{ \begin{array}{l}
   \neg B \lor \ldots \\
   \ldots \lor B \lor B \\
   \ldots \lor B \\
\end{array} \\
\bigg\}
\begin{array}{l}
   \lor B \\
   \ldots \\
   \lor B \\
\end{array}
\end{array}
$$

all clauses $C$ with $\text{maxatom}(C) = A$

all clauses $D$ with $\text{maxatom}(D) = B$

Closure of Clause Sets under $\text{Res}$

$$\text{Res}(N) = \{ C \mid C \text{ is conclusion of an inference in } \text{Res} \text{ with premises in } N \}$$

$\text{Res}^0(N) = N$

$\text{Res}^{n+1}(N) = \text{Res}(\text{Res}^n(N)) \cup \text{Res}^n(N)$, for $n \geq 0$  

$\text{Res}^*(N) = \bigcup_{n \geq 0} \text{Res}^n(N)$

$N$ is called saturated (w.r.t. resolution), if $\text{Res}(N) \subseteq N$.

Proposition 3.17

(i) $\text{Res}^*(N)$ is saturated.

(ii) $\text{Res}$ is refutationally complete, iff for each set $N$ of ground clauses:

$$N \models \bot \text{ implies } \bot \in \text{Res}^*(N)$$
Construction of Interpretations

Given: set $N$ of ground clauses, atom ordering $\succ$.
Wanted: Herbrand interpretation $I$ such that

$$I \models N \quad \text{if } N \text{ is saturated and } \bot \not\in N$$

Construction according to $\succ$, starting with the smallest clause.

Main Ideas of the Construction

- Clauses are considered in the order given by $\succ$.
- When considering $C$, one already has an interpretation so far available ($I_C$). Initially $I_C = \emptyset$.
- If $C$ is true in this interpretation, nothing needs to be changed.
- Otherwise, one would like to change the interpretation such that $C$ becomes true.
- Changes should, however, be monotone. One never deletes atoms from the interpretation, and the truth value of clauses smaller than $C$ should not change from true to false.
- Hence, one adds $\Delta_C = \{A\}$, if and only if $C$ is false in $I_C$, if $A$ occurs positively in $C$ (adding $A$ will make $C$ become true) and if this occurrence in $C$ is strictly maximal in the ordering on literals (changing the truth value of $A$ has no effect on smaller clauses). Otherwise, $\Delta_C = \emptyset$.
- We say that the construction fails for a clause $C$, if $C$ is false in $I_C$ and $\Delta_C = \emptyset$.
- We will show: If there are clauses for which the construction fails, then some inference with the smallest such clause (the so-called “minimal counterexample”) has not been computed. Otherwise, the limit interpretation is a model of all clauses.

Construction of Candidate Interpretations

Let $N, \succ$ be given. We define sets $I_C$ and $\Delta_C$ for all ground clauses $C$ over the given signature inductively over $\succ$:

$$I_C := \bigcup_{C \succ D} \Delta_D$$

$$\Delta_C := \begin{cases} 
\{A\}, & \text{if } C \in N, C = C' \lor A, A \succ C', I_C \not\models C \\
\emptyset, & \text{otherwise} 
\end{cases}$$
We say that \( C \) produces \( A \), if \( \Delta_C = \{ A \} \).

Note that the definitions satisfy the conditions of Thm. 1.8; so they are well-defined even if \( \{ D \mid C \succ D \} \) is infinite.

The candidate interpretation for \( N \) (w. r. t. \( \succ \)) is given as \( I_N^\succ := \bigcup C \Delta_C \). (We also simply write \( I_N \) or \( I \) for \( I_N^\succ \) if \( \succ \) is either irrelevant or known from the context.)

**Example**

Let \( A_5 \succ A_4 \succ A_3 \succ A_2 \succ A_1 \succ A_0 \) (max. literals in red)

<table>
<thead>
<tr>
<th>clauses ( C )</th>
<th>( I_C )</th>
<th>( \Delta_C )</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 ( \neg A_1 \lor A_5 )</td>
<td>( { A_1, A_2, A_4 } )</td>
<td>( { A_5 } )</td>
<td>max. lit. ( \neg A_4 ) neg.; \textit{min. counter-ex.}</td>
</tr>
<tr>
<td>6 ( \neg A_1 \lor A_3 \lor \neg A_4 )</td>
<td>( { A_1, A_2, A_4 } )</td>
<td>( \emptyset )</td>
<td>( A_4 ) maximal</td>
</tr>
<tr>
<td>5 ( A_0 \lor \neg A_1 \lor A_3 \lor A_4 )</td>
<td>( { A_1, A_2 } )</td>
<td>( { A_4 } )</td>
<td>( A_2 ) maximal</td>
</tr>
<tr>
<td>4 ( \neg A_1 \lor A_2 )</td>
<td>( { A_1 } )</td>
<td>( { A_2 } )</td>
<td>true in ( I_C )</td>
</tr>
<tr>
<td>3 ( A_1 \lor A_2 )</td>
<td>( { A_1 } )</td>
<td>( \emptyset )</td>
<td>( A_1 ) maximal</td>
</tr>
<tr>
<td>2 ( A_0 \lor A_1 )</td>
<td>( \emptyset )</td>
<td>( { A_1 } )</td>
<td>true in ( I_C )</td>
</tr>
<tr>
<td>1 ( \neg A_0 )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td></td>
</tr>
</tbody>
</table>

\( I = \{ A_1, A_2, A_4, A_5 \} \) is not a model of the clause set
\( \Rightarrow \) there exists a \textit{counterexample}.

**Structure of \( N, \succ \)**

Let \( A \succ B \). Note that producing a new atom does not change the truth value of smaller clauses.
Some Properties of the Construction

Proposition 3.18

(i) If \( D = D' \lor \neg A \), then no \( C \supseteq D \) produces \( A \).

(ii) If \( I_D \models D \), then \( I_C \models D \) for every \( C \supseteq D \) and \( I_N^C \models D \).

(iii) If \( D = D' \lor A \) produces \( A \), then \( I_C \models D \) for every \( C \supseteq D \) and \( I_N^C \models D \).

(iv) If \( D = D' \lor A \) produces \( A \), then \( I_C \not\models D' \) for every \( C \supseteq D \) and \( I_N^C \not\models D' \).

(v) If for every clause \( C \in N \), \( C \) is productive or \( I_C \models C \), then \( I_N^C \models N \).

Proof. (i) If \( C \) produces \( A \), then \( A \geq L \) for every literal \( L \) of \( C \). On the other hand, \( D \) contains \( \neg A \), and \( \neg A \succ A \). Since \( \neg A \succ L \) for every literal \( L \) of \( C \), we obtain \( D \succ C \).

(ii) Suppose that \( I_D \models D \) and \( C \supseteq D \). If \( I_D \models A \) for some positive literal \( A \) of \( D \), then \( A \in I_D \subseteq I_C \subseteq I_N^C \), so \( I_C \models D \) and \( I_N^C \models D \). Otherwise \( I_D \models \neg A \) for some negative literal \( \neg A \) of \( D \), hence \( A \notin I_D \). By (i), no clause that is larger than or equal to \( D \) produces \( A \), so \( A \notin I_C \) and \( A \notin I_N^C \). Again, \( I_C \models D \) and \( I_N^C \models D \).

(iii) Obvious, since \( C \supseteq D \) implies \( A \in \Delta_D \subseteq I_C \subseteq I_N^C \).

(iv) If \( D = D' \lor A \) produces \( A \), then \( A \geq L \) for every literal \( L \) of \( D' \) and \( I_D \not\models A \). Since \( I_D \not\models D \), we have \( I_D \not\models L \) for every literal \( L \) of \( D' \). Let \( C \supseteq D \). If \( L \) is a positive literal \( A' \), then \( A' \notin I_D \). Since all atoms in \( I_C \setminus I_D \) and \( I_N^C \setminus I_D \) are larger than or equal to \( A \), we get \( A' \notin I_C \) and \( A' \notin I_N^C \). Otherwise \( L \) is a negative literal \( \neg A' \), then obviously \( A' \in I_D \subseteq I_C \subseteq I_N^C \). In both cases \( L \) is false in \( I_C \) and \( I_N^C \).

(v) By (ii) and (iii).

Model Existence Theorem

Proposition 3.19 Let \( \succ \) be a clause ordering. If \( N \) is saturated w. r. t. \( \text{Res} \) and \( \bot \notin N \), then for every clause \( C \in N \), \( C \) is productive or \( I_C \models C \).

Proof. Let \( N \) be saturated w. r. t. \( \text{Res} \) and \( \bot \notin N \). Assume that the proposition does not hold. By well-foundedness, there must exist a minimal clause \( C \in N \) (w. r. t. \( \succ \)) such that \( C \) is neither productive nor \( I_C \models C \). As \( C \not\models \bot \) there exists a maximal literal in \( C \). There are two possible reasons why \( C \) is not productive:

Case 1: The maximal literal \( \neg A \) is negative, i. e., \( C = C' \lor \neg A \). Then \( I_C \models A \) and \( I_C \not\models C' \). So some \( D = D' \lor A \in N \) with \( C \succ D \) produces \( A \), and \( I_C \not\models D' \). The inference

\[
\begin{array}{c}
D' \lor A \\
\hline
D' \lor C'
\end{array}
\]

yields a clause \( D' \lor C' \in N \) that is smaller than \( C \). As \( I_C \not\models D' \lor C' \), we know that \( D' \lor C' \) is neither productive nor \( I_{D' \lor C'} \models D' \lor C' \). This contradicts the minimality of \( C \).
Case 2: The maximal literal $A$ is positive, but not strictly maximal, i.e., $C = C' \lor A \lor A$. Then there is an inference
\[
\frac{C' \lor A \lor A}{C' \lor A}
\]
that yields a smaller clause $C' \lor A \in N$. As $I_C \not\models C' \lor A$, this clause is neither productive nor $I_{C' \lor A} \models C' \lor A$. Since $C \succ C' \lor A$, this contradicts the minimality of $C$. \hfill \Box

**Theorem 3.20 (Bachmair & Ganzinger 1990)** Let $\succ$ be a clause ordering. If $N$ is saturated w. r. t. $\text{Res}$ and $\bot \not\in N$, then $I_N \models N$.

**Proof.** By Prop. 3.19 and part (v) of Prop. 3.18. \hfill \Box

**Corollary 3.21** Let $N$ be saturated w. r. t. $\text{Res}$. Then $N \models \bot$ if and only if $\bot \in N$.

**Compactness of Propositional Logic**

**Lemma 3.22** Let $N$ be a set of propositional (or first-order ground) clauses. Then $N$ is unsatisfiable, if and only if some finite subset $N' \subseteq N$ is unsatisfiable.

**Proof.** The “if” part is trivial. For the “only if” part, assume that $N$ is unsatisfiable. Consequently, $\text{Res}^*(N)$ is unsatisfiable as well. By refutational completeness of resolution, $\bot \in \text{Res}^*(N)$. So there exists an $n \geq 0$ such that $\bot \in \text{Res}^n(N)$, which means that $\bot$ has a finite resolution proof. Now choose $N'$ as the set of assumptions in this proof. \hfill \Box

**Theorem 3.23 (Compactness for Propositional Formulas)** Let $S$ be a set of propositional (or first-order ground) formulas. Then $S$ is unsatisfiable, if and only if some finite subset $S' \subseteq S$ is unsatisfiable.

**Proof.** The “if” part is again trivial. For the “only if” part, assume that $S$ is unsatisfiable. Transform $S$ into an equivalent set $N$ of clauses. By the previous lemma, $N$ has a finite unsatisfiable subset $N'$. Now choose for every clause $C$ in $N'$ one formula $F$ of $S$ such that $C$ is contained in the CNF of $F$. Let $S'$ be the set of these formulas. \hfill \Box
3.11 General Resolution

Propositional (ground) resolution:

- refutationally complete,

- in its most naive version: not guaranteed to terminate for satisfiable sets of clauses, (improved versions do terminate, however)

- inferior to the CDCL procedure.

But: in contrast to the CDCL procedure, resolution can be easily extended to non-ground clauses.

**Observation**

If $A$ is a model of an (implicitly universally quantified) clause $C$, then by Lemma 3.8 it is also a model of all (implicitly universally quantified) instances $C\sigma$ of $C$.

Consequently, if we show that some instances of clauses in a set $N$ are unsatisfiable, then we have also shown that $N$ itself is unsatisfiable.

**General Resolution through Instantiation**

Idea: instantiate clauses appropriately:

\[
\begin{align*}
P(z', z') &\lor \neg Q(z) &\neg P(a, y) & &\neg P(a, y) & &\neg P(x', b) &\lor Q(f(x', x)) \\
\{ z' \mapsto a, z \mapsto f(a, b) \} & &\{ y \mapsto a \} & &\{ y \mapsto b \} & &\{ x' \mapsto a, x \mapsto b \} \\
P(a, a) &\lor \neg Q(f(a, b)) &\neg P(a, a) & &\neg P(a, b) & &\neg P(a, b) &\lor Q(f(a, b)) \\
\neg Q(f(a, b)) & &\neg P(a, a) & &P(a, b) & &Q(f(a, b)) \\
\bot & &\neg Q(f(a, b)) & &P(a, b) & &Q(f(a, b)) \\
\end{align*}
\]
Early approaches (Gilmore 1960, Davis and Putnam 1960):

Generate ground instances of clauses.

Try to refute the set of ground instances by resolution.

If no contradiction is found, generate more ground instances.

Problems:

More than one instance of a clause can participate in a proof.

Even worse: There are infinitely many possible instances.

Observation:

Instantiation must produce complementary literals (so that inferences become possible).

Idea (Robinson 1965):

Do not instantiate more than necessary to get complementary literals
⇒ most general unifiers (mgu).

Calculus works with non-ground clauses; inferences with non-ground clauses represent
infinite sets of ground inferences which are computed simultaneously
⇒ lifting principle.

Computation of instances becomes a by-product of boolean reasoning.
**Unification**

Let \( E = \{ s_1 \doteq t_1, \ldots, s_n \doteq t_n \} \) (\( s_i, t_i \) terms or atoms) be a multiset of equality problems. A substitution \( \sigma \) is called a unifier of \( E \) if \( s_i \sigma = t_i \sigma \) for all \( 1 \leq i \leq n \).

If a unifier of \( E \) exists, then \( E \) is called unifiable.

A substitution \( \sigma \) is called more general than a substitution \( \tau \), denoted by \( \sigma \leq \tau \), if there exists a substitution \( \rho \) such that \( \rho \circ \sigma = \tau \), where \( (\rho \circ \sigma)(x) := (x \sigma)\rho \) is the composition of \( \sigma \) and \( \rho \) as mappings. (Note that \( \rho \circ \sigma \) has a finite domain as required for a substitution.)

If a unifier of \( E \) is more general than any other unifier of \( E \), then we speak of a most general unifier of \( E \), denoted by \( \text{mgu}(E) \).

**Proposition 3.24**

(i) \( \leq \) is a quasi-ordering on substitutions, and \( \circ \) is associative.

(ii) If \( \sigma \leq \tau \) and \( \tau \leq \sigma \) (we write \( \sigma \sim \tau \) in this case), then \( x \sigma \) and \( x \tau \) are equal up to (bijective) variable renaming, for any \( x \) in \( X \).

A substitution \( \sigma \) is called idempotent, if \( \sigma \circ \sigma = \sigma \).

**Proposition 3.25** \( \sigma \) is idempotent iff \( \text{dom}(\sigma) \cap \text{codom}(\sigma) = \emptyset \).

**Rule-Based Naive Standard Unification**

\[
\begin{align*}
t & \doteq t, E \quad \Rightarrow_{SU} \quad E \\
f(s_1, \ldots, s_n) & \doteq f(t_1, \ldots, t_n), E \quad \Rightarrow_{SU} \quad s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) & \doteq g(\ldots), E \quad \Rightarrow_{SU} \quad \bot \\
& \quad \text{if } f \neq g \\
x & \doteq t, E \quad \Rightarrow_{SU} \quad x \doteq t, E \{ x \mapsto t \} \\
& \quad \text{if } x \in \text{var}(E), x \notin \text{var}(t) \\
x & \doteq t, E \quad \Rightarrow_{SU} \quad \bot \\
& \quad \text{if } x \neq t, x \in \text{var}(t) \\
t & \doteq x, E \quad \Rightarrow_{SU} \quad x \doteq t, E \\
& \quad \text{if } t \notin X
\end{align*}
\]
SU: Main Properties

If $E = \{x_1 \doteq u_1, \ldots, x_k \doteq u_k\}$, with $x_i$ pairwise distinct, $x_i \not\in \text{var}(u_j)$, then $E$ is called an (equational problem in) solved form representing the solution $\sigma_E = \{x_1 \mapsto u_1, \ldots, x_k \mapsto u_k\}$.

Proposition 3.26 If $E$ is a solved form then $\sigma_E$ is an mgu of $E$.

Theorem 3.27

1. If $E \Rightarrow_{SU} E'$ then $\sigma$ is a unifier of $E$ iff $\sigma$ is a unifier of $E'$
2. If $E \Rightarrow^*_{SU} \perp$ then $E$ is not unifiable.
3. If $E \Rightarrow^*_{SU} E'$ with $E'$ in solved form, then $\sigma_{E'}$ is an mgu of $E$.

Proof. (1) We have to show this for each of the rules. Let’s treat the case for the 4th rule here. Suppose $\sigma$ is a unifier of $x \doteq t$, that is, $x\sigma = t\sigma$. Thus, $\sigma \circ \{x \mapsto t\} = \sigma[x \mapsto t\sigma] = \sigma[x \mapsto x\sigma] = \sigma$. Therefore, for any equation $u \doteq v$ in $E$: $u\sigma = v\sigma$, iff $u\{x \mapsto t\}\sigma = v\{x \mapsto t\}\sigma$. (2) and (3) follow by induction from (1) using Proposition 3.26. □

Main Unification Theorem

Theorem 3.28 $E$ is unifiable if and only if there is a most general unifier $\sigma$ of $E$, such that $\sigma$ is idempotent and $\text{dom}(\sigma) \cup \text{codom}(\sigma) \subseteq \text{var}(E)$.

Proof. The right-to-left implication is trivial. For the left-to-right implication we observe the following:

- $\Rightarrow_{SU}$ is terminating. A suitable lexicographic ordering on the multisets $E$ (with $\perp$ minimal) shows this. Compare in this order:
  (1) the number of variables that occur in $E$ below a function or predicate symbol, or on the right-hand side of an equation, or at least twice;
  (2) the multiset of the sizes (numbers of symbols) of all equations in $E$;
  (3) the number of non-variable left-hand sides of equations in $E$.
- A system $E$ that is irreducible w.r.t. $\Rightarrow_{SU}$ is either $\perp$ or a solved form.
- Therefore, reducing any $E$ by SU will end (no matter what reduction strategy we apply) in an irreducible $E'$ having the same unifiers as $E$, and we can read off the mgu (or non-unifiability) of $E$ from $E'$ (Theorem 3.27, Proposition 3.26).
- $\sigma$ is idempotent because of the substitution in rule 4. $\text{dom}(\sigma) \cup \text{codom}(\sigma) \subseteq \text{var}(E)$, as no new variables are generated. □
Rule-Based Polynomial Unification

Problem: using $\Rightarrow_{SU}$, an exponential growth of terms is possible.

The following unification algorithm avoids this problem, at least if the final solved form is represented as a DAG.

\[
t \doteq t, \ E \ \Rightarrow_{PU} \ E
\]

\[
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), \ E \ \Rightarrow_{PU} s_1 \doteq t_1, \ldots, s_n \doteq t_n, \ E
\]

\[
f(\ldots) \doteq g(\ldots), \ E \ \Rightarrow_{PU} \bot
\quad \text{if } f \neq g
\]

\[
x \doteq y, \ E \ \Rightarrow_{PU} x \doteq y, \ E\{x \mapsto y\}
\quad \text{if } x \in \text{var}(E), x \neq y
\]

\[
x_1 \doteq t_1, \ldots, x_n \doteq t_n, \ E \ \Rightarrow_{PU} \bot
\quad \text{if there are positions } p_i \text{ with}
\quad t_i|_{p_i} = x_{i+1}, t_n|_{p_n} = x_1
\quad \text{and some } p_i \neq \varepsilon
\]

\[
x \doteq t, \ E \ \Rightarrow_{PU} \bot
\quad \text{if } x \neq t, x \in \text{var}(t)
\]

\[
t \doteq x, \ E \ \Rightarrow_{PU} x \doteq t, E
\quad \text{if } t \notin X
\]

\[
x \doteq t, x \doteq s, \ E \ \Rightarrow_{PU} x \doteq t, t \doteq s, E
\quad \text{if } t, s \notin X \text{ and } |t| \leq |s|
\]

Properties of PU

Theorem 3.29

1. If $E \Rightarrow_{PU} E'$ then $\sigma$ is a unifier of $E$ iff $\sigma$ is a unifier of $E'$

2. If $E \Rightarrow_{PU}^* \bot$ then $E$ is not unifiable.

3. If $E \Rightarrow_{PU}^* E'$ with $E'$ in solved form, then $\sigma_{E'}$ is an mgu of $E$.

Note: The solved form of $\Rightarrow_{PU}$ is different from the solved form obtained from $\Rightarrow_{SU}$. In order to obtain the unifier $\sigma_{E'}$, we have to sort the list of equality problems $x_i \doteq t_i$ in such a way that $x_i$ does not occur in $t_j$ for $j < i$, and then we have to compose the substitutions $\{x_1 \mapsto t_1\} \circ \cdots \circ \{x_k \mapsto t_k\}$.

76
Resolution for General Clauses

We obtain the resolution inference rules for non-ground clauses from the inference rules for ground clauses by replacing equality by unifiability:

General resolution $\text{Res}$:

$$
\frac{D \lor B \quad C \lor \neg A}{(D \lor C) \sigma} \quad \text{if } \sigma = \text{mgu}(A, B) \quad \text{[resolution]}
$$

$$
\frac{C \lor A \lor B}{(C \lor A) \sigma} \quad \text{if } \sigma = \text{mgu}(A, B) \quad \text{[factorization]}
$$

For inferences with more than one premise, we assume that the variables in the premises are (bijectively) renamed such that they become different to any variable in the other premises. We do not formalize this. Which names one uses for variables is otherwise irrelevant.

Lifting Lemma

Lemma 3.30 Let $C$ and $D$ be variable-disjoint clauses. If

$$
\begin{array}{c}
D \\
\downarrow \theta_1 \\
D\theta_1
\end{array}
\qquad
\begin{array}{c}
C \\
\downarrow \theta_2 \\
C\theta_2
\end{array}

\Rightarrow
\begin{array}{c}
C'
\end{array}
\quad \text{[ground resolution]}
$$

then there exists a substitution $\rho$ such that

$$
\begin{array}{c}
D \\
\downarrow \rho \\
C' = C'' \rho
\end{array}
\quad \text{[general resolution]}
$$

An analogous lifting lemma holds for factorization.
Saturation of Sets of General Clauses

**Corollary 3.31** Let $N$ be a set of general clauses saturated under $\text{Res}$, i.e., $\text{Res}(N) \subseteq N$. Then also $G_\Sigma(N)$ is saturated, that is,

$$\text{Res}(G_\Sigma(N)) \subseteq G_\Sigma(N).$$

**Proof.** W.l.o.g. we may assume that clauses in $N$ are pairwise variable-disjoint. (Otherwise make them disjoint, and this renaming process changes neither $\text{Res}(N)$ nor $G_\Sigma(N)$.)

Let $C' \in \text{Res}(G_\Sigma(N))$. Then either (i) there exist resolvable ground instances $D\theta_1$ and $C\theta_2$ of $N$ with resolvent $C'$, or else (ii) $C'$ is a factor of a ground instance $C\theta$ of $C$.

Case (i): By the Lifting Lemma, $D$ and $C$ are resolvable with a resolvent $C''$ with $C'' \rho = C'$, for a suitable substitution $\rho$. As $C'' \in N$ by assumption, we obtain that $C' \in G_\Sigma(N)$.

Case (ii): Similar. $\square$

**Soundness for General Clauses**

**Proposition 3.32** The general resolution calculus is sound.

**Proof.** We have to show that, if $\sigma = \text{mgu}(A, B)$ then $\{\forall \bar{x} (D \lor B), \forall \bar{y} (C \lor \neg A)\} \models \forall \bar{z} (D \lor C)\sigma$ and $\{\forall \bar{x} (C \lor A \lor B)\} \models \forall \bar{z} (C \lor A)\sigma$.

Let $\mathcal{A}$ be a model of $\forall \bar{x} (D \lor B)$ and $\forall \bar{y} (C \lor \neg A)$. By Lemma 3.8, $\mathcal{A}$ is also a model of $\forall \bar{z} (D \lor B)\sigma$ and $\forall \bar{z} (C \lor \neg A)\sigma$ and by Lemma 3.7, $\mathcal{A}$ is also a model of $(D \lor B)\sigma$ and $(C \lor \neg A)\sigma$. Let $\beta$ be an assignment. If $\mathcal{A}(\beta)(B\sigma) = 0$, then $\mathcal{A}(\beta)(D\sigma) = 1$. Otherwise $\mathcal{A}(\beta)(B\sigma) = \mathcal{A}(\beta)(A\sigma) = 1$, hence $\mathcal{A}(\beta)(\neg A\sigma) = 0$ and therefore $\mathcal{A}(\beta)(C\sigma) = 1$. In both cases $\mathcal{A}(\beta)((D \lor C)\sigma) = 1$, so $\mathcal{A} \models (D \lor C)\sigma$ and by Lemma 3.7, $\mathcal{A} \models \forall \bar{z} (D \lor C)\sigma$.

The proof for factorization inferences is similar. $\square$
Herbrand’s Theorem

Lemma 3.33 Let $N$ be a set of $\Sigma$-clauses, let $A$ be an interpretation. Then $A \models N$ implies $A \models G_\Sigma(N)$.

Lemma 3.34 Let $N$ be a set of $\Sigma$-clauses, let $A$ be a Herbrand interpretation. Then $A \models G_\Sigma(N)$ implies $A \models N$.

Proof. Let $A$ be a Herbrand model of $G_\Sigma(N)$. We have to show that $A \models \forall \bar{x} \ C$ for all clauses $\forall \bar{x} \ C$ in $N$. This is equivalent to $A \models C$, which in turn is equivalent to $A(\beta)(C) = 1$ for all assignments $\beta$.

Choose $\beta : X \to U_A$ arbitrarily. Since $A$ is a Herbrand interpretation, $\beta(x)$ is a ground term for every variable $x$, so there is a substitution $\sigma$ such that $x\sigma = \beta(x)$ for all variables $x$ occurring in $C$. Now let $\gamma$ be an arbitrary assignment, then for every variable occurring in $C$ we have $(\gamma \circ \sigma)(x) = A(\gamma)(x\sigma) = x\sigma = \beta(x)$ and consequently $A(\beta)(C) = A(\gamma)(C\sigma) = A(\gamma)(C\sigma)$. Since $C\sigma \in G_\Sigma(N)$ and $A$ is a Herbrand model of $G_\Sigma(N)$, we get $A(\gamma)(C\sigma) = 1$, so $A$ is a model of $C$. $\square$

Theorem 3.35 (Herbrand) A set $N$ of $\Sigma$-clauses is satisfiable if and only if it has a Herbrand model over $\Sigma$.

Proof. The “$\Leftarrow$” part is trivial. For the “$\Rightarrow$” part let $N \not\models \bot$. Since resolution is sound, this implies that $\bot \notin Res^*(N)$. Obviously, a ground instance of a clause has the same number of literals as the clause itself, so we can conclude that $\bot \notin G_\Sigma(Res^*(N))$. Since $Res^*(N)$ is saturated, $G_\Sigma(Res^*(N))$ is saturated as well by Cor. 3.31. Now $I_{G_\Sigma(Res^*(N))}$ is a Herbrand interpretation over $\Sigma$ and by Thm. 3.20 it is a model of $G_\Sigma(Res^*(N))$. By Lemma 3.34, every Herbrand model of $G_\Sigma(Res^*(N))$ is a model of $Res^*(N)$. Now $N \subseteq Res^*(N)$, so $I_{G_\Sigma(Res^*(N))} \models N$. $\square$

Corollary 3.36 A set $N$ of $\Sigma$-clauses is satisfiable if and only if its set of ground instances $G_\Sigma(N)$ is satisfiable.

Proof. The “$\Rightarrow$” part follows directly from Lemma 3.33. For the “$\Leftarrow$” part assume that $G_\Sigma(N)$ is satisfiable. By Thm. 3.35 $G_\Sigma(N)$ has a Herbrand model. By Lemma 3.34, every Herbrand model of $G_\Sigma(N)$ is a model of $N$. $\square$
Refutational Completeness of General Resolution

**Theorem 3.37** Let $N$ be a set of general clauses that is saturated w.r.t. $Res$. Then $N \models \bot$ if and only if $\bot \in N$.

**Proof.** The “$\Leftarrow$” part is trivial. For the “$\Rightarrow$” part assume that $N$ is saturated, that is, $Res(N) \subseteq N$. By Corollary 3.31, $G_\Sigma(N)$ is saturated as well, i.e., $Res(G_\Sigma(N)) \subseteq G_\Sigma(N)$. By Cor. 3.36, $N \models \bot$ implies $G_\Sigma(N) \models \bot$. By the refutational completeness of ground resolution, $G_\Sigma(N) \models \bot$ implies $\bot \in G_\Sigma(N)$, so $\bot \in N$. 

3.12 Theoretical Consequences

We get some classical results on properties of first-order logic as easy corollaries.

The Theorem of Löwenheim-Skolem

**Theorem 3.38** (Löwenheim–Skolem) Let $\Sigma$ be a countable signature and let $S$ be a set of closed $\Sigma$-formulas. Then $S$ is satisfiable iff $S$ has a model over a countable universe.

**Proof.** If both $X$ and $\Sigma$ are countable, then $S$ can be at most countably infinite. Now generate, maintaining satisfiability, a set $N$ of clauses from $S$. This extends $\Sigma$ by at most countably many new Skolem functions to $\Sigma'$. As $\Sigma'$ is countable, so is $T_{\Sigma'}$, the universe of Herbrand-interpretations over $\Sigma'$. Now apply Theorem 3.35.

There exist more refined versions of this theorem. For instance, one can show that, if $S$ has some infinite model, then $S$ has a model with a universe of cardinality $\kappa$ for every $\kappa$ that is larger than or equal to the cardinalty of the signature $\Sigma$.

Compactness of Predicate Logic

**Theorem 3.39** (Compactness Theorem for First-Order Logic) Let $S$ be a set of closed first-order formulas. $S$ is unsatisfiable $\iff$ some finite subset $S' \subseteq S$ is unsatisfiable.

**Proof.** The “$\Leftarrow$” part is trivial. For the “$\Rightarrow$” part let $S$ be unsatisfiable and let $N$ be the set of clauses obtained by Skolemization and CNF transformation of the formulas in $S$. Clearly $Res^\ast(N)$ is unsatisfiable. By Theorem 3.37, $\bot \in Res^\ast(N)$, and therefore $\bot \in Res^\ast(N)$ for some $n \in \mathbb{N}$. Consequently, $\bot$ has a finite resolution proof $B$ of depth $\leq n$. Choose $S'$ as the subset of formulas in $S$ such that the corresponding clauses contain the assumptions (leaves) of $B$. 

80
3.13 Ordered Resolution with Selection

Motivation: Search space for Res very large.

Ideas for improvement:

1. In the completeness proof (Model Existence Theorem 3.20) one only needs to resolve and factor maximal atoms
   ⇒ if the calculus is restricted to inferences involving maximal atoms, the proof remains correct
   ⇒ ordering restrictions

2. In the proof, it does not really matter with which negative literal an inference is performed
   ⇒ choose a negative literal don’t-care-nondeterministically
   ⇒ selection

Ordering Restrictions

In the completeness proof one only needs to resolve and factor maximal atoms
⇒ If we impose ordering restrictions on ground inferences, the proof remains correct:

*(Ground) Ordered Resolution:*

\[
\begin{array}{c}
D \lor A \\
C \lor \neg A
\end{array} \quad \Rightarrow \quad \begin{array}{c}
D \lor C
\end{array}
\]

if \( A \succ L \) for all \( L \) in \( D \) and \( \neg A \succeq L \) for all \( L \) in \( C \).

*(Ground) Ordered Factorization:*

\[
\begin{array}{c}
C \lor A \lor A
\end{array} \quad \Rightarrow \quad \begin{array}{c}
C \lor A
\end{array}
\]

if \( A \succeq L \) for all \( L \) in \( C \).

Problem: How to extend this to non-ground inferences?

In the completeness proof, we talk about (strictly) maximal literals of ground clauses.

In the non-ground calculus, we have to consider those literals that correspond to (strictly) maximal literals of ground instances.
An ordering \( \succ \) on atoms (or terms) is called stable under substitutions, if \( A \succ B \) implies \( A\sigma \succ B\sigma \).

Note:
- We can not require that \( A \succ B \) iff \( A\sigma \succ B\sigma \).
- We can not require that \( \succ \) is total on non-ground atoms.

Consequence: In the ordering restrictions for non-ground inferences, we have to replace \( \succ \) by \( \not\preceq \) and \( \preceq \) by \( \not\prec \).

**Ordered Resolution:**

\[
\begin{array}{c}
D \lor B \\
C \lor \neg A
\end{array}
\]

if \( \sigma = \text{mgu}(A, B) \) and \( B\sigma \not\preceq L\sigma \) for all \( L \) in \( D \) and \( \neg A\sigma \not\prec L\sigma \) for all \( L \) in \( C \).

**Ordered Factorization:**

\[
\begin{array}{c}
C \lor A \lor B
\end{array}
\]

if \( \sigma = \text{mgu}(A, B) \) and \( A\sigma \not\prec L\sigma \) for all \( L \) in \( C \).

**Selection Functions**

Selection functions can be used to override ordering restrictions for individual clauses.

A selection function is a mapping

\[ \text{sel} : C \rightarrow \text{set of occurrences of negative literals in } C \]

Example of selection with selected literals indicated as \( \boxed{X} \):

\[
\begin{array}{c}
\neg A \lor \neg A \lor B \\
\neg B_0 \lor \neg B_1 \lor A
\end{array}
\]

Intuition:
- If a clause has at least one selected literal, compute only inferences that involve a selected literal.
- If a clause has no selected literals, compute only inferences that involve a maximal literal.
Resolution Calculus $\text{Res}_{\text{sel}}^>$

The resolution calculus $\text{Res}_{\text{sel}}^>$ is parameterized by

- a selection function sel
- and a well-founded ordering $\succ$ on atoms that is total on ground atoms and stable under substitutions.

*(Ground) Ordered Resolution with Selection:*

\[
\frac{D \lor A \quad C \lor \neg A}{D \lor C}
\]

if the following conditions are satisfied:

(i) $A \succ L$ for all $L$ in $D$;
(ii) nothing is selected in $D \lor A$ by sel;
(iii) $\neg A$ is selected in $C \lor \neg A$, or nothing is selected in $C \lor \neg A$ and $\neg A \succeq L$ for all $L$ in $C$.

*(Ground) Ordered Factorization with Selection:*

\[
\frac{C \lor A \lor A}{C \lor A}
\]

if the following conditions are satisfied:

(i) $A \succeq L$ for all $L$ in $C$;
(ii) nothing is selected in $C \lor A \lor A$ by sel.
The extension from ground inferences to non-ground inferences is analogous to ordered resolution (replace $\succ$ by $\preceq$ and $\succeq$ by $\preceq$). Again we assume that $\succ$ is stable under substitutions.

Ordered Resolution with Selection:

\[
\frac{D \lor B \quad C \lor \lnot A}{(D \lor C)\sigma}
\]

if the following conditions are satisfied:

(i) $\sigma = \text{mgu}(A, B)$;
(ii) $B\sigma \not\preceq L\sigma$ for all $L$ in $D$;
(iii) nothing is selected in $D \lor B$ by sel;
(iv) $\lnot A$ is selected in $C \lor \lnot A$, or nothing is selected in $C \lor \lnot A$ and $\lnot A\sigma \not\preceq L\sigma$ for all $L$ in $C$.

Ordered Factorization with Selection:

\[
\frac{C \lor A \lor B}{(C \lor A)\sigma}
\]

if the following conditions are satisfied:

(i) $\sigma = \text{mgu}(A, B)$;
(ii) $A\sigma \not\preceq L\sigma$ for all $L$ in $C$.
(iii) nothing is selected in $C \lor A \lor B$ by sel.
Lifting Lemma for $Res_{sel}$

Lemma 3.40 Let $C$ and $D$ be variable-disjoint clauses. If

\[
\begin{array}{ccl}
D & \xrightarrow{\theta_1} & C \\
\downarrow & & \downarrow \\
D\theta_1 & \equiv & C\theta_2 \\
\end{array}
\]

[ground inference in $Res_{sel}$]

and if $sel(D\theta_1) \simeq sel(D)$, $sel(C\theta_2) \simeq sel(C)$ (that is, “corresponding” literals are selected), then there exists a substitution $\rho$ such that

\[
\begin{array}{ccl}
D & C & C'' \\
\downarrow & \equiv & \downarrow \\
C' & \equiv & C''\rho
\end{array}
\]

[inference in $Res_{sel}$]

An analogous lifting lemma holds for factorization.

Saturation of Sets of General Clauses

Corollary 3.41 Let $N$ be a set of general clauses saturated under $Res_{sel}$, i.e., $Res_{sel}(N) \subseteq N$. Then there exists a selection function $sel'$ such that $sel|_N = sel'|_N$ and $G_{(N)}$ is also saturated, i.e.,

\[Res_{sel}' (G_{(N)}) \subseteq G_{(N)} \]

Proof. We first define the selection function $sel'$ such that $sel'(C) = sel(C)$ for all clauses $C \in G_{(N)} \cap N$. For $C \in G_{(N)} \setminus N$ we choose a fixed but arbitrary clause $D \in N$ with $C \in G_{(D)}$ and define $sel'(C)$ to be those occurrences of literals that are ground instances of the occurrences selected by $sel$ in $D$. Then proceed as in the proof of Cor. 3.31 using the lifting lemma above. \qed
Soundness and Refutational Completeness

**Theorem 3.42** Let \( \succ \) be an atom ordering and sel a selection function such that \( \text{Res}^\succ_{\text{sel}}(N) \subseteq N \). Then

\[ N \models \bot \iff \bot \in N \]

**Proof.** The “\( \Leftarrow \)” part is trivial. For the “\( \Rightarrow \)” part consider first the propositional level: Construct a candidate interpretation \( I_N \) as for unrestricted resolution, except that clauses \( C \) in \( N \) that have selected literals are never productive (even if they are false in \( I_C \) and if their maximal atom occurs only once and is positive). The result for general clauses follows using Corollary 3.41. \( \Box \)

**What Do We Gain?**

Search spaces become smaller:

1. \( P \lor Q \)
2. \( P \lor \neg Q \)
3. \( \neg P \lor Q \)
4. \( \neg P \lor \neg Q \)
5. \( Q \lor Q \) Res 1, 3
6. \( Q \) Fact 5
7. \( \neg P \) Res 6, 4
8. \( P \) Res 6, 2
9. \( \bot \) Res 8, 7

In this example, the ordering and selection function even ensure that the refutation proceeds strictly deterministically.

Rotation redundancy can be avoided:

From

\[
\begin{array}{c}
C_1 \lor A \\
C_2 \lor \neg A \lor B
\end{array}
\rightarrow
\begin{array}{c}
C_1 \lor C_2 \lor B \\
C_1 \lor C_2 \lor C_3
\end{array}
\]

we can obtain by rotation

\[
\begin{array}{c}
C_1 \lor A \\
C_2 \lor \neg A \lor B
\end{array}
\rightarrow
\begin{array}{c}
C_2 \lor \neg A \lor C_3 \\
C_1 \lor C_2 \lor C_3
\end{array}
\]

another proof of the same clause. In large proofs many rotations are possible. However, if \( A \succ B \), then the second proof does not fulfill the ordering restrictions.
Craig-Interpolation

**Theorem 3.43 (Craig 1957)** Let \( F \) and \( G \) be two propositional formulas such that \( F \models G \). Then there exists a formula \( H \) (called the interpolant for \( F \models G \)), such that \( H \) contains only propositional variables occurring both in \( F \) and in \( G \), and such that \( F \models H \) and \( H \models G \).

**Proof.** Let \( \Pi_F \), \( \Pi_G \), and \( \Pi_{FG} \) be the sets of propositional variables that occur only in \( F \), only in \( G \), or both in \( F \) and \( G \). Translate \( F \) and \( \neg G \) into CNF; let \( N \) and \( M \), respectively, denote the resulting clause set. Choose an atom ordering \( \succ \) for which the propositional variables in \( \Pi_F \) are larger than those in \( \Pi_{FG} \cup \Pi_G \). Saturate \( N \) into \( N' \) w.r.t. \( Res_{\succ}^{sel} \) with an empty selection function \( sel \). Then saturate \( N' \cup M \) w.r.t. \( Res_{\succ}^{sel} \) to derive \( \bot \). As \( N' \) is already saturated, due to the ordering restrictions only inferences need to be considered where premises, if they are from \( N' \), only contain symbols from \( \Pi_{FG} \). The conjunction of these premises is an interpolant \( H \).

The theorem also holds for first-order formulas, but in the general case, a proof based on resolution technology is complicated because of Skolemization.

### 3.14 Redundancy

So far: local restrictions of the resolution inference rules using orderings and selection functions.

Is it also possible to delete clauses altogether? Under which circumstances are clauses unnecessary? (e.g., if they are tautologies)

Intuition: If a clause is guaranteed to be neither a minimal counterexample nor productive, then we do not need it.

**A Formal Notion of Redundancy**

Let \( N \) be a set of ground clauses and \( C \) a ground clause (not necessarily in \( N \)). \( C \) is called **redundant** w.r.t. \( N \), if there exist \( C_1, \ldots, C_n \in N \), \( n \geq 0 \), such that \( C_i \prec C \) and \( C_1, \ldots, C_n \models C \).

Redundancy for general clauses: \( C \) is called **redundant** w.r.t. \( N \), if all ground instances \( C\sigma \) of \( C \) are redundant w.r.t. \( G_\Sigma(N) \).

Intuition: If a ground clause \( C \) is redundant and all clauses smaller than \( C \) hold in \( I_C \), then \( C \) holds in \( I_C \) (so \( C \) is neither a minimal counterexample nor productive).

Note: The same ordering \( \succ \) is used for ordering restrictions and for redundancy (and for the completeness proof).
Examples of Redundancy

In general, redundancy is undecidable. Decidable approximations are sufficient for us, however.

**Proposition 3.44** Some redundancy criteria:

- $C$ tautology (i.e., $\models C$) $\Rightarrow$ $C$ redundant w. r. t. any set $N$.
- $C \sigma \subset D$ $\Rightarrow$ $D$ redundant w. r. t. $N \cup \{C\}$.

(Under certain conditions one may also use non-strict subsumption, but this requires a slightly more complicated definition of redundancy.)

**Saturation up to Redundancy**

$N$ is called saturated up to redundancy (w. r. t. Res$_{sel}^\succ$) if

$$Res_{sel}^\succ(N \setminus Red(N)) \subseteq N \cup Red(N)$$

**Theorem 3.45** Let $N$ be saturated up to redundancy. Then

$$N \models \bot \iff \bot \in N$$

**Proof (Sketch).**

(i) Ground case: Consider the construction of the candidate interpretation $I_N^\succ$ for Res$_{sel}^\succ$.

If a clause $C \in N$ is redundant, then there exist $C_1, \ldots, C_n \in N$, $n \geq 0$, such that $C_i \prec C$ and $C_1, \ldots, C_n \models C$.

If $I_C \models C$, by minimality, then $I_C \models C$.

In particular, $C$ is not productive.

$\Rightarrow$ Redundant clauses are not used as premises for “essential” inferences.

By saturation, the conclusion $D' \lor C'$ of a resolution inference is contained in $N$ (as before) or in $Red(N)$. In the first case, minimality of $C$ ensures that $D' \lor C'$ is productive or $I_{D'\lor C'} \models D' \lor C'$; in the second case, it ensures that $I_{D'\lor C'} \models D' \lor C'$.

So in both cases we get a contradiction (analogously for factorization). The rest of the proof works as before.

(ii) Lifting: no additional problems over the proof of Theorem 3.42.
Monotonicity Properties of Redundancy

When we want to delete redundant clauses during a derivation, we have to ensure that redundant clauses remain redundant in the rest of the derivation.

Theorem 3.46

(i) \( N \subseteq M \Rightarrow \text{Red}(N) \subseteq \text{Red}(M) \)

(ii) \( M \subseteq \text{Red}(N) \Rightarrow \text{Red}(N) \subseteq \text{Red}(N \setminus M) \)

Proof. (i) Obvious.

(ii) For ground clause sets \( N \), the well-foundedness of the multiset extension of the clause ordering implies that every clause in \( \text{Red}(N) \) is entailed by smaller clauses in \( N \) that are themselves not in \( \text{Red}(N) \).

For general clause sets \( N \), the result follows from the fact that every clause in \( G^\Sigma(N) \setminus \text{Red}(G^\Sigma(N)) \) is an instance of a clause in \( N \setminus \text{Red}(N) \). \( \square \)

Recall that \( \text{Red}(N) \) may include clauses that are not in \( N \).

Computing Saturated Sets

Redundancy is preserved when, during a theorem proving derivation one adds new clauses or deletes redundant clauses. This motivates the following definitions:

A run of the resolution calculus is a sequence \( N_0 \vdash N_1 \vdash N_2 \vdash \ldots \), such that

(i) \( N_i \mid= N_{i+1} \), and

(ii) all clauses in \( N_i \setminus N_{i+1} \) are redundant w.r.t. \( N_{i+1} \).

In other words, during a run we may add a new clause if it follows from the old ones, and we may delete a clause, if it is redundant w.r.t. the remaining ones.

For a run, we define \( N_\infty = \bigcup_{i \geq 0} N_i \) and \( N_* = \bigcap_{i \geq 0} \bigcap_{j \geq i} N_j \). The set \( N_* \) of all persistent clauses is called the limit of the run.

Lemma 3.47 Let \( N_0 \vdash N_1 \vdash N_2 \vdash \ldots \) be a run. Then \( \text{Red}(N_i) \subseteq \text{Red}(N_\infty) \) and \( \text{Red}(N_i) \subseteq \text{Red}(N_*) \) for every \( i \).

Proof. Exercise. \( \square \)

Corollary 3.48 \( N_i \subseteq N_* \cup \text{Red}(N_*) \) for every \( i \).

Proof. If \( C \in N_i \setminus N_* \), then there is a \( k \geq i \) such that \( C \in N_k \setminus N_{k+1} \), so \( C \) must be redundant w.r.t. \( N_{k+1} \). Consequently, \( C \) is redundant w.r.t. \( N_* \). \( \square \)
Even if a set $N$ is inconsistent, it could happen that $\bot$ is never derived, because some required inference is never computed.

The following definition rules out such runs:

A run is called fair, if the conclusion of every inference from clauses in $N_* \setminus \text{Red}(N_*)$ is contained in some $N_i \cup \text{Red}(N_i)$.

**Lemma 3.49** If a run is fair, then its limit is saturated up to redundancy.

**Proof.** If the run is fair, then the conclusion of every inference from non-redundant clauses in $N_*$ is contained in some $N_i \cup \text{Red}(N_i)$, and therefore contained in $N_* \cup \text{Red}(N_*)$. Hence $N_*$ is saturated up to redundancy. $\square$

**Theorem 3.50** (Refutational Completeness: Dynamic View) Let $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$ be a fair run, let $N_*$ be its limit. Then $N_0$ has a model if and only if $\bot \notin N_*$. 

**Proof.** ($\Leftarrow$): By fairness, $N_*$ is saturated up to redundancy. If $\bot \notin N_*$, then it has a Herbrand model. Since every clause in $N_0$ is contained in $N_*$ or redundant w.r.t. $N_*$, this model is also a model of $G_\Sigma(N_0)$ and therefore a model of $N_0$.

($\Rightarrow$): Obvious, since $N_0 \models N_*$. $\square$

**Simplifications**

In theory, the definition of a run permits to add arbitrary clauses that are entailed by the current ones.

In practice, we restrict to two cases:

- We add conclusions of $Res_{sel}$-inferences from non-redundant premises. $\sim$ necessary to guarantee fairness
- We add clauses that are entailed by the current ones if this makes other clauses redundant:

  $N \cup \{C\} \vdash N \cup \{C, D\} \vdash N \cup \{D\}$

  if $N \cup \{C\} \models D$ and $C \in \text{Red}(N \cup \{D\})$.

Net effect: $C$ is simplified to $D$

$\sim$ useful to get easier/smaller clause sets

90
Examples of simplification techniques:

- Deletion of duplicated literals:
  \[ N \cup \{ C \lor L \lor L \} \vdash N \cup \{ C \lor L \} \]

- Subsumption resolution:
  \[ N \cup \{ D \lor L, \; C \lor D\sigma \lor \overline{L}\sigma \} \vdash N \cup \{ D \lor L, \; C \lor D\sigma \} \]

3.15 Hyperresolution

There are many variants of resolution. One well-known example is hyperresolution (Robinson 1965):

Assume that several negative literals are selected in a clause \( C \). If we perform an inference with \( C \), then one of the selected literals is eliminated.

Suppose that the remaining selected literals of \( C \) are again selected in the conclusion. Then we must eliminate the remaining selected literals one by one by further resolution steps.

Hyperresolution replaces these successive steps by a single inference. As for \( Res_{\text{sel}} \), the calculus is parameterized by an atom ordering \( \succ \) and a selection function \( \text{sel} \).

\[
\frac{D_1 \lor B_1 \ldots \lor D_n \lor B_n \quad C \lor \neg A_1 \lor \ldots \lor \neg A_n}{(D_1 \lor \ldots \lor D_n \lor C)\sigma}
\]

with \( \sigma = \text{mgu}(A_1 \equiv B_1, \ldots, A_n \equiv B_n) \), if

(i) \( B_i\sigma \) strictly maximal in \( D_i\sigma \), \( 1 \leq i \leq n \);

(ii) nothing is selected in \( D_i \);

(iii) the indicated occurrences of the \( \neg A_i \) are exactly the ones selected by \( \text{sel} \), or nothing is selected in the right premise and \( n = 1 \) and \( \neg A_1\sigma \) is maximal in \( C\sigma \).

Similarly to resolution, hyperresolution has to be complemented by a factorization inference.

As we have seen, hyperresolution can be simulated by iterated binary resolution. However this yields intermediate clauses which HR might not derive, and many of them might not be extendable into a full HR inference.
3.16 Implementing Resolution: The Main Loop

Standard approach:

Select one clause (“Given clause”).

Find many partner clauses that can be used in inferences together with the “given clause” using an appropriate index data structure.

Compute the conclusions of these inferences; add them to the set of clauses.

The set of clauses is split into two subsets:

- \( WO \) = “Worked-off” (or “active”) clauses: Have already been selected as “given clause”.
- \( U \) = “Usable” (or “passive”) clauses: Have not yet been selected as “given clause”.

During each iteration of the main loop:

Select a new given clause \( C \) from \( U \);
\[ U := U \setminus \{C\}. \]

Find partner clauses \( D_i \) from \( WO \);
\[ \text{New} := \text{Conclusions of inferences from } \{D_i \mid i \in I\} \cup C \text{ where one premise is } C;\]
\[ U := U \cup \text{New}; \]
\[ WO := WO \cup \{C\} \]

\[ \Rightarrow \text{At any time, all inferences between clauses in } WO \text{ have been computed.} \]
\[ \Rightarrow \text{The procedure is fair, if no clause remains in } U \text{ forever.} \]

Additionally:

Try to simplify \( C \) using \( WO \). (Skip the remainder of the iteration, if \( C \) can be eliminated.)

Try to simplify (or even eliminate) clauses from \( WO \) using \( C \).

Design decision: should one also simplify \( U \) using \( C \)?

yes \( \sim \) “Otter loop”:
Advantage: simplifications of \( U \) may be useful to derive the empty clause.

no \( \sim \) “Discount loop”:
Advantage: clauses in \( U \) are really passive; only clauses in \( WO \) have to be kept in index data structure. (Hence: can use index data structure for which retrieval is faster, even if update is slower and space consumption is higher.)
3.17 Summary: Resolution Theorem Proving

- Resolution is a machine calculus.
- Subtle interleaving of enumerating instances and proving inconsistency through the use of unification.
- Parameters: atom ordering $\succ$ and selection function sel. On the non-ground level, ordering constraints can (only) be solved approximatively.
- Completeness proof by constructing candidate interpretations from productive clauses $C \lor A, A \succ C$.
- Local restrictions of inferences via $\succ$ and sel $\Rightarrow$ fewer proof variants.
- Global restrictions of the search space via elimination of redundancy $\Rightarrow$ computing with “smaller”/“easier” clause sets; $\Rightarrow$ termination on many decidable fragments.
- However: not good enough for dealing with orderings, equality and more specific algebraic theories (lattices, abelian groups, rings, fields) $\Rightarrow$ further specialization of inference systems required.

3.18 Semantic Tableaux

Literature:


Like resolution, semantic tableaux were developed in the sixties, independently by Zbigniew Lis and Raymond Smullyan on the basis of work by Gentzen in the 30s and of Beth in the 50s.
Idea

Idea (for the propositional case):

A set \( \{ F \land G \} \cup N \) of formulas has a model if and only if \( \{ F \land G, F, G \} \cup N \) has a model.

A set \( \{ F \lor G \} \cup N \) of formulas has a model if and only if \( \{ F \lor G, F \} \cup N \) or \( \{ F \lor G, G \} \cup N \) has a model.

(and similarly for other connectives).

To avoid duplication, represent sets as paths of a tree.

Continue splitting until two complementary formulas are found \( \Rightarrow \) inconsistency detected.

A Tableau for \( \{ P \land \neg (Q \lor \neg R), \neg Q \lor \neg R \} \)

\[
\begin{align*}
1. & \quad P \land \neg (Q \lor \neg R) \\
2. & \quad \neg Q \lor \neg R \\
3. & \quad \neg Q \\
4. & \quad \neg R \\
5. & \quad P \\
6. & \quad \neg (Q \lor \neg R) \\
7. & \quad \neg Q \\
8. & \quad \neg \neg R \\
9. & \quad R
\end{align*}
\]

This tableau is not “maximal”, however the first “path” is. This path is not “closed”, hence the set \( \{1, 2\} \) is satisfiable. (These notions will all be defined below.)

Properties

Properties of tableau calculi:

analytic: inferences correspond closely to the logical meaning of the symbols.

goal oriented: inferences operate directly on the goal to be proved (unlike, e.g., ordered resolution).

global: some inferences affect the entire proof state (set of formulas), as we will see later.
Propositional Expansion Rules

Expansion rules are applied to the formulas in a tableau and expand the tableau at a leaf. We append the conclusions of a rule (horizontally or vertically) at a leaf, whenever the premise of the expansion rule matches a formula appearing anywhere on the path from the root to that leaf.

Negation Elimination

\[
\begin{array}{ccc}
\neg\neg F & \Rightarrow \neg \top & \Rightarrow \neg \bot \\
\end{array}
\]

α-Expansion

(for formulas that are essentially conjunctions: append subformulas \( \alpha_1 \) and \( \alpha_2 \) one on top of the other)

\[
\frac{\alpha}{\alpha_1} \frac{\alpha_2}{\alpha}
\]

β-Expansion

(for formulas that are essentially disjunctions: append \( \beta_1 \) and \( \beta_2 \) horizontally, i.e., branch into \( \beta_1 \) and \( \beta_2 \))

\[
\frac{\beta}{\beta_1 \mid \beta_2}
\]

Classification of Formulas

<table>
<thead>
<tr>
<th>conjunctive</th>
<th>disjunctive</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>( \alpha_1 ) ( \alpha_2 )</td>
</tr>
<tr>
<td>( F \land G )</td>
<td>( F ) ( G )</td>
</tr>
<tr>
<td>( \neg(F \lor G) )</td>
<td>( \neg F ) ( \neg G )</td>
</tr>
<tr>
<td>( \neg(F \to G) )</td>
<td>( F ) ( \neg G )</td>
</tr>
</tbody>
</table>

We assume that the binary connective \( \leftrightarrow \) has been eliminated in advance.
Tableaux: Notions

A semantic tableau is a marked (by formulas), finite, unordered tree and inductively defined as follows: Let \( \{F_1, \ldots, F_n\} \) be a set of formulas.

(i) The tree consisting of a single path

\[
F_1 \\
\vdots \\
F_n
\]

is a tableau for \( \{F_1, \ldots, F_n\} \). (We do not draw edges if nodes have only one successor.)

(ii) If \( T \) is a tableau for \( \{F_1, \ldots, F_n\} \) and if \( T' \) results from \( T \) by applying an expansion rule then \( T' \) is also a tableau for \( \{F_1, \ldots, F_n\} \).

Note: We may also consider the limit tableau of a tableau expansion; this can be an infinite tree.

A path (from the root to a leaf) in a tableau is called closed, if it either contains \( \bot \), or else it contains both some formula \( F \) and its negation \( \neg F \). Otherwise the path is called open.

A tableau is called closed, if all paths are closed.

A tableau proof for \( F \) is a closed tableau for \( \{\neg F\} \).

A path \( \pi \) in a tableau is called maximal, if for each formula \( F \) on \( \pi \) that is neither a literal nor \( \bot \) nor \( \top \) there exists a node in \( \pi \) at which the expansion rule for \( F \) has been applied.

In that case, if \( F \) is a formula on \( \pi \), \( \pi \) also contains:

(i) \( \alpha_1 \) and \( \alpha_2 \), if \( F \) is a \( \alpha \)-formula,

(ii) \( \beta_1 \) or \( \beta_2 \), if \( F \) is a \( \beta \)-formula, and

(iii) \( F' \), if \( F \) is a negation formula, and \( F' \) the conclusion of the corresponding elimination rule.

A tableau is called maximal, if each path is closed or maximal.

A tableau is called strict, if for each formula the corresponding expansion rule has been applied at most once on each path containing that formula.

A tableau is called clausal, if each of its formulas is a clause.
A Sample Proof

One starts out from the negation of the formula to be proved.

1. \( \neg((P \rightarrow (Q \rightarrow R)) \rightarrow ((P \lor S) \rightarrow ((Q \rightarrow R) \lor S))) \)
2. \( (P \rightarrow (Q \rightarrow R)) \) [1]
3. \( \neg((P \lor S) \rightarrow ((Q \rightarrow R) \lor S)) \) [12]
4. \( P \lor S \) [31]
5. \( \neg((Q \rightarrow R) \lor S) \) [32]
6. \( \neg(Q \rightarrow R) \) [51]
7. \( \neg S \) [52]
8. \( \neg P \) [21]
9. \( Q \rightarrow R \) [2]
10. \( P \) [41]
11. \( S \) [42]

There are three paths, each of them closed.

Properties of Propositional Tableaux

We assume that \( T \) is a tableau for \( \{F_1, \ldots, F_n\} \).

Theorem 3.51 \( \{F_1, \ldots, F_n\} \) satisfiable \( \iff \) some path (i.e., the set of its formulas) in \( T \) is satisfiable.

Proof. (\( \Rightarrow \)) Trivial, since every path contains in particular \( F_1, \ldots, F_n \).
(\( \Leftarrow \)) By induction over the structure of \( T \). \( \square \)

Corollary 3.52 \( T \) closed \( \Rightarrow \) \( \{F_1, \ldots, F_n\} \) unsatisfiable

Theorem 3.53 Every strict propositional tableau expansion is finite.

Proof. New formulas resulting from expansion are either \( \bot, \top \) or subformulas of the expanded formula (modulo de Morgan’s law), so the number of formulas that can occur is finite. By strictness, on each path a formula can be expanded at most once. Therefore, each path is finite, and a finitely branching tree with finite paths is finite by Lemma 1.9. \( \square \)

Conclusion: Strict and maximal tableaux can be effectively constructed.
Refutational Completeness

A set $\mathcal{H}$ of propositional formulas is called a Hintikka set, if

1. there is no $P \in \Pi$ with $P \in \mathcal{H}$ and $\neg P \in \mathcal{H}$;
2. $\bot \notin \mathcal{H}$, $\neg \top \notin \mathcal{H}$;
3. if $\neg \neg F \in \mathcal{H}$, then $F \in \mathcal{H}$;
4. if $\alpha \in \mathcal{H}$, then $\alpha_1 \in \mathcal{H}$ and $\alpha_2 \in \mathcal{H}$;
5. if $\beta \in \mathcal{H}$, then $\beta_1 \in \mathcal{H}$ or $\beta_2 \in \mathcal{H}$.

Lemma 3.54 (Hintikka’s Lemma) Every Hintikka set is satisfiable.

Proof. Let $\mathcal{H}$ be a Hintikka set. Define a valuation $A$ by $A(P) = 1$ if $P \in \mathcal{H}$ and $A(P) = 0$ otherwise. Then show that $A(F) = 1$ for all $F \in \mathcal{H}$ by induction over the size of formulas. 

Theorem 3.55 Let $\pi$ be a maximal open path in a tableau. Then the set of formulas on $\pi$ is satisfiable.

Proof. We show that set of formulas on $\pi$ is a Hintikka set: Conditions (3), (4), (5) follow from the fact that $\pi$ is maximal; conditions (1) and (2) follow from the fact that $\pi$ is open and from maximality for the second negation elimination rule.

Note: The theorem holds also for infinite trees that are obtained as the limit of a tableau expansion.

Theorem 3.56 $\{F_1, \ldots, F_n\}$ satisfiable $\iff$ there exists no closed strict tableau for $\{F_1, \ldots, F_n\}$.

Proof. ($\Rightarrow$) Clear by Cor. 3.52.
($\Leftarrow$) Let $T$ be a strict maximal tableau for $\{F_1, \ldots, F_n\}$ and let $\pi$ be an open path in $T$. By the previous theorem, the set of formulas on $\pi$ is satisfiable, and hence by Theorem 3.51 the set $\{F_1, \ldots, F_n\}$, is satisfiable.
Consequences

The validity of a propositional formula $F$ can be established by constructing a strict maximal tableau for \( \{\neg F\} \):

- $T$ closed $\iff$ $F$ valid.
- It suffices to test complementarity of paths w.r.t. atomic formulas (cf. reasoning in the proof of Theorem 3.55).
- Which of the potentially many strict maximal tableaux one computes does not matter. In other words, tableau expansion rules can be applied don’t-care non-deterministically (“proof confluence”).
- The expansion strategy, however, can have a dramatic impact on the tableau size.

A Variant of the $\beta$-Rule

Since $F \lor G \models F \lor (G \land \neg F)$, the $\beta$ expansion rule

\[
\begin{array}{c}
\beta \\
\hline
\beta_1 \mid \beta_2
\end{array}
\]

can be replaced by the following variant:

\[
\begin{array}{c}
\beta \\
\hline
\beta_1 \\
\beta_2 \\
\neg \beta_1
\end{array}
\]

The variant $\beta$-rule can lead to much shorter proofs, but it is not always beneficial.

In general, it is most helpful if $\neg \beta_1$ can be at most (iteratively) $\alpha$-expanded.
3.19 Semantic Tableaux for First-Order Logic

There are two ways to extend the tableau calculus to quantified formulas:

- using ground instantiation,
- using free variables.

### Tableaux with Ground Instantiation

Classification of quantified formulas:

<table>
<thead>
<tr>
<th>universal</th>
<th>existential</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall x F$</td>
<td>$\exists x F$</td>
</tr>
<tr>
<td>$\neg \exists x F$</td>
<td>$\neg \forall x F$</td>
</tr>
</tbody>
</table>

Idea:

Replace universally quantified formulas by appropriate ground instances.

**$\gamma$-expansion**

$$ \frac{\gamma}{\gamma(t)} \quad \text{where } t \text{ is some ground term} $$

**$\delta$-expansion**

$$ \frac{\delta}{\delta(c)} \quad \text{where } c \text{ is a new Skolem constant} $$

Skolemization becomes part of the calculus and needs not necessarily be applied in a preprocessing step. Of course, one could do Skolemization beforehand, and then the $\delta$-rule would not be needed.

Note:

Skolem constants are sufficient:
In a $\delta$-formula $\exists x F$, $\exists$ is the outermost quantifier and $x$ is the only free variable in $F$.

Problems:

Having to guess ground terms is impractical.

Even worse, we may have to guess several ground instances, as strictness for $\gamma$ is incomplete. For instance, constructing a closed tableau for

$$ \{ \forall x (P(x) \to P(f(x))), \ P(b), \ \neg P(f(f(b))) \} $$

is impossible without applying $\gamma$-expansion twice on one path.
Free-Variable Tableaux

An alternative approach:

Delay the instantiation of universally quantified variables.
Replace universally quantified variables by new free variables.

Intuitively, the free variables are universally quantified outside of the entire tableau.

$\gamma$-expansion

$$\frac{\gamma}{\gamma(x)} \text{ where } x \text{ is a new free variable}$$

$\delta$-expansion

$$\frac{\delta}{\delta(f(x_1, \ldots, x_n))}$$

where $f$ is a new Skolem function, and the $x_i$ are the free variables in $\delta$

Application of expansion rules has to be supplemented by a substitution rule:

(iii) If $T$ is a tableau for $\{F_1, \ldots, F_n\}$ and if $\sigma$ is a substitution, then $T\sigma$ is also a tableau for $\{F_1, \ldots, F_n\}$.

The substitution rule may, potentially, modify all the formulas of a tableau. This feature is what makes the tableau method a global proof method. (Resolution, by comparison, is a local method.)

One can show that it is sufficient to consider substitutions $\sigma$ for which there is a path in $T$ containing two literals $\neg A$ and $B$ such that $\sigma = \text{mgu}(A, B)$. Such tableaux are called AMGU-Tableaux.
Example

1. \( \neg (\exists w \forall x \ P(x, w, f(x, w)) \rightarrow \exists w \forall x \exists y \ P(x, w, y)) \)
2. \( \exists w \forall x \ P(x, w, f(x, w)) \)
3. \( \neg \exists w \forall x \exists y \ P(x, w, y) \)
4. \( \forall x \ P(x, c, f(x, c)) \)
5. \( \neg \forall x \exists y \ P(x, v_1, y) \)
6. \( \neg \exists y \ P(b(v_1), v_1, y) \)
7. \( P(v_2, c, f(v_2, c)) \)
8. \( \neg P(b(v_1), v_1, v_3) \)

7. and 8. are complementary (modulo unification):

\[ \{v_2 \doteq b(v_1), \ c \doteq v_1, \ f(v_2, c) \doteq v_3\} \]

is solvable with an mgu \( \sigma = \{v_1 \mapsto c, \ v_2 \mapsto b(c), \ v_3 \mapsto f(b(c), c)\} \), and hence, \( T\sigma \) is a closed (linear) tableau for the formula in 1.

Problem:

Strictness for \( \gamma \) is still incomplete. For instance, constructing a closed tableau for

\[ \{\forall x \ (P(x) \rightarrow P(f(x))), \ P(b), \ \neg P(f(b)))\} \]

is impossible without applying \( \gamma \)-expansion twice on one path.

Semantic Tableaux vs. Resolution

- Tableaux: global, goal-oriented, “backward”.
- Resolution: local, “forward”.
- Goal-orientation is a clear advantage if only a small subset of a large set of formulas is necessary for a proof. (Note that resolution provers saturate also those parts of the clause set that are irrelevant for proving the goal.)
- Resolution can be combined with more powerful redundancy elimination methods; because of its global nature this is more difficult for the tableau method.
- Resolution can be refined to work well with equality; for tableaux this seems to be impossible.
- On the other hand tableau calculi can be easily extended to other logics; in particular tableau provers are very successful in modal and description logics.

102
3.20 Other Deductive Systems

- Instantiation-based methods
  - Resolution-based instance generation
  - Disconnection calculus
  -...
- Natural deduction
- Sequent calculus/Gentzen calculus
- Hilbert calculus

**Instantiation-Based Methods for FOL**

Idea:

Overlaps of complementary literals produce instantiations (as in resolution);

However, contrary to resolution, clauses are not recombined.

Instead: treat remaining variables as constant and use efficient propositional proof methods, such as CDCL.

There are both saturation-based variants, such as partial instantiation (Hooker et al. 2002) or resolution-based instance generation (Inst-Gen) (Ganzinger and Korovin 2003), and tableau-style variants, such as the disconnection calculus (Billon 1996; Letz and Stenz 2001).

Successful in practice for problems that are “almost propositional” (i.e., no non-constant function symbols, no equality).

**Natural Deduction**

Idea:

Model the concept of proofs from assumptions as humans do it.

To prove $F \rightarrow G$, assume $F$ and try to derive $G$.

Initial ideas: Jaśkowski (1934), Gentzen (1934); extended by Prawitz (1965).

Popular in interactive proof systems.
Sequent Calculus

Idea:
Assumptions internalized into the data structure of sequents

\[ F_1, \ldots, F_m \vdash G_1, \ldots, G_k \]

meaning

\[ F_1 \land \cdots \land F_m \rightarrow G_1 \lor \cdots \lor G_k \]

Inferences rules, e.g.:

\[ \Gamma \vdash \Delta \quad \Gamma, F \vdash \Delta \quad \Sigma, G \vdash \Pi \quad (\lor L) \]

\[ \Gamma \vdash \Delta \quad \Gamma, F, \Delta \quad (\lor R) \]

\[ \Gamma \vdash \Delta \quad \Gamma, F, \Delta \quad \Sigma, G \vdash \Pi \quad (\land L) \]

\[ \Gamma, \Sigma \vdash F \land G, \Delta, \Pi \quad (\land R) \]

Initial idea: Gentzen 1934.

Perfect symmetry between the handling of assumptions and their consequences; interesting for proof theory.

Can be used both backwards and forwards.

Allows to simulate both natural deduction and semantic tableaux.

Hilbert Calculus

Idea:

Direct proof method (proves a theorem from axioms, rather than refuting its negation)

Axiom schemes, e.g.,

\[ F \rightarrow (G \rightarrow F) \]

\[ (F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H)) \]

plus Modus ponens:

\[ \frac{F}{F \rightarrow G} \quad G \]

Unsuitable for finding or reading proofs, but sometimes used for specifying (e.g. modal) logics.
4 First-Order Logic with Equality

Equality is the most important relation in mathematics and functional programming.

In principle, problems in first-order logic with equality can be handled by any prover for first-order logic without equality:

4.1 Handling Equality Naively

Proposition 4.1 Let $F$ be a closed first-order formula with equality. Let $\sim \notin \Pi$ be a new predicate symbol. The set $\text{Eq}(\Sigma)$ contains the formulas

$$\forall x (x \sim x)$$
$$\forall x, y (x \sim y \rightarrow y \sim x)$$
$$\forall x, y, z (x \sim y \land y \sim z \rightarrow x \sim z)$$
$$\forall \bar{x}, \bar{y} (x_1 \sim y_1 \land \cdots \land x_n \sim y_n \rightarrow f(x_1, \ldots, x_n) \sim f(y_1, \ldots, y_n))$$
$$\forall \bar{x}, \bar{y} (x_1 \sim y_1 \land \cdots \land x_m \sim y_m \land P(x_1, \ldots, x_m) \rightarrow P(y_1, \ldots, y_m))$$

for every $f/n \in \Omega$ and $P/m \in \Pi$. Let $\tilde{F}$ be the formula that one obtains from $F$ if every occurrence of $\approx$ is replaced by $\sim$. Then $F$ is satisfiable if and only if $\text{Eq}(\Sigma) \cup \{\tilde{F}\}$ is satisfiable.

Proof. Let $\Sigma = (\Omega, \Pi)$, let $\Sigma_1 = (\Omega, \Pi \cup \{\approx/2\})$.

For the “only if” part assume that $F$ is satisfiable and let $A$ be a $\Sigma$-model of $F$. Then we define a $\Sigma_1$-algebra $B$ in such a way that $B$ and $A$ have the same universe, $f_B = f_A$ for every $f \in \Omega$, $P_B = P_A$ for every $P \in \Pi$, and $\sim_B$ is the identity relation on the universe. It is easy to check that $B$ is a model of both $\tilde{F}$ and of $\text{Eq}(\Sigma)$.

For the “if” part assume that the $\Sigma_1$-algebra $B = (U_B, \ (f_B : U_B^n \rightarrow U_B)_{f \in \Omega}, \ (P_B \subseteq U_B^m)_{P \in \Pi})$ is a model of $\text{Eq}(\Sigma) \cup \{\tilde{F}\}$. Then the interpretation $\sim_B$ of $\sim$ in $B$ is a congruence relation on $U_B$ with respect to the functions $f_B$ and the predicates $P_B$.

We will now construct a $\Sigma$-algebra $A$ from $B$ and the congruence relation $\sim_B$. Let $[a]$ be the congruence class of an element $a \in U_B$ with respect to $\sim_B$. The universe $U_A$ of $A$ is the set $\{[a] \mid a \in U_B\}$ of congruence classes of the universe of $B$. For a function symbol $f \in \Omega$, we define $f_A([a_1], \ldots, [a_n]) = [f_B(a_1, \ldots, a_n)]$, and for a predicate symbol $P \in \Pi$, we define $([a_1], \ldots, [a_n]) \in P_A$ if and only if $(a_1, \ldots, a_n) \in P_B$. Observe that this is well-defined: If we take different representatives of the same congruence class, we get the same result by congruence of $\sim_B$. For any $\mathcal{A}$-assignment $\gamma$ choose some $B$-assignment $\beta$ such that $B(\beta)(x) \in A(\gamma)(x)$ for every $x$, then for every $\Sigma$-term $t$ we have $A(\gamma)(t) = [B(\beta)(t)]$, and analogously for every $\Sigma$-formula $G$, $A(\gamma)(G) = B(\beta)(G)$. Both properties can easily shown by structural induction. Therefore, $A$ is a model of $F$. \qed
An analogous proposition holds for sets of closed first-order formulas with equality.

By giving the equality axioms explicitly, first-order problems with equality can in principle be solved by a standard resolution or tableaux prover.

But this is unfortunately not efficient (mainly due to the transitivity and congruence axioms).

Equality is theoretically difficult: First-order functional programming is Turing-complete. But: resolution theorem provers cannot even solve equational problems that are intuitively easy.

Consequence: to handle equality efficiently, knowledge must be integrated into the theorem prover.

Roadmap

How to proceed:

- This semester: Equations (unit clauses with equality)
  - Term rewrite systems
  - Expressing semantic consequence syntactically
  - Knuth-Bendix-Completion
  - Entailment for equations

- Next semester: Equational clauses
  - Combining resolution and KB-completion → Superposition
  - Entailment for clauses with equality

4.2 Rewrite Systems

Let $E$ be a set of (implicitly universally quantified) equations.

The rewrite relation $\rightarrow_E \subseteq T_\Sigma(X) \times T_\Sigma(X)$ is defined by

$$s \rightarrow_E t \text{ iff there exist } (l \approx r) \in E, \; p \in \text{pos}(s), \text{ and } \sigma : X \rightarrow T_\Sigma(X), \text{ such that } s|_p = l\sigma \text{ and } t = s|r\sigma|_p.$$
An instance of the lhs (left-hand side) of an equation is called a redex (reducible expression). Contracting a redex means replacing it with the corresponding instance of the rhs (right-hand side) of the rule.

An equation \( l \approx r \) is also called a rewrite rule, if \( l \) is not a variable and \( \text{var}(l) \supseteq \text{var}(r) \).

Notation: \( l \rightarrow r \).

A set of rewrite rules is called a term rewrite system (TRS).

We say that a set of equations \( E \) or a TRS \( R \) is terminating, if the rewrite relation \( \rightarrow_E \) or \( \rightarrow_R \) has this property.

(Analogously for other properties of abstract reduction systems).

Note: If \( E \) is terminating, then it is a TRS.

**E-Algebras**

Let \( E \) be a set of universally quantified equations. A model of \( E \) is also called an \( E \)-algebra.

If \( E \models \forall \vec{x}(s \approx t) \), i.e., \( \forall \vec{x}(s \approx t) \) is valid in all \( E \)-algebras, we write this also as \( s \approx_E t \).

Goal:
Use the rewrite relation \( \rightarrow_E \) to express the semantic consequence relation syntactically:
\[
s \approx_E t \text{ if and only if } s \leftrightarrow_{E}^* t.
\]

Let \( E \) be a set of equations over \( T_{\Sigma}(X) \). The following inference system allows to derive consequences of \( E \):

\[
E \vdash t \approx t \quad \text{(Reflexivity)}
\]

for every \( t \in T_{\Sigma}(X) \)

\[
E \vdash t \approx t' \quad \text{and} \quad E \vdash t' \approx t
\]

\[
E \vdash t \approx t' \quad E \vdash t' \approx t'' \quad \text{(Transitivity)}
\]

\[
E \vdash t_1 \approx t'_1 \quad \ldots \quad E \vdash t_n \approx t'_n
\]

\[
E \vdash f(t_1, \ldots, t_n) \approx f(t'_1, \ldots, t'_n) \quad \text{(Congruence)}
\]

\[
E \vdash t \sigma \approx t' \sigma \quad \text{if } (t \approx t') \in E \text{ and } \sigma : X \rightarrow T_{\Sigma}(X)
\]

(Instance)
Lemma 4.2 The following properties are equivalent:

(i) \( s \leftrightarrow^*_E t \)

(ii) \( E \vdash s \approx t \) is derivable.

Proof. (i)\( \Rightarrow \) (ii): \( s \leftrightarrow^*_E t \) implies \( E \vdash s \approx t \) by induction on the depth of the position where the equation is applied; then \( s \leftrightarrow^*_E t \) implies \( E \vdash s \approx t \) by induction on the number of rewrite steps in \( s \leftrightarrow^*_E t \).

(ii)\( \Rightarrow \) (i): By induction on the size (number of symbols) of the derivation for \( E \vdash s \approx t \).
\[\blacksquare\]

Constructing a quotient algebra:

Let \( X \) be a set of variables.

For \( t \in T_\Sigma(X) \) let \([t] = \{ t' \in T_\Sigma(X) \mid E \vdash t \approx t' \}\) be the congruence class of \( t \).

Define a \( \Sigma \)-algebra \( T_\Sigma(X)/E \) (abbreviated by \( \mathcal{T} \)) as follows:

\[ U_\mathcal{T} = \{ [t] \mid t \in T_\Sigma(X) \} \]

\[ f_\mathcal{T}([t_1], \ldots, [t_n]) = [f(t_1, \ldots, t_n)] \text{ for } f/n \in \Omega. \]

Lemma 4.3 \( f_\mathcal{T} \) is well-defined: If \([t_i] = [t'_i]\), then \([f(t_1, \ldots, t_n)] = [f(t'_1, \ldots, t'_n)]\).

Proof. Follows directly from the Congruence rule for \( \vdash \).
\[\blacksquare\]

Lemma 4.4 \( \mathcal{T} = T_\Sigma(X)/E \) is an \( E \)-algebra.

Proof. Let \( \forall x_1 \ldots x_n(s \approx t) \) be an equation in \( E \); let \( \beta \) be an arbitrary assignment.

We have to show that \( \mathcal{T}(\beta)(\forall x(s \approx t)) = 1 \), or equivalently, that \( \mathcal{T}(\gamma)(s) = \mathcal{T}(\gamma)(t) \) for all \( \gamma = \beta[x_i \mapsto [v_i] \mid 1 \leq i \leq n] \) with \([v_i] \in U_\mathcal{T}\).

Let \( \sigma = \{ x_1 \mapsto v_1, \ldots, x_n \mapsto v_n \} \), then we get by structural induction that \( u\sigma \in \mathcal{T}(\gamma)(u) \) for every \( u \in T_\Sigma(\{x_1, \ldots, x_n\}) \). In particular, \( s\sigma \in \mathcal{T}(\gamma)(s) \) and \( t\sigma \in \mathcal{T}(\gamma)(t) \).

By the Instance rule, \( E \vdash s\sigma \approx t\sigma \) is derivable, hence \( \mathcal{T}(\gamma)(s) = [s\sigma] = [t\sigma] = \mathcal{T}(\gamma)(t) \).
\[\blacksquare\]
Lemma 4.5 Let $X$ be a countably infinite set of variables; let $s, t \in T_\Sigma(Y)$. If $T_\Sigma(X)/E \models \forall \bar{x}(s \approx t)$, then $E \vdash s \approx t$ is derivable.

Proof. Without loss of generality, we assume that all variables in $\bar{x}$ are contained in $X$. (Otherwise, we rename the variables in the equation. Since $X$ is countably infinite, this is always possible.) Assume that $T \models \forall \bar{x}(s \approx t)$, i.e., $T(\beta)(\forall \bar{x}(s \approx t)) = 1$. Consequently, $T(\gamma)(s) = T(\gamma)(t)$ for all $\gamma = \beta[x_i \mapsto [v_i] \mid 1 \leq i \leq n]$ with $[v_i] \in U_T$.

Choose $v_i := x_i$, then by structural induction $[u] = T(\gamma)(u)$ for every $u \in T_\Sigma(\{x_1, ..., x_n\})$, so $[s] = T(\gamma)(s) = T(\gamma)(t) = [t]$. Therefore $E \vdash s \approx t$ is derivable by definition of $T$. \qed

Theorem 4.6 ("Birkhoff’s Theorem") Let $X$ be a countably infinite set of variables, let $E$ be a set of (universally quantified) equations. Then the following properties are equivalent for all $s, t \in T_\Sigma(X)$:

(i) $s \leftrightarrow^*_E t$.

(ii) $E \vdash s \approx t$ is derivable.

(iii) $s \approx_E t$, i.e., $E \models \forall \bar{x}(s \approx t)$.

(iv) $T_\Sigma(X)/E \models \forall \bar{x}(s \approx t)$.

Proof. (i)$\Leftrightarrow$(ii): Lemma 4.2.

(ii)$\Rightarrow$(iii): By induction on the size of the derivation for $E \vdash s \approx t$.

(iii)$\Rightarrow$(iv): Obvious, since $T = T_\Sigma(X)/E$ is an $E$-algebra.

(iv)$\Rightarrow$(ii): Lemma 4.5. \qed

Universal Algebra

$T_\Sigma(X)/E = T_\Sigma(X)/\approx_E = T_\Sigma(X)/\leftrightarrow^*_E$ is called the free $E$-algebra with generating set $X/\approx_E = \{[x] \mid x \in X\}$:

Every mapping $\varphi : X/\approx_E \to \mathcal{B}$ for some $E$-algebra $\mathcal{B}$ can be extended to a homomorphism $\hat{\varphi} : T_\Sigma(X)/E \to \mathcal{B}$.

$T_\Sigma(\emptyset)/E = T_\Sigma(\emptyset)/\approx_E = T_\Sigma(\emptyset)/\leftrightarrow^*_E$ is called the initial $E$-algebra.

$\approx_E = \{ (s, t) \mid E \models s \approx t \}$ is called the equational theory of $E$.

$\approx^I_E = \{ (s, t) \mid T_\Sigma(\emptyset)/E \models s \approx t \}$ is called the inductive theory of $E$.

Example:

Let $E = \{ \forall x(x + 0 \approx x), \forall x\forall y(x + s(y) \approx s(x + y)) \}$. Then $x + y \approx^I_E y + x$, but $x + y \not\approx_E y + x$. 

109
4.3 Confluence

Let \((A, \rightarrow)\) be an abstract reduction system. 

\(b\) and \(c\in A\) are joinable, if there is a \(a\) such that \(b \rightarrow^* a \leftarrow^* c\).  
Notation: \(b \downarrow c\).

The relation \(\rightarrow\) is called

- Church-Rosser, if \(b \leftrightarrow^* c\) implies \(b \downarrow c\).
- confluent, if \(b \leftarrow^* a \rightarrow^* c\) implies \(b \downarrow c\).
- locally confluent, if \(b \leftarrow a \rightarrow c\) implies \(b \downarrow c\).
- convergent, if it is confluent and terminating.

**Theorem 4.7** The following properties are equivalent:

(i) \(\rightarrow\) has the Church-Rosser property.

(ii) \(\rightarrow\) is confluent.

**Proof.** (i)⇒(ii): trivial.

(ii)⇒(i): by induction on the number of peaks in the derivation \(b \leftrightarrow^* c\). □

**Lemma 4.8** If \(\rightarrow\) is confluent, then every element has at most one normal form.

**Proof.** Suppose that some element \(a\in A\) has normal forms \(b\) and \(c\), then \(b \leftarrow^* a \rightarrow^* c\).  

If \(\rightarrow\) is confluent, then \(b \rightarrow^* d \leftarrow^* c\) for some \(d\in A\).  

Since \(b\) and \(c\) are normal forms, both derivations must be empty, hence \(b \rightarrow^0 d \leftarrow^0 c\), so \(b, c,\) and \(d\) must be identical. □

**Corollary 4.9** If \(\rightarrow\) is normalizing and confluent, then every element \(b\) has a unique normal form.

**Proposition 4.10** If \(\rightarrow\) is normalizing and confluent, then \(b \leftrightarrow^* c\) if and only if \(b \downarrow = c \downarrow\).

**Proof.** Either using Thm. 4.7 or directly by induction on the length of the derivation of \(b \leftrightarrow^* c\). □
Confluence and Local Confluence

**Theorem 4.11 (“Newman’s Lemma”)** If a terminating relation $\rightarrow$ is locally confluent, then it is confluent.

**Proof.** Let $\rightarrow$ be a terminating and locally confluent relation. Then $\rightarrow^+$ is a well-founded ordering. Define $\phi(a) \iff (\forall b, c : b \leftarrow^* a \rightarrow^* c \Rightarrow b \downarrow c)$.

We prove $\phi(a)$ for all $a \in A$ by well-founded induction over $\rightarrow^+$:

Case 1: $b \leftarrow^0 a \rightarrow^* c$: trivial.

Case 2: $b \leftarrow^* a \rightarrow^0 c$: trivial.

Case 3: $b \leftarrow^* b' \leftarrow a \rightarrow c' \rightarrow^* c$: use local confluence, then use the induction hypothesis.

$\square$

**Rewrite Relations**

**Corollary 4.12** If $E$ is convergent (i.e., terminating and confluent), then $s \approx_E t$ if and only if $s \leftrightarrow^* E t$ if and only if $s \downarrow_E = t \downarrow_E$.

**Corollary 4.13** If $E$ is finite and convergent, then $\approx_E$ is decidable.

Reminder:
If $E$ is terminating, then it is confluent if and only if it is locally confluent.

Problems:

- Show local confluence of $E$.
- Show termination of $E$.
- Transform $E$ into an equivalent set of equations that is locally confluent and terminating.
4.4 Critical Pairs

Showing local confluence (Sketch):

Problem: If \( t_1 \leftarrow E t_0 \rightarrow E t_2 \), does there exist a term \( s \) such that \( t_1 \rightarrow^* s \leftarrow^* t_2 \)?

If the two rewrite steps happen in different subtrees (disjoint red exes): yes.

If the two rewrite steps happen below each other (overlap at or below a variable position): yes.

If the left-hand sides of the two rules overlap at a non-variable position: needs further investigation.

Question:
Are there rewrite rules \( l_1 \rightarrow r_1 \) and \( l_2 \rightarrow r_2 \) such that some subterm \( l_1|_p \) and \( l_2 \) have a common instance \( (l_1|_p)\sigma_1 = l_2\sigma_2 \)?

Observation:
If we assume w.l.o.g. that the two rewrite rules do not have common variables, then only a single substitution is necessary: \( (l_1|_p)\sigma = l_2\sigma \).

Further observation:
The mgu of \( l_1|_p \) and \( l_2 \) subsumes all unifiers \( \sigma \) of \( l_1|_p \) and \( l_2 \).

Let \( l_i \rightarrow r_i \) (\( i = 1, 2 \)) be two rewrite rules in a TRS \( R \) whose variables have been renamed such that \( \text{var}(l_1) \cap \text{var}(l_2) = \emptyset \). (Remember that \( \text{var}(l_i) \supseteq \text{var}(r_i) \).)

Let \( p \in \text{pos}(l_1) \) be a position such that \( l_1|_p \) is not a variable and \( \sigma \) is an mgu of \( l_1|_p \) and \( l_2 \).

Then \( r_1\sigma \leftarrow l_1\sigma \rightarrow (l_1\sigma)[r_2\sigma]_p \).

\( \langle r_1\sigma, (l_1\sigma)[r_2\sigma]_p \rangle \) is called a critical pair of \( R \).

The critical pair is joinable (or: converges), if \( r_1\sigma \downarrow_R (l_1\sigma)[r_2\sigma]_p \).

**Theorem 4.14 (“Critical Pair Theorem”)** A TRS \( R \) is locally confluent if and only if all its critical pairs are joinable.

**Proof.** “only if”: obvious, since joinability of a critical pair is a special case of local confluence.

“if”: Suppose \( s \) rewrites to \( t_1 \) and \( t_2 \) using rewrite rules \( l_i \rightarrow r_i \in R \) at positions \( p_i \in \text{pos}(s) \), where \( i = 1, 2 \). Without loss of generality, we can assume that the two rules are variable disjoint, hence \( s|_{p_i} = l_i\theta \) and \( t_i = s[r_i\theta]_{p_i} \).

We distinguish between two cases: Either \( p_1 \) and \( p_2 \) are in disjoint subtrees \( (p_1 \parallel p_2) \), or one is a prefix of the other (w.l.o.g., \( p_1 \leq p_2 \)).
Case 1: \( p_1 \parallel p_2 \).

Then \( s = s[l_1\theta]_{p_1}[l_2\theta]_{p_2} \), and therefore \( t_1 = s[r_1\theta]_{p_1}[l_2\theta]_{p_2} \) and \( t_2 = s[l_1\theta]_{p_1}[r_2\theta]_{p_2} \).

Let \( t_0 = s[r_1\theta]_{p_1}[r_2\theta]_{p_2} \). Then clearly \( t_1 \rightarrow_R t_0 \) using \( l_2 \rightarrow r_2 \) and \( t_2 \rightarrow_R t_0 \) using \( l_1 \rightarrow r_1 \).

Case 2: \( p_1 \leq p_2 \).

Case 2.1: \( p_2 = p_1q_1q_2 \), where \( l_1|_{q_1} \) is some variable \( x \).

In other words, the second rewrite step takes place at or below a variable in the first rule. Suppose that \( x \) occurs \( m \) times in \( l_1 \) and \( n \) times in \( r_1 \) (where \( m \geq 1 \) and \( n \geq 0 \)).

Then \( t_1 \rightarrow_R t_0 \) by applying \( l_2 \rightarrow r_2 \) at all positions \( p_1q'q_2 \), where \( q' \) is a position of \( x \) in \( r_1 \).

Conversely, \( t_2 \rightarrow_R t_0 \) by applying \( l_2 \rightarrow r_2 \) at all positions \( p_1qq_2 \), where \( q \) is a position of \( x \) in \( l_1 \) different from \( q_1 \), and by applying \( l_1 \rightarrow r_1 \) at \( p_1 \) with the substitution \( \theta' \), where \( \theta' = \theta[x \mapsto (x\theta)|r_2\theta]_{q_2} \).

Case 2.2: \( p_2 = p_1p \), where \( p \) is a non-variable position of \( l_1 \).

Then \( s|_{p_2} = l_2\theta \) and \( s|_{p_2} = (s|_{p_1})|_{p} = (l_1\theta)|_{p} = (l_1|_{p})\theta \), so \( \theta \) is a unifier of \( l_2 \) and \( l_1|_{p} \).

Let \( \sigma \) be the mgu of \( l_2 \) and \( l_1|_{p} \), then \( \theta = \tau \circ \sigma \) and \( \langle r_1\sigma, (l_1\sigma)[r_2\sigma]_p \rangle \) is a critical pair.

By assumption, it is joinable, so \( r_1\sigma \rightarrow^*_{R} \tau \leftarrow^*_{R} (l_1\sigma)[r_2\sigma]_p \).

Consequently, \( t_1 = s[r_1\theta]_{p_1} = s[r_1\sigma\tau]_{p_1} \rightarrow^*_{R} s[v\tau]_{p_1} \) and \( t_2 = s[r_2\theta]_{p_2} = s[(l_1\theta)[r_2\theta]_p]_{p_1} = s[(l_1\sigma)[r_2\sigma]_p]_{p_1} \rightarrow^*_{R} s[v\tau]_{p_1} \).

This completes the proof of the Critical Pair Theorem.

Note: Critical pairs between a rule and (a renamed variant of) itself must be considered – except if the overlap is at the root (i.e., \( p = \varepsilon \)).

**Corollary 4.15** A terminating TRS \( R \) is confluent if and only if all its critical pairs are joinable.

**Proof.** By Newman’s Lemma and the Critical Pair Theorem.

**Corollary 4.16** For a finite terminating TRS, confluence is decidable.

**Proof.** For every pair of rules and every non-variable position in the first rule there is at most one critical pair \( \langle u_1, u_2 \rangle \).

Reduce every \( u_1 \) to some normal form \( u'_1 \). If \( u'_1 = u'_2 \) for every critical pair, then \( R \) is confluent, otherwise there is some non-confluent situation \( u'_1 \leftarrow^*_{R} u_1 \leftarrow_R s \rightarrow_R u_2 \rightarrow^*_{R} u'_2 \).
4.5 Termination

Termination problems:

Given a finite TRS $R$ and a term $t$, are all $R$-reductions starting from $t$ terminating?

Given a finite TRS $R$, are all $R$-reductions terminating?

Proposition 4.17 Both termination problems for TRSs are undecidable in general.

Proof. Encode Turing machines using rewrite rules and reduce the (uniform) halting problems for TMs to the termination problems for TRSs. \(\square\)

Consequence:
Decidable criteria for termination are not complete.

Two Different Scenarios

Depending on the application, the TRS whose termination we want to show can be

(i) fixed and known in advance, or

(ii) evolving (e.g., generated by some saturation process).

Methods for case (ii) are also usable for case (i).
Many methods for case (i) are not usable for case (ii).

We will first consider case (ii);
additional techniques for case (i) will be considered later.

Reduction Orderings

Goal:

Given a finite TRS $R$, show termination of $R$ by looking at finitely many rules $l \rightarrow r \in R$, rather than at infinitely many possible replacement steps $s \rightarrow_R s'$.

A binary relation $\sqsupset$ over $T_\Sigma(X)$ is called compatible with $\Sigma$-operations, if $s \sqsupset s'$ implies $f(t_1, \ldots, s, \ldots, t_n) \sqsupset f(t_1, \ldots, s', \ldots, t_n)$ for all $f \in \Omega$ and $s, s', t_i \in T_\Sigma(X)$.

Lemma 4.18 The relation $\sqsupset$ is compatible with $\Sigma$-operations, if and only if $s \sqsupset s'$ implies $t[s]_p \sqsupset t[s']_p$ for all $s, s', t \in T_\Sigma(X)$ and $p \in \text{pos}(t)$.
Note: compatible with \( \Sigma \)-operations = compatible with contexts.

A binary relation \( \sqsubseteq \) over \( T_\Sigma(X) \) is called stable under substitutions, if \( s \sqsubseteq s' \) implies \( s\sigma \sqsubseteq s'\sigma \) for all \( s, s' \in T_\Sigma(X) \) and substitutions \( \sigma \).

A binary relation \( \sqsubseteq \) is called a rewrite relation, if it is compatible with \( \Sigma \)-operations and stable under substitutions.

Example: If \( R \) is a TRS, then \( \rightarrow_R \) is a rewrite relation.

A strict partial ordering over \( T_\Sigma(X) \) that is a rewrite relation is called rewrite ordering.

A well-founded rewrite ordering is called reduction ordering.

**Theorem 4.19** A TRS \( R \) terminates if and only if there exists a reduction ordering \( \succ \) such that \( l \succ r \) for every rule \( l \rightarrow r \in R \).

**Proof.** “if”: \( s \rightarrow_R s' \) if and only if \( s = t[l\sigma]_p, s' = t[r\sigma]_p \). If \( l \succ r \), then \( l\sigma \succ r\sigma \) and therefore \( t[l\sigma]_p \succ t[r\sigma]_p \). This implies \( \rightarrow_R \subseteq \succ \). Since \( \succ \) is a well-founded ordering, \( \rightarrow_R \) is terminating.

“only if”: Define \( \succ = \rightarrow^*_R \). If \( \rightarrow_R \) is terminating, then \( \succ \) is a reduction ordering. \( \square \)

**The Interpretation Method**

Proving termination by interpretation:

Let \( \mathcal{A} \) be a \( \Sigma \)-algebra; let \( \succ \) be a well-founded strict partial ordering on its universe.

Define the ordering \( \succ_\mathcal{A} \) over \( T_\Sigma(X) \) by \( s \succ_\mathcal{A} t \) iff \( \mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(t) \) for all assignments \( \beta : X \rightarrow U_\mathcal{A} \).

Is \( \succ_\mathcal{A} \) a reduction ordering?

**Lemma 4.20** \( \succ_\mathcal{A} \) is stable under substitutions.

**Proof.** Let \( s \succ_\mathcal{A} s' \), that is, \( \mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(s') \) for all assignments \( \beta : X \rightarrow U_\mathcal{A} \). Let \( \sigma \) be a substitution. We have to show that \( \mathcal{A}(\gamma)(s\sigma) \succ \mathcal{A}(\gamma)(s'\sigma) \) for all assignments \( \gamma : X \rightarrow U_\mathcal{A} \). Choose \( \beta = \gamma \circ \sigma \), then by the substitution lemma, \( \mathcal{A}(\gamma)(s\sigma) = \mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(s') = \mathcal{A}(\gamma)(s'\sigma) \). Therefore \( s\sigma \succ_\mathcal{A} s'\sigma \). \( \square \)

A function \( \phi : U^n_\mathcal{A} \rightarrow U_\mathcal{A} \) is called monotone (with respect to \( \succ \)), if \( a \succ a' \) implies \( \phi(b_1, \ldots, a, \ldots, b_n) \succ \phi(b_1, \ldots, a', \ldots, b_n) \) for all \( a, a', b_i \in U_\mathcal{A} \).

**Lemma 4.21** If the interpretation \( f_\mathcal{A} \) of every function symbol \( f \) is monotone w. r. t. \( \succ \), then \( \succ_\mathcal{A} \) is compatible with \( \Sigma \)-operations.
Proof. Let \( s \succ_{A} s' \), that is, \( A(\beta)(s) \succ A(\beta)(s') \) for all \( \beta : X \to U_{A} \). Let \( \beta : X \to U_{A} \) be an arbitrary assignment. Then
\[
A(\beta)(f(t_{1}, \ldots, s, \ldots, t_{n})) = f_{A}(A(\beta)(t_{1}), \ldots, A(\beta)(s), \ldots, A(\beta)(t_{n}))
\succ f_{A}(A(\beta)(t_{1}), \ldots, A(\beta)(s'), \ldots, A(\beta)(t_{n}))
= A(\beta)(f(t_{1}, \ldots, s', \ldots, t_{n}))
\]
Therefore \( f(t_{1}, \ldots, s, \ldots, t_{n}) \succ_{A} f(t_{1}, \ldots, s', \ldots, t_{n}) \). \( \square \)

Theorem 4.22 If the interpretation \( f_{A} \) of every function symbol \( f \) is monotone w. r. t. \( \succ \), then \( \succ_{A} \) is a reduction ordering.

Proof. By the previous two lemmas, \( \succ_{A} \) is a rewrite relation. If there were an infinite chain \( s_{1} \succ_{A} s_{2} \succ_{A} \ldots \), then it would correspond to an infinite chain \( A(\beta)(s_{1}) \succ A(\beta)(s_{2}) \succ \ldots \) (with \( \beta \) chosen arbitrarily). Thus \( \succ_{A} \) is well-founded. Irreflexivity and transitivity are proved similarly. \( \square \)

Polynomial Orderings

Polynomial orderings:

Instance of the interpretation method:

The carrier set \( U_{A} \) is \( \mathbb{N} \) or some subset of \( \mathbb{N} \).

To every function symbol \( f/n \) we associate a polynomial \( P_{f}(X_{1}, \ldots, X_{n}) \in \mathbb{N}[X_{1}, \ldots, X_{n}] \) with coefficients in \( \mathbb{N} \) and indeterminates \( X_{1}, \ldots, X_{n} \). Then we define \( f_{A}(a_{1}, \ldots, a_{n}) = P_{f}(a_{1}, \ldots, a_{n}) \) for \( a_{i} \in U_{A} \).

Requirement 1:

If \( a_{1}, \ldots, a_{n} \in U_{A} \), then \( f_{A}(a_{1}, \ldots, a_{n}) \in U_{A} \). (Otherwise, \( \mathcal{A} \) would not be a \( \Sigma \)-algebra.)

Requirement 2:

\( f_{A} \) must be monotone (w. r. t. \( \succ \)).

From now on:

\( U_{A} = \{ n \in \mathbb{N} \mid n \geq 1 \} \).

If \( \text{arity}(f) = 0 \), then \( P_{f} \) is a constant \( \geq 1 \).

If \( \text{arity}(f) = n \geq 1 \), then \( P_{f} \) is a polynomial \( P(X_{1}, \ldots, X_{n}) \), such that every \( X_{i} \) occurs in some monomial \( m \cdot X_{i}^{j_{i}} \cdots X_{k}^{j_{k}} \) with exponent at least 1 and non-zero coefficient \( m \in \mathbb{N} \).

\( \Rightarrow \) Requirements 1 and 2 are satisfied.

116
The mapping from function symbols to polynomials can be extended to terms: A term $t$ containing the variables $x_1, \ldots, x_n$ yields a polynomial $P_t$ with indeterminates $X_1, \ldots, X_n$ (where $X_i$ corresponds to $\beta(x_i)$).

Example:

$$
\begin{align*}
\Omega &= \{b/0, f/1, g/3\} \\
P_b &= 3, \quad P_f(X_1) = X_1^2, \quad P_g(X_1, X_2, X_3) = X_1 + X_2X_3.
\end{align*}
$$

Let $t = g(f(b), f(x), y)$, then $P_t(X, Y) = 9 + X^2Y$.

If $P, Q$ are polynomials in $\mathbb{N}[X_1, \ldots, X_n]$, we write $P > Q$ if $P(a_1, \ldots, a_n) > Q(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in U_A$.

Clearly, $s \succ_A t$ iff $P_s > P_t$ iff $P_s - P_t > 0$.

Question: Can we check $P_s - P_t > 0$ automatically?

**Hilbert’s 10th Problem:**

Given a polynomial $P \in \mathbb{Z}[X_1, \ldots, X_n]$ with integer coefficients, is $P = 0$ for some $n$-tuple of natural numbers?

**Theorem 4.23** Hilbert’s 10th Problem is undecidable.

**Proposition 4.24** Given a polynomial interpretation and two terms $s, t$, it is undecidable whether $P_s > P_t$.

**Proof.** By reduction of Hilbert’s 10th Problem. \qed

One easy case:

If we restrict to linear polynomials, deciding whether $P_s - P_t > 0$ is trivial:

$$
\sum k_ia_i + k > 0 \text{ for all } a_1, \ldots, a_n \geq 1 \text{ if and only if }
$$

$$
k_i \geq 0 \text{ for all } i \in \{1, \ldots, n\},
$$

and $\sum k_i + k > 0$

Another possible solution:

Test whether $P_s(a_1, \ldots, a_n) > P_t(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in \{x \in \mathbb{R} \mid x \geq 1\}$.

This is decidable (but hard). Since $U_A \subseteq \{x \in \mathbb{R} \mid x \geq 1\}$, it implies $P_s > P_t$.

Alternatively:

Use fast overapproximations.
**Simplification Orderings**

The proper subterm ordering $\triangleright$ is defined by $s \triangleright t$ if and only if $s|_p = t$ for some position $p \neq \varepsilon$ of $s$.

A rewrite ordering $\succ$ over $T_\Sigma(X)$ is called simplification ordering, if it has the subterm property: $s \triangleright t$ implies $s \succ t$ for all $s, t \in T_\Sigma(X)$.

Example:

Let $R_{\text{emb}}$ be the rewrite system $R_{\text{emb}} = \{ f(x_1, \ldots, x_n) \rightarrow x_i \mid f/n \in \Omega, 1 \leq i \leq n \}$. Define $\triangleright_{\text{emb}} = \rightarrow^+_{R_{\text{emb}}}$ and $\succeq_{\text{emb}} = \rightarrow^*_{R_{\text{emb}}}$ ("homeomorphic embedding relation"). $\triangleright_{\text{emb}}$ is a simplification ordering.

**Lemma 4.25** If $\succ$ is a simplification ordering, then $s \triangleright_{\text{emb}} t$ implies $s \succ t$ and $s \succeq_{\text{emb}} t$ implies $s \succeq t$.

**Proof.** Since $\succ$ is transitive and $\succeq$ is transitive and reflexive, it suffices to show that $s \rightarrow_{R_{\text{emb}}} t$ implies $s \succ t$. By definition, $s \rightarrow_{R_{\text{emb}}} t$ if and only if $s = s[l\sigma]$ and $t = t[r\sigma]$ for some rule $l \rightarrow r \in R_{\text{emb}}$. Obviously, $l \triangleright r$ for all rules in $R_{\text{emb}}$, hence $l \succ r$. Since $\succ$ is a rewrite relation, $s = s[l\sigma] \succ s[r\sigma] = t$.

Goal:

Show that every simplification ordering is well-founded (and therefore a reduction ordering).

Note: This works only for finite signatures!

To fix this for infinite signatures, the definition of simplification orderings and the definition of embedding have to be modified.

**Theorem 4.26** ("Kruskal’s Theorem") Let $\Sigma$ be a finite signature, let $X$ be a finite set of variables. Then for every infinite sequence $t_1, t_2, t_3, \ldots$ there are indices $j > i$ such that $t_j \succeq_{\text{emb}} t_i$. ($\succeq_{\text{emb}}$ is called a well-partial-ordering (wpo).)

**Proof.** See Baader and Nipkow, page 113–115.
**Theorem 4.27 (Dershowitz)** If $\Sigma$ is a finite signature, then every simplification ordering $\succ$ on $T_\Sigma(X)$ is well-founded (and therefore a reduction ordering).

**Proof.** Suppose that $t_1 \succ t_2 \succ t_3 \succ \ldots$ is an infinite descending chain.

First assume that there is an $x \in \text{var}(t_{i+1}) \setminus \text{var}(t_i)$. Let $\sigma = \{x \mapsto t_i\}$, then $t_{i+1}\sigma \succeq x\sigma = t_i$ and therefore $t_i = t_i\sigma \succ t_{i+1}\sigma \succeq t_i$, contradicting irreflexivity.

Consequently, $\text{var}(t_i) \supseteq \text{var}(t_{i+1})$ and $t_i \in T_\Sigma(V)$ for all $i$, where $V$ is the finite set $\text{var}(t_1)$. By Kruskal’s Theorem, there are $i < j$ with $t_i \preceq_{\text{emb}} t_j$. Hence $t_i \preceq t_j$, contradicting $t_i \succ t_j$. $\square$

There are reduction orderings that are not simplification orderings and terminating TRSs that are not contained in any simplification ordering.

**Example:**

Let $R = \{f(f(x)) \rightarrow f(g(f(x)))\}$.

$R$ terminates and $\rightarrow_R$ is therefore a reduction ordering.

Assume that $\rightarrow_R$ were contained in a simplification ordering $\succ$. Then $f(f(x)) \rightarrow_R f(g(f(x)))$ implies $f(f(x)) \succ f(g(f(x)))$, and $f(g(f(x))) \preceq_{\text{emb}} f(f(x))$ implies $f(g(f(x))) \succeq f(f(x))$, hence $f(f(x)) \succ f(f(x))$.

**Path Orderings**

Let $\Sigma = (\Omega, \Pi)$ be a finite signature, let $\succ$ be a strict partial ordering (“precedence”) on $\Omega$.

The **lexicographic path ordering** $\succ_{\text{lo}}$ on $T_\Sigma(X)$ induced by $\succ$ is defined by: $s \succ_{\text{lo}} t$ iff

1. $t \in \text{var}(s)$ and $t \neq s$, or
2. $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and
   (a) $s_i \succeq_{\text{lo}} t$ for some $i$, or
   (b) $f \succ g$ and $s \succ_{\text{lo}} t_j$ for all $j$, or
   (c) $f = g$, $s \succ_{\text{lo}} t_j$ for all $j$, and $(s_1, \ldots, s_m) \succ_{\text{lo}}(t_1, \ldots, t_n)$.

where $(\succ_{\text{lo}})_{\text{lex}}$ is the $m$-fold lexicographic combination of $\succ_{\text{lo}}$ (note that $f = g$ implies $m = n$).

**Lemma 4.28** $s \succ_{\text{lo}} t$ implies $\text{var}(s) \supseteq \text{var}(t)$.

**Proof.** By induction on $|s| + |t|$ and case analysis. $\square$
Theorem 4.29 \( \succ_{lpo} \) is a simplification ordering on \( T_\Sigma(X) \).

**Proof.** Show transitivity, subterm property, stability under substitutions, compatibility with \( \Sigma \)-operations, and irreflexivity, usually by induction on the sum of the term sizes and case analysis. Details: Baader and Nipkow, page 119/120. \( \Box \)

Theorem 4.30 If the precedence \( \succ \) is total, then the lexicographic path ordering \( \succ_{lpo} \) is total on ground terms, i.e., for all \( s, t \in T_\Sigma(\emptyset) \): \( s \succ_{lpo} t \lor t \succ_{lpo} s \lor s = t \).

**Proof.** By induction on \(|s| + |t|\) and case analysis. \( \Box \)

Recapitulation:

Let \( \Sigma = (\Omega, \Pi) \) be a finite signature, let \( \succ \) be a strict partial ordering ("precedence") on \( \Omega \). The lexicographic path ordering \( \succ_{lpo} \) on \( T_\Sigma(X) \) induced by \( \succ \) is defined by: \( s \succ_{lpo} t \) iff

1. \( t \in \text{var}(s) \) and \( t \neq s \), or
2. \( s = f(s_1, \ldots, s_m), t = g(t_1, \ldots, t_n) \), and
   - (a) \( s_i \succeq_{lpo} t \) for some \( i \), or
   - (b) \( f \succ g \) and \( s \succ_{lpo} t_\pi \) for all \( j \), or
   - (c) \( f = g, s \succ_{lpo} t_\pi \) for all \( j \), and \( (s_1, \ldots, s_m) \succeq_{lpo} \text{lex} (t_1, \ldots, t_n) \).

There are several possibilities to compare subterms in (2)(c):

- compare list of subterms lexicographically left-to-right ("lexicographic path ordering (lpo)", Kamin and Lévy)
- compare list of subterms lexicographically right-to-left (or according to some permutation \( \pi \))
- compare multiset of subterms using the multiset extension ("multiset path ordering (mpo)", Dershowitz)
- to each function symbol \( f/n \in \Omega \) with \( n \geq 1 \) associate a status \( \in \{\text{mul}\} \cup \{\text{lex}_\pi \mid \pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}\} \) and compare according to that status ("recursive path ordering (rpo) with status")
The Knuth-Bendix Ordering

Let $\Sigma = (\Omega, \Pi)$ be a finite signature, let $\succ$ be a strict partial ordering ("precedence") on $\Omega$, let $w : \Omega \cup X \rightarrow \mathbb{R}_0^+$ be a weight function, such that the following admissibility conditions are satisfied:

$$w(x) = w_0 \in \mathbb{R}^+ \text{ for all variables } x \in X; \ w(c) \geq w_0 \text{ for all constants } c \in \Omega.$$  

If $w(f) = 0$ for some $f/1 \in \Omega$, then $f \succ g$ for all $g/n \in \Omega$ with $f \neq g$.

The weight function $w$ can be extended to terms recursively:

$$w(f(t_1, \ldots, t_n)) = w(f) + \sum_{1 \leq i \leq n} w(t_i)$$

or alternatively

$$w(t) = \sum_{x \in \text{var}(t)} w(x) \cdot \#(x, t) + \sum_{f \in \Omega} w(f) \cdot \#(f, t).$$

where $\#(a, t)$ is the number of occurrences of $a$ in $t$.

The Knuth-Bendix ordering $\succ_{\text{kbo}}$ on $T_{\Sigma}(X)$ induced by $\succ$ and $w$ is defined by: $s \succ_{\text{kbo}} t$ iff

1. $\#(x, s) \geq \#(x, t)$ for all variables $x$ and $w(s) > w(t)$, or
2. $\#(x, s) \geq \#(x, t)$ for all variables $x$, $w(s) = w(t)$, and
   - (a) $t = x$, $s = f^n(x)$ for some $n \geq 1$, or
   - (b) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and $f \succ g$, or
   - (c) $s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_n)$, and $(s_1, \ldots, s_m) \succ_{\text{kbo}\text{lex}} (t_1, \ldots, t_m)$.

**Theorem 4.31** The Knuth-Bendix ordering induced by $\succ$ and $w$ is a simplification ordering on $T_{\Sigma}(X)$.

**Proof.** Baader and Nipkow, pages 125–129. $\square$

**Remark**

If $\Pi \neq \emptyset$, then all the term orderings described in this section can also be used to compare non-equational atoms by treating predicate symbols like function symbols.
4.6 Knuth-Bendix Completion

**Completion:**

Goal: Given a set $E$ of equations, transform $E$ into an equivalent convergent set $R$ of
rewrite rules.
(If $R$ is finite: decision procedure for $E$.)

**Knuth-Bendix Completion: Idea**

How to ensure termination?

Fix a reduction ordering $\succ$ and construct $R$ in such a way that $\rightarrow_R \subseteq \succ$ (i.e., $l \succ r$
for every $l \rightarrow r \in R$).

How to ensure confluence?

Check that all critical pairs are joinable.

Note: Every critical pair $\langle s, t \rangle$ can be made joinable by adding $s \rightarrow t$ or $t \rightarrow s$ to $R$.
(Actually, we first add $s \approx t$ to $E$ and later try to turn it into a rule that is contained
in $\succ$; this gives us some additional degree of freedom.)

**Knuth-Bendix Completion: Inference Rules**

The completion procedure is presented as a set of inference rules working on a set of
equations $E$ and a set of rules $R$: $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$

At the beginning, $E = E_0$ is the input set and $R = R_0$ is empty. At the end, $E$ should
be empty; then $R$ is the result.

For each step $E, R \vdash E', R'$, the equational theories of $E \cup R$ and $E' \cup R'$ agree: $\approx_{E \cup R} = \approx_{E' \cup R'}$.

Notations:

The formula $s \approx t$ denotes either $s \approx t$ or $t \approx s$.

$\text{CP}(R)$ denotes the set of all critical pairs between rules in $R$. 

122
Orient:
\[
\frac{E \cup \{s \approx t\}, R}{E, R \cup \{s \rightarrow t\}} \quad \text{if } s \succ t
\]
Note: There are equations \( s \approx t \) that cannot be oriented, i.e., neither \( s \succ t \) nor \( t \succ s \).

Trivial equations cannot be oriented – but we don’t need them anyway:
Delete:
\[
\frac{E \cup \{s \approx s\}, R}{E, R}
\]

Critical pairs between rules in \( R \) are turned into additional equations:
Deduce:
\[
\frac{E, R}{E \cup \{s \approx t\}, R} \quad \text{if } \langle s, t \rangle \in \text{CP}(R).
\]
Note: If \( \langle s, t \rangle \in \text{CP}(R) \) then \( s \leftarrow_R u \rightarrow_R t \) and hence \( R \models s \approx t \).

The following inference rules are not absolutely necessary, but very useful (e.g., to get rid of joinable critical pairs and to deal with equations that cannot be oriented):
Simplify-Eq:
\[
\frac{E \cup \{s \approx t\}, R}{E \cup \{u \approx t\}, R} \quad \text{if } s \rightarrow_R u.
\]

Simplification of the right-hand side of a rule is unproblematic:
R-Simplify-Rule:
\[
\frac{E, R \cup \{s \rightarrow t\}}{E, R \cup \{s \rightarrow u\}} \quad \text{if } t \rightarrow_R u.
\]

Simplification of the left-hand side may influence orientability and orientation. Therefore, it yields an equation:
L-Simplify-Rule:
\[
\frac{E, R \cup \{s \rightarrow t\}}{E \cup \{u \approx t\}, R} \quad \text{if } s \rightarrow_R u \text{ using a rule } l \rightarrow r \in R
\]
such that \( s \sqsubseteq l \) (see below).
For technical reasons, the lhs of $s \rightarrow t$ may only be simplified using a rule $l \rightarrow r$, if $l \rightarrow r$ cannot be simplified using $s \rightarrow t$, that is, if $s \sqsubseteq l$, where the encompassment quasi-ordering $\sqsubseteq$ is defined by

$$s \sqsubseteq l \text{ if } s|_p = l\sigma \text{ for some } p \text{ and } \sigma$$

and $\sqsubset = \sqsubseteq \setminus \sqsubseteq$ is the strict part of $\sqsubseteq$.

**Lemma 4.32** $\sqsubset$ is a well-founded strict partial ordering.

**Lemma 4.33** If $E, R \vdash E', R'$, then $\approx_{E \cup R} = \approx_{E' \cup R'}$.

**Lemma 4.34** If $E, R \vdash E', R'$ and $\rightarrow_R \subseteq \succ$, then $\rightarrow_{R'} \subseteq \succ$.

Note: Like in ordered resolution, simplification should be preferred to deduction:

- Simplify/delete whenever possible.
- Otherwise, orient an equation, if possible.
- Last resort: compute critical pairs.

**Knuth-Bendix Completion: Correctness Proof**

If we run the completion procedure on a set $E$ of equations, different things can happen:

1. We reach a state where no more inference rules are applicable and $E$ is not empty.
   
   $\Rightarrow$ Failure (try again with another ordering?)

2. We reach a state where $E$ is empty and all critical pairs between the rules in the current $R$ have been checked.

3. The procedure runs forever.

In order to treat these cases simultaneously, we need some definitions.

A (finite or infinite sequence) $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$ with $R_0 = \emptyset$ is called a run of the completion procedure with input $E_0$ and $\succ$.

For a run, $E_\infty = \bigcup_{i \geq 0} E_i$ and $R_\infty = \bigcup_{i \geq 0} R_i$.

The sets of persistent equations or rules of the run are $E_* = \bigcup_{i \geq 0} \bigcap_{j \geq i} E_j$ and $R_* = \bigcup_{i \geq 0} \bigcap_{j \geq i} R_j$.

Note: If the run is finite and ends with $E_n, R_n$, then $E_* = E_n$ and $R_* = R_n$. 

124
A run is called fair, if \( CP(R_s) \subseteq E_\infty \) (i.e., if every critical pair between persisting rules is computed at some step of the derivation).

Goal:

Show: If a run is fair and \( E_* \) is empty, then \( R_* \) is convergent and equivalent to \( E_0 \).

In particular: If a run is fair and \( E_* \) is empty, then

\[
\approx E_0 = \approx E_\infty \cup R_\infty = \leftrightarrow E_\infty \cup R_\infty = \downarrow R_* .
\]

General assumptions from now on:

\( E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots \) is a fair run.

\( R_0 \) and \( E_* \) are empty.

A proof of \( s \approx t \) in \( E_\infty \cup R_\infty \) is a finite sequence \((s_0, \ldots, s_n)\) such that \( s = s_0 \), \( t = s_n \), and for all \( i \in \{1, \ldots, n\} \):

1. \( s_{i-1} \leftrightarrow_{E_\infty} s_i \), or
2. \( s_{i-1} \rightarrow_{R_\infty} s_i \), or
3. \( s_{i-1} \leftarrow_{R_\infty} s_i \).

The pairs \((s_{i-1}, s_i)\) are called proof steps.

A proof is called a rewrite proof in \( R_* \), if there is a \( k \in \{0, \ldots, n\} \) such that

\( s_{i-1} \rightarrow_{R_*} s_i \) for \( 1 \leq i \leq k \) and \( s_{i-1} \leftarrow_{R_*} s_i \) for \( k + 1 \leq i \leq n \)

Idea (Bachmair, Dershowitz, Hsiang):

Define a well-founded ordering on proofs, such that for every proof that is not a rewrite proof in \( R_* \) there is an equivalent smaller proof.

Consequence: For every proof there is an equivalent rewrite proof in \( R_* \).

We associate a cost \( c(s_{i-1}, s_i) \) with every proof step as follows:

1. If \( s_{i-1} \leftrightarrow_{E_\infty} s_i \), then \( c(s_{i-1}, s_i) = (\{s_{i-1}, s_i\}, -, -) \), where the first component is a multiset of terms and \(-\) denotes an arbitrary (irrelevant) term.
2. If \( s_{i-1} \rightarrow_{R_\infty} s_i \), then \( c(s_{i-1}, s_i) = (\{s_{i-1}\}, l, s_i) \).
3. If \( s_{i-1} \leftarrow_{R_\infty} s_i \), then \( c(s_{i-1}, s_i) = (\{s_i\}, l, s_{i-1}) \).

Proof steps are compared using the lexicographic combination of the multiset extension of the reduction ordering \( \succ \), the encompassment ordering \( \sqsupset \), and the reduction ordering \( \succ \).

The cost \( c(P) \) of a proof \( P \) is the multiset of the costs of its proof steps.

The proof ordering \( \succ_C \) compares the costs of proofs using the multiset extension of the proof step ordering.
Lemma 4.35 $\succ_C$ is a well-founded ordering.

Lemma 4.36 Let $P$ be a proof in $E_\infty \cup R_\infty$. If $P$ is not a rewrite proof in $R_\ast$, then there exists an equivalent proof $P'$ in $E_\infty \cup R_\infty$ such that $P \succ_C P'$.

Proof. If $P$ is not a rewrite proof in $R_\ast$, then it contains

(a) a proof step that is in $E_\infty$, or
(b) a proof step that is in $R_\infty \setminus R_\ast$, or
(c) a subproof $s_{i-1} \leftarrow_{R_\ast} s_i \rightarrow_{R_\ast} s_{i+1}$ (peak).

We show that in all three cases the proof step or subproof can be replaced by a smaller subproof:

Case (a): A proof step using an equation $s \approx t$ is in $E_\infty$. This equation must be deleted during the run.

If $s \approx t$ is deleted using Orient:
\[ \ldots s_{i-1} \leftrightarrow_{E_\infty} s_i \ldots \implies \ldots s_{i-1} \rightarrow_{R_\infty} s_i \ldots \]

If $s \approx t$ is deleted using Delete:
\[ \ldots s_{i-1} \leftrightarrow_{E_\infty} s_i \ldots \implies \ldots s_{i-1} \ldots \]

If $s \approx t$ is deleted using Simplify-Eq:
\[ \ldots s_{i-1} \leftrightarrow_{E_\infty} s_i \ldots \implies \ldots s_{i-1} \rightarrow_{R_\infty} s' \leftrightarrow_{E_\infty} s_i \ldots \]

Case (b): A proof step using a rule $s \rightarrow t$ is in $R_\infty \setminus R_\ast$. This rule must be deleted during the run.

If $s \rightarrow t$ is deleted using $R$-Simplify-Rule:
\[ \ldots s_{i-1} \rightarrow_{R_\infty} s_i \ldots \implies \ldots s_{i-1} \rightarrow_{R_\ast} s' \leftrightarrow_{E_\infty} s_i \ldots \]

If $s \rightarrow t$ is deleted using $L$-Simplify-Rule:
\[ \ldots s_{i-1} \rightarrow_{R_\infty} s_i \ldots \implies \ldots s_{i-1} \rightarrow_{R_\ast} s' \leftrightarrow_{E_\infty} s_i \ldots \]

Case (c): A subproof has the form $s_{i-1} \leftarrow_{R_\ast} s_i \rightarrow_{R_\ast} s_{i+1}$.

If there is no overlap or a non-critical overlap:
\[ \ldots s_{i-1} \leftarrow_{R_\ast} s_i \rightarrow_{R_\ast} s_{i+1} \ldots \implies \ldots s_{i-1} \rightarrow_{R_\ast} s' \leftrightarrow_{E_\ast} s_{i+1} \ldots \]

If there is a critical pair that has been added using Deduce:
\[ \ldots s_{i-1} \leftarrow_{R_\ast} s_i \rightarrow_{R_\ast} s_{i+1} \ldots \implies \ldots s_{i-1} \leftrightarrow_{E_\ast} s_{i+1} \ldots \]

In all cases, checking that the replacement subproof is smaller than the replaced subproof is routine. $\square$
Theorem 4.37 Let $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdots$ be a fair run and let $R_0$ and $E_\ast$ be empty. Then

1. every proof in $E_\infty \cup R_\infty$ is equivalent to a rewrite proof in $R_\ast$,
2. $R_\ast$ is equivalent to $E_0$, and
3. $R_\ast$ is convergent.

Proof. (1) By well-founded induction on $\succ_C$ using the previous lemma.

(2) Clearly $\approx_{E_\infty \cup R_\infty} = \approx_{E_0}$. Since $R_\ast \subseteq R_\infty$, we get $\approx_{R_\ast} \subseteq \approx_{E_\infty \cup R_\infty}$. On the other hand, by (1), $\approx_{E_\infty \cup R_\infty} \subseteq \approx_{R_\ast}$.

(3) Since $\rightarrow_{R_\ast} \subseteq \succ$, $R_\ast$ is terminating. By (1), $R_\ast$ is confluent. $\square$

4.7 Unfailing Completion

Classical completion:

Try to transform a set $E$ of equations into an equivalent convergent TRS.

Fail, if an equation can neither be oriented nor deleted.

Unfailing completion (Bachmair, Dershowitz and Plaisted):

If an equation cannot be oriented, we can still use orientable instances for rewriting.

Note: If $\succ$ is total on ground terms, then every ground instance of an equation is trivial or can be oriented.

Goal: Derive a ground convergent set of equations.

Let $E$ be a set of equations, let $\succ$ be a reduction ordering.

We define the relation $\rightarrow_{E\succ}$ by

$s \rightarrow_{E\succ} t$ iff there exist $(u \approx v) \in E$ or $(v \approx u) \in E$,

$p \in \text{pos}(s)$, and $\sigma : X \rightarrow T_\Sigma(X)$,

such that $s|_p = u\sigma$ and $t = s[v\sigma]|_p$ and $u\sigma \succ v\sigma$.

Note: $\rightarrow_{E\succ}$ is terminating by construction.

From now on let $\succ$ be a reduction ordering that is total on ground terms.

$E$ is called ground convergent w.r.t. $\succ$, if for all ground terms $s$ and $t$ with $s \leftrightarrow_{E\succ}^* t$ there exists a ground term $v$ such that $s \rightarrow_{E\succ}^* v \leftarrow_{E\succ}^* t$. (Analogously for $E \cup R$.)

As for standard completion, we establish ground convergence by computing critical pairs.
However, the ordering $\succ$ is not total on non-ground terms. Since $s\theta \succ t\theta$ implies $s \not\preceq t$, we approximate $\succ$ on ground terms by $\not\preceq$ on arbitrary terms.

Let $u_i \approx v_i$ ($i = 1, 2$) be equations in $E$ whose variables have been renamed such that $\text{var}(u_1 \approx v_1) \cap \text{var}(u_2 \approx v_2) = \emptyset$. Let $p \in \text{pos}(u_1)$ be a position such that $u_1|_p$ is not a variable, $\sigma$ is an mgu of $u_1|_p$ and $u_2$, and $u_i\sigma \not\preceq v_i\sigma$ ($i = 1, 2$). Then $\langle v_1\sigma, (u_1\sigma)[v_2\sigma]|_p \rangle$ is called a semi-critical pair of $E$ with respect to $\succ$.

The set of all semi-critical pairs of $E$ is denoted by $\text{SP}_{\succ}(E)$.

Semi-critical pairs of $E \cup R$ are defined analogously. If $\rightarrow_R \subseteq \succ$, then $\text{CP}(R)$ and $\text{SP}_{\succ}(R)$ agree.

Note: In contrast to critical pairs, it may be necessary to consider overlaps of an equation with itself at the top. For instance, if $E = \{f(x) \approx g(y)\}$, then $\langle g(y), g(y') \rangle$ is a non-trivial semi-critical pair.

The Deduce rule takes now the following form:

Deduce:

\[
\frac{E_0, R}{E \cup \{s \approx t\}, R} \text{ if } \langle s, t \rangle \in \text{SP}_{\succ}(E \cup R).
\]

Moreover, the fairness criterion for runs is replaced by

\[
\text{SP}_{\succ}(E_0 \cup R_0) \subseteq E_\infty
\]

(i.e., if every semi-critical pair between persisting rules or equations is computed at some step of the derivation).

Analogously to Thm. 4.37 we obtain now the following theorem:

**Theorem 4.38** Let $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$ be a fair run; let $R_0 = \emptyset$. Then

1. $E_0 \cup R_0$ is equivalent to $E_0$, and
2. $E_0 \cup R_0$ is ground convergent.

Moreover one can show that, whenever there exists a reduced convergent $R$ such that $\approx_{E_0} = \downarrow_R$ and $\rightarrow_R \subseteq \succ$, then for every fair and simplifying run $E_\ast = \emptyset$ and $R_\ast = R$ up to variable renaming.

Here $R$ is called reduced, if for every $l \rightarrow r \in R$, both $l$ and $r$ are irreducible w.r.t. $R \setminus \{l \rightarrow r\}$. A run is called simplifying, if $R_\ast$ is reduced, and for all equations $u \approx v \in E_\ast$, $u$ and $v$ are incomparable w.r.t. $\succ$ and irreducible w.r.t. $R_\ast$.

Unfailing completion is refutationally complete for equational theories:
Theorem 4.39 Let $E$ be a set of equations, let $\succ$ be a reduction ordering that is total on ground terms. For any two terms $s$ and $t$, let $\hat{s}$ and $\hat{t}$ be the terms obtained from $s$ and $t$ by replacing all variables by Skolem constants. Let $eq/2$, $true/0$ and $false/0$ be new operator symbols, such that $true$ and $false$ are smaller than all other terms. Let $E_0 = E \cup \{eq(\hat{s}, \hat{t}) \approx true, eq(x, x) \approx false\}$. If $E_0, \emptyset \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$ be a fair run of unfailing completion, then $s \approx_E t$ iff some $E_i \cup R_i$ contains $true \approx false$.

Outlook:

Combine ordered resolution and unfailing completion to get a calculus for equational clauses:

compute inferences between (strictly) maximal literals as in ordered resolution,
compute overlaps between maximal sides of equations as in unfailing completion

$\Rightarrow$ Superposition calculus.
5 Termination Revisited

So far: Termination as a subordinate task for entailment checking.

TRS is generated by some saturation process; ordering must be chosen before the saturation starts.

Now: Termination as a main task (e.g., for program analysis).

TRS is fixed and known in advance.

Literature:


5.1 Dependency Pairs

Invented by T. Arts and J. Giesl in 1996, many refinements since then.

Given: finite TRS $R$ over $\Sigma = (\Omega, \emptyset)$.

$T_0 := \{ t \in T_\Sigma(X) \mid \text{there is an infinite derivation } t \rightarrow_R t_1 \rightarrow_R t_2 \rightarrow_R \ldots \}$.

$T_\infty := \{ t \in T_0 \mid \forall p > \varepsilon : t|_p \notin T_0 \} = \text{minimal elements of } T_0 \text{ w.r.t. } \triangleright$.

$t \in T_0 \Rightarrow \text{there exists a } t' \in T_\infty \text{ such that } t \triangleright t'$.

$R$ is non-terminating iff $T_0 \neq \emptyset$ iff $T_\infty \neq \emptyset$.

Assume that $T_\infty \neq \emptyset$ and consider some non-terminating derivation starting from $t \in T_\infty$.

Since all subterms of $t$ allow only finite derivations, at some point a rule $l \rightarrow r \in R$ must be applied at the root of $t$ (possibly preceded by rewrite steps below the root):

$t = f(t_1, \ldots, t_n) \xrightarrow{\varepsilon} f(s_1, \ldots, s_n) = l\sigma \xrightarrow{\varepsilon} r\sigma$.

In particular, root($t$) = root($l$), so we see that the root symbol of any term in $T_\infty$ must be contained in $D := \{ \text{root}(l) \mid l \rightarrow r \in R \}$. $D$ is called the set of defined symbols of $R$; $C := \Omega \setminus D$ is called the set of constructor symbols of $R$.

The term $r\sigma$ is contained in $T_0$, so there exists a $v \in T_\infty$ such that $r\sigma \triangleright v$.

If $v$ occurred in $r\sigma$ at or below a variable position of $r$, then $x\sigma|_{x} = v$ for some $x \in \text{var}(r) \subseteq \text{var}(l)$, hence $s_i \triangleright x\sigma$ and there would be an infinite derivation starting from some $t_i$. This contradicts $t \in T_\infty$, though.
Therefore, $v = u\sigma$ for some non-variable subterm $u$ of $r$. As $v \in T_\infty$, we see that $\text{root}(u) = \text{root}(v) \in D$. Moreover, $u$ cannot be a proper subterm of $l$, since otherwise again there would be an infinite derivation starting from some $t_i$.

Putting everything together, we obtain

$$t = f(t_1, \ldots, t_n) \xrightarrow{\varepsilon_R} f(s_1, \ldots, s_n) = l\sigma \xrightarrow{\varepsilon} R r\sigma \supseteq u\sigma$$

where $r \supseteq u$, $u$ is not a variable, $\text{root}(u) \in D$, $l \not\supset u$.

Since $u\sigma \in T_\infty$, we can continue this process and obtain an infinite sequence.

If we define $S := \{ l \rightarrow u \mid l \rightarrow r \in R, r \supseteq u, u \not\in X, \text{root}(u) \in D, l \not\supset u \}$, we can combine the rewrite step at the root and the subterm step and obtain

$$t \xrightarrow{\varepsilon_R} l\sigma \xrightarrow{\varepsilon} S u\sigma.$$  

To get rid of the superscripts $\varepsilon$ and $>\varepsilon$, it turns out to be useful to introduce a new set of function symbols $f^\sharp$ that are only used for the root symbols of this derivation:

$$\Omega^\sharp := \{ f^\sharp/n \mid f/n \in \Omega \}.$$  

For a term $t = f(t_1, \ldots, t_n)$ we define $t^\sharp := f^\sharp(t_1, \ldots, t_n)$; for a set of terms $T$ we define $T^\sharp := \{ t^\sharp \mid t \in T \}$.

The set of dependency pairs of a TRS $R$ is then defined by

$$\text{DP}(R) := \{ l^\sharp \rightarrow u^\sharp \mid l \rightarrow r \in R, r \supseteq u, u \not\in X, \text{root}(u) \in D, l \not\supset u \}.$$  

For $t \in T_\infty$, the sequence using the $S$-rule corresponds now to

$$t^\sharp \rightarrow^* S l\sigma \rightarrow_{\text{DP}(R)} u^\sharp\sigma$$

where $t^\sharp \in T_\infty^\sharp$ and $u^\sharp\sigma \in T_\infty^\sharp$.

(Note that rules in $R$ do not contain symbols from $\Omega^\sharp$, whereas all roots of terms in $\text{DP}(R)$ come from $\Omega^\sharp$, so rules from $R$ can only be applied below the root and rules from $\text{DP}(R)$ can only be applied at the root.)

Since $u^\sharp\sigma$ is again in $T_\infty^\sharp$, we can continue the process in the same way. We obtain: $R$ is non-terminating iff there is an infinite sequence

$$t_1 \rightarrow^* R t_2 \rightarrow_{\text{DP}(R)} t_3 \rightarrow^* R t_4 \rightarrow_{\text{DP}(R)} \ldots$$

with $t_i \in T_\infty^\sharp$ for all $i$.

Moreover, if there exists such an infinite sequence, then there exists an infinite sequence in which all DPs that are used are used infinitely often. (If some DP is used only finitely often, we can cut off the initial part of the sequence up to the last occurrence of that DP; the remainder is still an infinite sequence.)
Dependency Graphs

Such infinite sequences correspond to “cycles” in the “dependency graph”:

**Dependency graph** $DG(R)$ of a TRS $R$:

- directed graph
  - nodes: dependency pairs $s \rightarrow t \in DP(R)$
  - edges: from $s \rightarrow t$ to $u \rightarrow v$ if there are $\sigma, \tau$ such that $t\sigma \rightarrow^*_R u\tau$.

Intuitively, we draw an edge between two dependency pairs, if these two dependency pairs can be used after another in an infinite sequence (with some $R$-steps in between). While this relation is undecidable in general, there are reasonable overapproximations:

The functions $\text{cap}$ and $\text{ren}$ are defined by:

$$
cap(x) = x
$$

$$
cap(f(t_1, \ldots, t_n)) = \begin{cases} 
y & \text{if } f \in D \\
f(\cap(t_1), \ldots, \cap(t_n)) & \text{if } f \in C \cup D^\sharp
\end{cases}
$$

$$
\text{ren}(x) = y, \ y \text{ fresh}
$$

$$
\text{ren}(f(t_1, \ldots, t_n)) = f(\text{ren}(t_1), \ldots, \text{ren}(t_n))
$$

The overapproximated dependency graph contains an edge from $s \rightarrow t$ to $u \rightarrow v$ if $\text{ren}(\cap(t))$ and $u$ are unifiable.

A cycle in the dependency graph is a non-empty subset $K \subseteq DP(R)$ such that there is a non-empty path in $K$ from every DP in $K$ to every DP in $K$ (the two DPs may be identical).

Let $K \subseteq DP(R)$. An infinite rewrite sequence in $R \cup K$ of the form

$$
t_1 \rightarrow^*_R t_2 \rightarrow_K t_3 \rightarrow^*_R t_4 \rightarrow_K \ldots
$$

with $t_i \in T^\omega_\infty$ is called $K$-minimal, if all rules in $K$ are used infinitely often.

$R$ is non-terminating iff there is a cycle $K \subseteq DP(R)$ and a $K$-minimal infinite rewrite sequence.

### 5.2 Subterm Criterion

Our task is to show that there are no $K$-minimal infinite rewrite sequences.

Suppose that every dependency pair symbol $f^\sharp$ in $K$ has positive arity (i.e., no constants). A simple projection $\pi$ is a mapping $\pi : \Omega^\sharp \rightarrow \mathbb{N}$ such that $\pi(f^\sharp) = i \in \{1, \ldots, \text{arity}(f^\sharp)\}$.

We define $\pi(f^\sharp(t_1, \ldots, t_n)) = t_{\pi(f^\sharp)}$. 

132
Theorem 5.1 (Hirokawa and Middeldorp) Let $K$ be a cycle in $DG(R)$. If there is a simple projection $\pi$ for $K$ such that $\pi(l) \triangleright= \pi(r)$ for every $l \to r \in K$ and $\pi(l) \triangleright \pi(r)$ for some $l \to r \in K$, then there are no $K$-minimal sequences.

Proof. Suppose that $t_1 \to_R^* u_1 \to_K t_2 \to_R^* u_2 \to_K \ldots$ is a $K$-minimal infinite rewrite sequence. Apply $\pi$ to every $t_i$:

Case 1: $u_i \to_K t_{i+1}$. There is an $l \to r \in K$ such that $u_i = l\sigma$, $t_{i+1} = r\sigma$. Then $\pi(u_i) = \pi(l)\sigma$ and $\pi(t_{i+1}) = \pi(r)\sigma$. By assumption, $\pi(l) \triangleright= \pi(r)$. If $\pi(l) = \pi(r)$, then $\pi(u_i) = \pi(t_{i+1})$. If $\pi(l) \triangleright \pi(r)$, then $\pi(u_i) = \pi(l)\sigma \triangleright \pi(r)\sigma = \pi(t_{i+1})$. In particular, $\pi(u_i) \triangleright \pi(t_{i+1})$ for infinitely many $i$ (since every DP is used infinitely often).

Case 2: $t_i \to_R^* u_i$. Then $\pi(t_i) \to_R^* \pi(u_i)$.

By applying $\pi$ to every term in the $K$-minimal infinite rewrite sequence, we obtain an infinite ($\to_R \cup \triangleright$)-sequence containing infinitely many $\triangleright$-steps. Since $\triangleright$ is well-founded, there must also exist infinitely many $\to_R$-steps (otherwise the infinite sequence would have an infinite tail consisting only of $\triangleright$-steps, contradicting well-foundedness.)

Now note that $\triangleright \circ \to_R \subseteq \to_R \circ \triangleright$. Therefore we can commute $\triangleright$-steps and $\to_R$-steps and move all $\to_R$-steps to the front. We obtain an infinite $\to_R$-sequence that starts with $\pi(t_1)$. However $t_1 \triangleright \pi(t_1)$ and $t_1 \in T_{\infty}^R$, so there cannot be an infinite $\to_R$-sequence starting from $\pi(t_1)$. \[\square\]

Problem: The number of cycles in $DG(R)$ can be exponential.

Better method: Analyze strongly connected components (SCCs).

SCC of a graph: maximal subgraph in which there is a non-empty path from every node to every node. (The two nodes can be identical.)

Important property: Every cycle is contained in some SCC.

Idea: Search for a simple projection $\pi$ such that $\pi(l) \triangleright= \pi(r)$ for all DPs $l \to r$ in the SCC. Delete all DPs in the SCC for which $\pi(l) \triangleright \pi(r)$ (by the previous theorem, there cannot be any $K$-minimal infinite rewrite sequences using these DPs). Then re-compute SCCs for the remaining graph and re-start.

No SCCs left $\Rightarrow$ no cycles left $\Rightarrow R$ is terminating.

Example: See Ex. 13 from Hirokawa and Middeldorp.

\[3\text{There are several definitions of SCCs that differ in the treatment of edges from a node to itself.}\]
5.3 Reduction Pairs and Argument Filterings

Goal: Show the non-existence of $K$-minimal infinite rewrite sequences

$t_1 \rightarrow^*_R u_1 \rightarrow_K t_2 \rightarrow^*_R u_2 \rightarrow_K \ldots$

using well-founded orderings.

We observe that the requirements for the orderings used here are less restrictive than for reduction orderings:

- $K$-rules are only used at the top, so we need stability under substitutions, but compatibility with contexts is unnecessary.
- While $\rightarrow_K$-steps should be decreasing, for $\rightarrow_R$-steps it would be sufficient to show that they are not increasing.

This motivates the following definitions:

**Rewrite quasi-ordering** $\succsim$:
reflexive and transitive binary relation, stable under substitutions, compatible with contexts.

**Reduction pair** $(\succsim, \succ)$:
- $\succsim$ is a rewrite quasi-ordering.
- $\succ$ is a well-founded ordering that is stable under substitutions.
- $\succsim$ and $\succ$ are compatible: $\succsim \circ \succ \subseteq \succ$ or $\succ \circ \succsim \subseteq \succ$.

(In practice, $\succ$ is almost always the strict part of the quasi-ordering $\succsim$.)

Clearly, for any reduction ordering $\succ$, $(\succsim, \succ)$ is a reduction pair. More general reduction pairs can be obtained using argument filterings:

**Argument filtering** $\pi$:

\[
\pi : \Omega \cup \Omega^2 \rightarrow \mathbb{N} \cup \text{list of } \mathbb{N}
\]

\[
\pi(f) = \begin{cases} 
    i \in \{1, \ldots, \text{arity}(f)\}, & \text{or} \\
    [i_1, \ldots, i_k], & \text{where } 1 \leq i_1 < \cdots < i_k \leq \text{arity}(f), \ 0 \leq k \leq \text{arity}(f)
\end{cases}
\]

Extension to terms:

\[
\pi(x) = x
\]

\[
\pi(f(t_1, \ldots, t_n)) = \pi(t_i), \text{ if } \pi(f) = i
\]

\[
\pi(f(t_1, \ldots, t_n)) = f'(\pi(t_{i_1}), \ldots, \pi(t_{i_k})), \text{ if } \pi(f) = [i_1, \ldots, i_k],
\]

where $f'/k$ is a new function symbol.
Let \( \succ \) be a reduction ordering, let \( \pi \) be an argument filtering. Define \( s \succ_{\pi} t \) iff \( \pi(s) \succ \pi(t) \) and \( s \succeq_{\pi} t \) iff \( \pi(s) \succeq \pi(t) \).

**Lemma 5.2** \( (\succeq_{\pi}, \succ_{\pi}) \) is a reduction pair.

**Proof.** Follows from the following two properties:

\[ \pi(s\sigma) = \pi(s)\sigma_{\pi}, \]  where \( \sigma_{\pi} \) is the substitution that maps \( x \) to \( \pi(\sigma(x)) \).

\[ \pi(s[u]_{\pi}) = \begin{cases} 
\pi(s), & \text{if } p \text{ does not correspond to any position in } \pi(s) \\
\pi(s)[\pi(u)]_{\pi}[q], & \text{if } p \text{ corresponds to } q \text{ in } \pi(s) 
\end{cases} \]

For interpretation-based orderings (such as polynomial orderings) the idea of “cutting out” certain subterms can be included directly in the definition of the ordering:

**Reduction pairs by interpretation:**

Let \( \mathcal{A} \) be a \( \Sigma \)-algebra; let \( \succ \) be a well-founded strict partial ordering on its universe.

Assume that all interpretations \( f_{\mathcal{A}} \) of function symbols are weakly monotone, i.e., \( a_i \succeq b_i \) implies \( f(a_1, \ldots, a_n) \succeq f(b_1, \ldots, b_n) \) for all \( a_i, b_i \in U_{\mathcal{A}} \).

Define \( s \succeq_{\mathcal{A}} t \) iff \( \mathcal{A}(\beta)(s) \succeq \mathcal{A}(\beta)(t) \) for all assignments \( \beta : X \rightarrow U_{\mathcal{A}} \); define \( s \succ_{\mathcal{A}} t \) iff \( \mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(t) \) for all assignments \( \beta : X \rightarrow U_{\mathcal{A}} \).

Then \( (\succeq_{\mathcal{A}}, \succ_{\mathcal{A}}) \) is a reduction pair.

For polynomial orderings, this definition permits interpretations of function symbols where some variable does not occur at all (e.g., \( P_f(X_1, X_2) = 2X_1 + 1 \) for a binary function symbol). It is no longer required that every variable must occur with some positive coefficient.

**Theorem 5.3 (Arts and Giesl)** Let \( K \) be a cycle in the dependency graph of the TRS \( R \). If there is a reduction pair \( (\succeq, \succ) \) such that

- \( l \succeq r \) for all \( l \rightarrow r \in R \),
- \( l \succeq r \) or \( l \succ r \) for all \( l \rightarrow r \in K \),
- \( l \succ r \) for at least one \( l \rightarrow r \in K \),

then there is no \( K \)-minimal infinite sequence.
Proof. Assume that

\[ t_1 \xrightarrow{R}^* u_1 \xrightarrow{K} t_2 \xrightarrow{R}^* u_2 \xrightarrow{K} \ldots \]

is a $K$-minimal infinite rewrite sequence.

As $l \gtrsim r$ for all $l \rightarrow r \in R$, we obtain $t_i \gtrsim u_i$ by stability under substitutions, compatibility with contexts, reflexivity and transitivity.

As $l \gneq r$ or $l \succ r$ for all $l \rightarrow r \in K$, we obtain $u_i (\gtrsim \cup \succ) t_{i+1}$ by stability under substitutions.

So we get an infinite $(\gtrsim \cup \succ)$-sequence containing infinitely many $\succ$-steps (since every DP in $K$, in particular the one for which $l \succ r$ holds, is used infinitely often).

By compatibility of $\gtrsim$ and $\succ$, we can transform this into an infinite $\succ$-sequence, contradicting well-foundedness. \qed

The idea can be extended to SCCs in the same way as for the subterm criterion:

Search for a reduction pair $(\gtrsim, \succ)$ such that $l \gtrsim r$ for all $l \rightarrow r \in R$ and $l \gtrsim r$ or $l \succ r$ for all DPs $l \rightarrow r$ in the SCC. Delete all DPs in the SCC for which $l \succ r$.

Then re-compute SCCs for the remaining graph and re-start.

Example: Consider the following TRS $R$ from [Arts and Giesl]:

\[
\begin{align*}
\text{minus}(x, 0) & \rightarrow x \\
\text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow 0 \\
\text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y)))
\end{align*}
\]

($R$ is not contained in any simplification ordering, since the left-hand side of rule (4) is embedded in the right-hand side after instantiating $y$ by $s(x)$.)

$R$ has three dependency pairs:

\[
\begin{align*}
\text{minus}^\sharp(s(x), s(y)) & \rightarrow \text{minus}^\sharp(x, y) \\
\text{quot}^\sharp(s(x), s(y)) & \rightarrow \text{quot}^\sharp(\text{minus}(x, y), s(y)) \\
\text{quot}^\sharp(s(x), s(y)) & \rightarrow \text{minus}^\sharp(x, y)
\end{align*}
\]

The dependency graph of $R$ is

\[
\begin{array}{c}
(5) \\
(7) \\
(6)
\end{array}
\]

136
There are exactly two SCCs (and also two cycles). The cycle at (5) can be handled using the subterm criterion with $\pi(\text{minus}^5) = 1$. For the cycle at (6) we can use an argument filtering $\pi$ that maps $\text{minus}$ to 1 and leaves all other function symbols unchanged (that is, $\pi(g) = [1, \ldots, \text{arity}(g)]$ for every $g$ different from $\text{minus}$.) After applying the argument filtering, we compare left and right-hand sides using an LPO with precedence $q_{\text{out}} > s$ (the precedence of other symbols is irrelevant). We obtain $l \succ r$ for (6) and $l \succsim r$ for (1), (2), (3), (4), so the previous theorem can be applied.

**DP Processors**

The methods described so far are particular cases of *DP processors*:

A DP processor

\[
(G, R) \quad (G_1, R_1), \ldots, (G_n, R_n)
\]

takes a graph $G$ and a TRS $R$ as input and produces a set of pairs consisting of a graph and a TRS.

It is sound and complete if there are $K$-minimal infinite sequences for $G$ and $R$ if and only if there are $K$-minimal infinite sequences for at least one of the pairs $(G_i, R_i)$.

Examples:

\[
(G, R) \quad (SCC_1, R), \ldots, (SCC_n, R)
\]

where $SCC_1, \ldots, SCC_n$ are the strongly connected components of $G$.

\[
(G, R) \quad (G \setminus N, R)
\]

if there is an SCC of $G$ and a simple projection $\pi$ such that $\pi(l) \succeq \pi(r)$ for all DPs $l \rightarrow r$ in the SCC, and $N$ is the set of DPs of the SCC for which $\pi(l) \succeq \pi(r)$.

(and analogously for reduction pairs)

**Innermost Termination**

The dependency method can also be used for proving termination of *innermost rewriting*: $s \xrightarrow{i} R t$ if $s \xrightarrow{} R t$ at position $p$ and no rule of $R$ can be applied at a position strictly below $p$. (DP processors for innermost termination are more powerful than for ordinary termination, and for program analysis, innermost termination is usually sufficient.)
6 Implementing Saturation Procedures

Problem:

Refutational completeness is nice in theory, but . . .

. . . it guarantees only that proofs will be found eventually, not that they will be found quickly.

Even though orderings and selection functions reduce the number of possible infer- ences, the search space problem is enormous.

First-order provers “look for a needle in a haystack”: It may be necessary to make some millions of inferences to find a proof that is only a few dozens of steps long.

Coping with Large Sets of Formulas

Consequently:

• We must deal with large sets of formulas.

• We must use efficient techniques to find formulas that can be used as partners in an inference.

• We must simplify/eliminate as many formulas as possible.

• We must use efficient techniques to check whether a formula can be simplified/elim- inated.

Note:

Often there are several competing implementation techniques.

Design decisions are not independent of each other.

Design decisions are not independent of the particular class of problems we want to solve. (FOL without equality/FOL with equality/unit equations, size of the signature, special algebraic properties like AC, etc.)
### 6.1 Term Representations

The obvious data structure for terms: Trees

\[ f(g(x_1), f(g(x_1), x_2)) \]

optionally: (full) sharing

An alternative: Flattens

\[ f(g(x_1), f(g(x_1), x_2)) \]

need more memory;
but: better suited for preorder term traversal and easier memory management.

### 6.2 Index Data Structures

Problem:

For a term \( t \), we want to find all terms \( s \) such that

- \( s \) is an instance of \( t \),
- \( s \) is a generalization of \( t \) (i.e., \( t \) is an instance of \( s \)),
- \( s \) and \( t \) are unifiable,
- \( s \) is a generalization of some subterm of \( t \),
- \( \ldots \)
Requirements:

fast insertion,
fast deletion,
fast retrieval,
small memory consumption.

Note: In applications like functional or logic programming, the requirements are different (insertion and deletion are much less important).

Many different approaches:

• Path indexing
• Discrimination trees
• Substitution trees
• Context trees
• Feature vector indexing
• …

Perfect filtering:

The indexing technique returns exactly those terms satisfying the query.

Imperfect filtering:

The indexing technique returns some superset of the set of all terms satisfying the query.

Retrieval operations must be followed by an additional check, but the index can often be implemented more efficiently.

Frequently: All occurrences of variables are treated as different variables.
Path Indexing

Path indexing:

Paths of terms are encoded in a trie (“retrieval tree”).

A star * represents arbitrary variables.

Example: Paths of $f(g(*, b), *)$: $f.1.g.1.*$

$\quad f.1.g.2.b$

$\quad f.2.*$

Each leaf of the trie contains the set of (pointers to) all terms that contain the respective path.

Example: Path index for \{f(g(d, *), c), g(b, h(c)), f(g(*, c), c), f(b, g(c, b)), f(b, g(*, b)), f(*, c), f(*, g(c, b))\}

Advantages:

Uses little space.

No backtracking for retrieval.

Efficient insertion and deletion.

Good for finding instances.

Disadvantages:

Retrieval requires combining intermediate results for all paths.
Discrimination Trees

Discrimination trees:

Preorder traversals of terms are encoded in a trie.
A star * represents arbitrary variables.
Example: String of $f(g(*, b), *)$: $f.g.*.b.$
Each leaf of the trie contains (a pointer to) the term that is represented by the path.
Example: Discrimination tree for \{ $f(g(d, *), c), g(b, h(c)), f(g(*, c), c), f(b, g(c, b)), f(b, g(*, b)), f(*, c), f(*, g(c, b))$ \}\n
Advantages:

Each leaf yields one term, hence retrieval does not require intersections of intermediate results for all paths.
Good for finding generalizations.

Disadvantages:

Uses more storage than path indexing (due to less sharing).
Uses still more storage, if jump lists are maintained to speed up the search for instances or unifiable terms.
Feature Vector Indexing

Goal:

$C'$ is subsumed by $C$ if $C' = C \sigma \lor D$.

Find all clauses $C'$ for a given $C$ or vice versa.

If $C'$ is subsumed by $C$, then

- $C'$ contains at least as many literals as $C$.
- $C'$ contains at least as many positive literals as $C$.
- $C'$ contains at least as many negative literals as $C$.
- $C'$ contains at least as many function symbols as $C$.
- $C'$ contains at least as many occurrences of $f$ as $C$.
- $C'$ contains at least as many occurrences of $f$ in negative literals as $C$.
- the deepest occurrence of $f$ in $C'$ is at least as deep as in $C$.
- . . .

Idea:

Select a list of these “features”.

Compute the “feature vector” (a list of natural numbers) for each clause and store it in a trie.

When searching for a subsuming clause: Traverse the trie, check all clauses for which all features are smaller or equal. (Stop if a subsuming clause is found.)

When searching for subsumed clauses: Traverse the trie, check all clauses for which all features are larger or equal.

Advantages:

Works on the clause level, rather than on the term level.

Specialized for subsumption testing.

Disadvantages:

Needs to be complemented by other index structure for other operations.
Literature


7 Outlook

7.1 Satisfiability Modulo Theories (SMT)

CDCL checks satisfiability of propositional formulas.

CDCL can also be used for ground first-order formulas without equality:
- Ground first-order atoms are treated like propositional variables.
- Truth values of $P(a), Q(a), Q(f(a))$ are independent.

For ground formulas with equality, independence is lost:
- If $b \approx c$ is true, then $f(b) \approx f(c)$ must also be true.
- Similarly for other theories, e.g. linear arithmetic: $b > 5$ implies $b > 3$.

We can still use CDCL, but we must combine it with a decision procedure for the theory part $T$:
- $M \models_T C$: $M$ and the theory axioms $T$ entail $C$.

New CDCL rules:

$T$-Propagate:
- $M \parallel N \Rightarrow_{CDCL(T)} M L \parallel N$
  - if $M \models_T L$ where $L$ is undefined in $M$ and $L$ or $\overline{L}$ occurs in $N$.

$T$-Learn:
- $M \parallel N \Rightarrow_{CDCL(T)} M \parallel N \cup \{C\}$
  - if $N \models_T C$ and each atom of $C$ occurs in $N$ or $M$.

$T$-Backjump:
- $M L^a M' \parallel N \cup \{C\} \Rightarrow_{CDCL(T)} M L' \parallel N \cup \{C\}$
  - if $M L^a M' \models \neg C$
  - and there is some “backjump clause” $C' \vee L'$ such that $N \cup \{C\} \models_T C' \vee L'$ and $M \models \neg C'$,
  - $L'$ is undefined under $M$, and
  - $L'$ or $\overline{L'}$ occurs in $N$ or in $M L^a M'$.
7.2 Sorted Logics

So far, we have considered only unsorted first-order logic.

In practice, one often considers many-sorted logics:

\[
\text{read} / 2 \quad \text{becomes} \quad \text{read} : \text{array} \times \text{nat} \rightarrow \text{data}.
\]

\[
\text{write} / 3 \quad \text{becomes} \quad \text{write} : \text{array} \times \text{nat} \times \text{data} \rightarrow \text{array}.
\]

Variables: \( x : \text{data} \)

Only one declaration per function/predicate-variable symbol.

All terms, atoms, substitutions must be well-sorted.

Algebras:

Instead of universe \( U_A \), one set per sort: \( \text{array}_A, \text{nat}_A \).

Interpretations of function and predicate symbols correspond to their declarations:

\[
\text{read}_A : \text{array}_A \times \text{nat}_A \rightarrow \text{data}_A
\]

Proof theory, calculi, etc.:

Essentially as in the unsorted case.

More difficult:

Subsorts

Overloading

7.3 Splitting

Tableau-like rule within resolution to eliminate variable-disjoint (positive) disjunctions:

\[
\frac{N \cup \{ C_1 \lor C_2 \}}{N \cup \{ C_1 \} \mid N \cup \{ C_2 \}}
\]

if \( \text{var}(C_1) \cap \text{var}(C_2) = \emptyset \).

Split clauses are smaller and more likely to be usable for simplification.

Splitting tree is explored using intelligent backtracking.
7.4 Integrating Theories into Resolution

Certain kinds of axioms are important in practice, but difficult for theorem provers.

Most important case: equality but also: orderings, (associativity and) commutativity, ...

Idea: Combine ordered resolution and critical pair computation.

Superposition (ground case):

\[
\begin{array}{c}
D' \lor t \approx t' \\
C'' \lor s[t] \approx s'
\end{array}
\]

\[
\frac{D' \lor C' \lor s[t] \approx s'}{}
\]

Superposition (non-ground case):

\[
\begin{array}{c}
D' \lor t \approx t' \\
C'' \lor s[u] \approx s'
\end{array}
\]

\[
\frac{(D' \lor C' \lor s[t'] \approx s')\sigma}{}
\]

where \(\sigma = \text{mgu}(t, u)\) and \(u\) is not a variable.

Advantages:

No variable overlaps (as in KB-completion).

Stronger ordering restrictions: Only overlaps of (strictly) maximal sides of (strictly) maximal literals are required.

Stronger redundancy criteria.

Similarly for orderings:

Ordered chaining:

\[
\begin{array}{c}
D' \lor t' < t \\
C'' \lor s < s'
\end{array}
\]

\[
\frac{(D' \lor C' \lor t' < s')\sigma}{}
\]

where \(\sigma\) is a most general unifier of \(t\) and \(s\).

Integrating other theories:

Black box:

Use external decision procedure.

Easy, but works only under certain restrictions.
White box:

Integrate using specialized inference rules and theory unification.

Hard work.

Often: integrating more theory axioms is better.
## Contents

### 1 Preliminaries
1.1 Mathematical Prerequisites ........................................ 2
1.2 Abstract Reduction Systems ........................................ 3
1.3 Orderings ............................................................ 4
1.4 Multisets ............................................................. 8
1.5 Complexity Theory Prerequisites .................................... 10

### 2 Propositional Logic
2.1 Syntax ............................................................... 12
2.2 Semantics ............................................................ 15
2.3 Models, Validity, and Satisfiability ................................ 16
2.4 Normal Forms ....................................................... 20
2.5 Improving the CNF Transformation ................................. 23
2.6 The DPLL Procedure ................................................ 27
2.7 From DPLL to CDCL .................................................. 29
2.8 Implementing CDCL .................................................. 34
2.9 OBDDs ................................................................. 37
2.10 FRAIGs ............................................................... 43
2.11 Other Calculi ........................................................ 43

### 3 First-Order Logic
3.1 Syntax ............................................................... 44
3.2 Semantics ............................................................ 50
3.3 Models, Validity, and Satisfiability ................................ 52
3.4 Algorithmic Problems ............................................... 54
3.5 Normal Forms and Skolemization ................................... 55
3.6 Getting Skolem Functions with Small Arity ....................... 58
3.7 Herbrand Interpretations ............................................. 61
3.8 Inference Systems and Proofs ....................................... 62
3.9 Ground (or propositional) Resolution ............................... 64
3.10 Refutational Completeness of Resolution ......................... 66
3.11 General Resolution .................................................. 72
3.12 Theoretical Consequences .......................................... 80
3.13 Ordered Resolution with Selection .................................. 81
3.14 Redundancy ........................................................ 87
3.15 Hyperresolution ..................................................... 91
3.16 Implementing Resolution: The Main Loop ......................... 92
3.17 Summary: Resolution Theorem Proving ............................ 93
3.18 Semantic Tableaux ................................................... 93
3.19 Semantic Tableaux for First-Order Logic ......................... 100
3.20 Other Deductive Systems .......................................... 103
4 First-Order Logic with Equality 105
4.1 Handling Equality Naively .......................... 105
4.2 Rewrite Systems .................................. 106
4.3 Confluence ..................................... 110
4.4 Critical Pairs ................................... 112
4.5 Termination ...................................... 114
4.6 Knuth-Bendix Completion ......................... 122
4.7 Unfailing Completion .............................. 127

5 Termination Revisited 130
5.1 Dependency Pairs .................................. 130
5.2 Subterm Criterion ................................. 132
5.3 Reduction Pairs and Argument Filterings ....... 134

6 Implementing Saturation Procedures 138
6.1 Term Representations ............................... 139
6.2 Index Data Structures ............................. 139

7 Outlook 145
7.1 Satisfiability Modulo Theories (SMT) .......... 145
7.2 Sorted Logics .................................... 146
7.3 Splitting ......................................... 146
7.4 Integrating Theories into Resolution .......... 147