Automated Reasoning I

Marek Kosta Christoph Weidenbach

Summer Term 2012

What is Computer Science about?

Theory

Graphics

Data Bases

Programming Languages

Algorithms

Hardware

Bioinformatics

Verification

What is Automated Deduction about?

Generic Problem Solving by a Computer Program.

Introductory Example: Solving 4×4 Sudoku

2	1		
		3	1
1		2	

Start

Introductory Example: Solving 4×4 Sudoku

2	1	4	3
3	4	1	2
4	2	3	1
1	3	2	4

Solution

Formal Model

Represent board by a function $f(x, y)$ mapping cells to their value.

2	1		
		3	1
1		2	

Start

$$N = f(1,1) \approx 2 \wedge f(1,2) \approx 1 \wedge$$
$$f(3,3) \approx 3 \wedge f(3,4) \approx 1 \wedge$$
$$f(4,1) \approx 1 \wedge f(4,3) \approx 2$$

$\wedge$ is conjunction and $\top$ the empty conjunction.

Formal Model

A state is described by a triple $(N; D; r)$ where

- N contains the equations for the starting Sudoku
- D a conjunction of further equations computed by the algorithm
- $r \in \{\top, \bot\}$

Initial state is $(N; \top; \top)$.

Formal Model

A square $f(x, y)$ where $x, y \in \{1, 2, 3, 4\}$ is called defined by $N \wedge D$ if there is an equation $f(x, y) \approx z$, $z \in \{1, 2, 3, 4\}$ in N or D. For otherwise $f(x, y)$ it is called undefined.

Rule-Based Algorithm

Deduce

$$(N; D; \top) \quad \Rightarrow \quad (N; D \wedge f(x,y) \approx 1; \top)$$

provided $f(x,y)$ is undefined in $N \wedge D$, for any $x, y \in \{1,2,3,4\}$.

Conflict

$$(N; D; \top) \quad \Rightarrow \quad (N; D; \bot)$$

provided for $y \neq z$ (i) $f(x,y) = f(x,z)$ for $f(x,y)$, $f(x,z)$ defined in $N \wedge D$ for some x, y, z or (ii) $f(y,x) = f(z,x)$ for $f(y,x)$, $f(z,x)$ defined in $N \wedge D$ for some x, y, z or (iii) $f(x,y) = f(x',y')$ for $f(x,y)$, $f(x',y')$ defined in $N \wedge D$ and $[x, x' \in \{1,2\}$ or $x, x' \in \{3,4\}]$ and $[y, y' \in \{1,2\}$ or $y, y' \in \{3,4\}]$ and $x \neq x'$ or $y \neq y'$.

Rule-Based Algorithm

Backtrack

$$(N; D' \wedge f(x,y) \approx z \wedge D''; \bot) \quad \Rightarrow \quad (N; D' \wedge f(x,y) \approx z + 1; \top)$$

provided $z < 4$ and $D'' = \top$ or D'' contains only equations of the form $f(x', y') \approx 4$.

Fail

$$(N; D; \bot) \quad \Rightarrow \quad (N; \top; \bot)$$

provided $D \neq \top$ and D contains only equations of the form $f(x,y) \approx 4$.

Rule-Based Algorithm

Properties: Rules are applied don't care non-deterministically.

An algorithm (set of rules) is sound if whenever it declares having found a solution it actually has computed a solution.

It is complete if it finds a solution if one exists.

It is terminating if it never runs forever.

Rule-Based Algorithm

Proposition 0.1 (Soundness):
The rules Deduce, Conflict, Backtrack and Fail are sound.
Starting from an initial state $(N; \top; \top)$:
(i) for any final state $(N; D; \top)$, the equations in $N \wedge D$ are a solution, and,
(ii) for any final state $(N; \top; \bot)$ there is no solution to the initial problem.

Rule-Based Algorithm

Proposition 0.2 (Completeness):
The rules Deduce, Conflict, Backtrack and Fail are complete. For any solution $N \wedge D$ of the Sudoku there is a sequence of rule applications such that $(N; D; \top)$ is a final state.

Rule-Based Algorithm

Proposition 0.3 (Termination):
The rules Deduce, Conflict, Backtrack and Fail terminate on any input state $(N; \top; \top)$.

Confluence

Another important property for don't care non-deterministic rule based definitions of algorithms is confluence.

It means that whenever several sequences of rules are applicable to a given states, the respective results can be rejoined by further rule applications to a common problem state.

Confluence

Proposition 0.4 (Deduce and Conflict are Locally Confluent): Given a state $(N; D; \top)$ out of which two different states $(N; D_1; \top)$ and $(N; D_2; \bot)$ can be generated by Deduce and Conflict in one step, respectively, then the two states can be rejoined to a state $(N; D'; *)$ via further rule applications.

Result

It works.

But: It looks like a lot of effort for a problem that one can solve with a little bit of thinking.

Reason: Our approach is very general, it can actually be used to "potentially solve" *any* problem in computer science.

Result

This difference is also important for automated reasoning:

- For problems that are well-known and frequently used, we can develop optimal specialized methods.
  $\Rightarrow$ Algorithms & Data-structures

- For new/unknown/changing problems, we have to develop generic methods that do "something useful".
  $\Rightarrow$ this lecture: Logic + Calculus + Implementation

- Combining the two approaches
  $\Rightarrow$ Automated Reasoning II (next semester): Logic modulo Theory + Calculus + Implementation

Topics of the Course

Preliminaries

math repetition

computer science repetition

orderings

induction (repetition)

rewrite systems

Propositional logic

logic: syntax, semantics

calculi: superposition, CDCL

implementation: 2-watched literal, clause learning

Topics of the Course

First-order predicate logic

- logic: syntax, semantics, model theory
- calculus: superposition
- implementation: sharing, indexing

First-order predicate logic with equality

- equational logic: unit equations
- calculus: term rewriting systems, Knuth-Bendix completion
- implementation: dependency pairs
- first-order logic with equality
- calculus: superposition
- implementation: rewriting

Literature

Is a big problem, actually you are the “guinea-pigs” for a new textbook.

Franz Baader and Tobias Nipkow: *Term rewriting and all that*, Cambridge Univ. Press, 1998. (Textbook on equational reasoning)

Armin Biere and Marijn Heule and Hans van Maaren and Toby Walsh (editors): *Handbook of Satisfiability*, IOS Press, 2009. (Be careful: Handbook, hard to read)

Alan Robinson and Andrei Voronkov (editors): *Handbook of Automated Reasoning*, Vol I & II, Elsevier, 2001. (Be careful: Handbook, very hard to read)

Part 1: Preliminaries

- math repetition
- computer science repetition
- orderings
- induction (repetition)
- rewrite systems

1.1 Mathematical Prerequisites

$\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers

$\mathbb{N}^+$ is the set of positive natural numbers without 0

$\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ denote the integers, rational numbers and the real numbers, respectively.

Multisets

Given a set M, a multi-set S over M is a mapping $S\colon M \to \mathbb{N}$, where S specifies the number of occurrences of elements m of the base set M within the multiset S.

We use the standard set notations $\in$, $\subset$, $\subseteq$, $\cup$, $\cap$ with the analogous meaning for multisets, e.g., $(S_1 \cup S_2)(m) = S_1(m) + S_2(m)$.

We also write multi-sets in a set like notation, e.g., the multi-set $S = \{1, 2, 2, 4\}$ denotes a multi-set over the set $\{1, 2, 3, 4\}$ where $S(1) = 1$, $S(2) = 2$, $S(3) = 0$, and $S(4) = 1$.

A multi-set S over a set M is finite if $\{m \in M \mid S(m) > 0\}$ is finite. In this lecture we only consider finite multi-sets.

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
$R \cup Q := \{(m_1, \dots, m_n) \in M \mid (m_1, \dots, m_n) \in R$ or $(m_1, \dots, m_n) \in Q\}$
$R \cap Q := \{(m_1, \dots, m_n) \in M \mid (m_1, \dots, m_n) \in R$ and $(m_1, \dots, m_n) \in Q\}$.

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

Relations

The characteristic function of a relation R or sometimes called predicate indicates membership. In addition of writing $(m_1, \ldots, m_n) \in R$ we also write $R(m_1, \ldots, m_n)$. So the predicate $R(m_1, \ldots, m_n)$ holds or is true if in fact $(m_1, \ldots, m_n)$ belongs to the relation R.

Words

Given a nonempty alphabet Σ the set Σ^* of finite words over Σ is defined by

(i) the empty word $\epsilon \in \Sigma^*$

(ii) for each letter $a \in \Sigma$ also $a \in \Sigma^*$

(iii) if $u, v \in \Sigma^*$ so $uv \in \Sigma^*$ where uv denotes the concatenation of u and v.

Words

The length $|u|$ of a word $u \in \Sigma^*$ is defined by

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

(ii) $|a| := 1$ for any $a \in \Sigma$ and

(iii) $|uv| := |u| + |v|$ for any $u, v \in \Sigma^*$.

1.2 Computer Science Prerequisites

A little bit on computational complexity theory.

Big O

Let $f(n)$ and $g(n)$ be functions from the naturals into the non-negative reals. Then

$$O(f(n)) = \{g(n) \mid \exists\, c > 0\, \exists\, n_0 \in \mathbb{N}^+\, \forall\, n \geq n_0\; g(n) \leq c \cdot f(n)\}$$

We use $\forall$, reads “for all”, and $\exists$, reads “exists”, on the object and meta level.

Decision Problem

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

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

NP

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

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

(ii) the predicate Q is in **P**.

Reducible,NP-Hard, NP-Complete

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

For example, if L is reducible to L' and $L' \in \mathbf{P}$ then $L \in \mathbf{P}$.

A decision problem is **NP**-hard if every problem in **NP** is polynomial-time reducible to it.

A decision problem is **NP**-complete if it is **NP**-hard and in **NP**.

1.3 Ordering

Termination of rewrite systems and proof theory is strongly related to the concept of (well-founded) orderings.

An ordering R is a binary relation on some set M.

Ordering

Relevant properties of orderings are: Depending on particular properties such as

$$\begin{array}{rl}
\text{(reflexivity)} & \forall x \in M\, R(x,x) \\
\text{(irreflexivity)} & \forall x \in M\, \neg R(x,x) \\
\text{(antisymmetry)} & \forall x,y \in M\, (R(x,y) \wedge R(y,x) \rightarrow x = y) \\
\text{(transitivity)} & \forall x,y,z \in M\, (R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \\
\text{(totality)} & \forall x,y \in M\, (R(x,y) \vee R(y,x))
\end{array}$$

where $=$ is the identity relation on M. The boolean connectives $\wedge$, $\vee$, and $\rightarrow$ read "and", "or", and "implies", respectively.

Partial Ordering

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

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

An $a \in M$ is called smallest, if $b \succ a$ for all $b \in M$ different from a.

Notation:
$\prec$ for the inverse relation $\succ^{-1}$
$\succeq$ for the reflexive closure $(\succ \cup =)$ of $\succ$

Well-Foundedness

A strict partial ordering $\succ$ on M is called well-founded (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

Let $\rightarrow$, $>$ be binary relations on the same set.

Lemma 1.1:
If $>$ is a well-founded partial ordering and $\rightarrow \subseteq >$, then $\rightarrow$ is terminating.

Lemma 1.2:
If $\rightarrow$ is a terminating binary relation over A, then $\rightarrow^{+}$ is a well-founded partial ordering.

Well-Founded Orderings: Examples

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

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

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

on $M_1 \times M_2$ be defined as

$$(a_1, a_2) \succ (b_1, b_2) \quad :\Leftrightarrow$$

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

(analogously for more than two orderings)

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

Well-Founded Orderings: Examples

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

$$w \succ w' \quad :\Leftrightarrow$$

$$|w| > |w'| \text{ or } (|w| = |w'| \text{ and } w >_{\Sigma,lex} w')$$

is a well-founded ordering on Σ^* (Exercise).

Counterexamples:

$(\mathbb{Z}, >)$

$(\mathbb{N}, <)$

the lexicographic ordering on Σ^*

Basic Properties of Well-Founded Orderings

Lemma 1.3:
$(M, \succ)$ is well-founded if and only if every $\emptyset \subset M' \subseteq M$ has a minimal element.

Lemma 1.4:
$(M_1, \succ_1)$ 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.

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 1.5:
If φ 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.

Multiset Orderings

Lemma 1.6 (König's Lemma):
Every finitely branching tree with infinitely many nodes contains an infinite path.

Multiset Orderings

Let $(M, \succ)$ be a strict partial ordering. The multiset extension of $\succ$ to multisets over M is defined by

$$
\begin{aligned}
& S_1 \succ_{\mathrm{mul}} S_2 \;\Leftrightarrow \\
& \quad S_1 \neq S_2 \text{ and} \\
& \quad\quad \forall m \in M\colon \big(S_2(m) > S_1(m) \\
& \quad\quad\quad\quad \Rightarrow\; \exists m' \in M\colon m' \succ m \text{ and } S_1(m') > S_2(m')\big)
\end{aligned}
$$

1.4 Induction

More or less all sets of objects in computer science or logic are defined *inductively*. Typically, this is done in a bottom-up way, where starting with some definite set, it is closed under a given set of operations.

Induction

Example 1.7 (Inductive Sets):

1. The set of all Sudoku problem states, consists of the set of start states $(N; \top; \top)$ for consistent assignments N plus all states that can be derived from the start states by the rules Deduce, Conflict, Backtrack, and Fail. This is a finite set.

2. The set $\mathbb{N}$ of the natural numbers, consists of 0 plus all numbers that can be computed from 0 by adding 1. This is an infinite set.

3. The set of all strings Σ^* over a finite alphabet Σ where all letters of Σ are contained in Σ^* and if u and v are words out of Σ^* so is the word uv. This is an infinite set.

Induction

All the previous examples have in common that there is an underlying well-founded ordering on the sets induced by the construction. The minimal elements for the Sudoku are the problem states $(N;\top;\top)$, for the natural numbers it is 0 and for the set of strings the empty word.

Now if we want to prove a property of an inductive set it is sufficient to prove it (i) for the minimal element(s) and (ii) assuming the property for an arbitrary set of elements, to prove that it holds for all elements that can be constructed "in one step" out those elements. This is the principle of *Noetherian Induction*.

Induction

Theorem 1.8 (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

> if $Q(m')$ for all $m' \in M$ such that $m \succ m'$,[a]
>
> then $Q(m)$.[b]

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

[a] induction hypothesis

[b] induction step

Induction

Theorem 1.9 (Properties Multi-Set Ordering):

(a) $\succ_{\mathrm{mul}}$ is a strict partial ordering.

(b) $\succ$ well-founded $\Rightarrow$ $\succ_{\mathrm{mul}}$ well-founded.

(c) $\succ$ total $\Rightarrow$ $\succ_{\mathrm{mul}}$ total.

1.5 Rewrite Systems

A rewrite system is a pair $(A, \rightarrow)$, where

A is a 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 $(a, b) \in \rightarrow$.

Rewrite Systems

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

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

Rewrite Systems

$\rightarrow^0$	$= \{\, (a, a) \mid a \in A \,\}$	identity
$\rightarrow^{i+1}$	$= \rightarrow^i \circ \rightarrow$	$i + 1$-fold composition
$\rightarrow^+$	$= \bigcup_{i>0} \rightarrow^i$	transitive closure
$\rightarrow^*$	$= \bigcup_{i\geq 0} \rightarrow^i \; = \; \rightarrow^+ \cup \rightarrow^0$	reflexive transitive closure
$\rightarrow^=$	$= \rightarrow \cup \rightarrow^0$	reflexive closure
$\rightarrow^{-1}$	$= \leftarrow \; = \{\, (b, c) \mid c \rightarrow b \,\}$	inverse
$\leftrightarrow$	$= \rightarrow \cup \leftarrow$	symmetric closure
$\leftrightarrow^+$	$= (\leftrightarrow)^+$	transitive symmetric closure
$\leftrightarrow^*$	$= (\leftrightarrow)^*$	refl. trans. symmetric closure

Rewrite Systems

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

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

c is a normal form of b, if $b \to^* c$ and c is in normal form.
Notation: $c = b{\downarrow}$ (if the normal form of b is unique).

Rewrite Systems

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.

Rewrite Systems

Lemma 1.10:

If $\rightarrow$ is terminating, then it is normalizing.

Note: The reverse implication does not hold.

Confluence

Let $(A, \to)$ be a rewrite system.

b and $c \in A$ are joinable, if there is an a such that $b \to^* a \; {}^*\!\!\leftarrow c$.
Notation: $b \downarrow c$.

The relation $\to$ is called

Church-Rosser, if $b \leftrightarrow^* c$ implies $b \downarrow c$.

confluent, if $b \; {}^*\!\!\leftarrow a \to^* c$ implies $b \downarrow c$.

locally confluent, if $b \leftarrow a \to c$ implies $b \downarrow c$.

convergent, if it is confluent and terminating.

Confluence

For a rewrite system $(M, \rightarrow)$ consider a sequence of elements a_i that are pairwise connected by the symmetric closure, i.e., $a_1 \leftrightarrow a_2 \leftrightarrow a_3 \ldots \leftrightarrow a_n$. We say that a_i is a peak in such a sequence, if actually $a_{i-1} \leftarrow a_i \rightarrow a_{i+1}$.

Confluence

Theorem 1.11:

The following properties are equivalent:

(i) $\rightarrow$ has the Church-Rosser property.

(ii) $\rightarrow$ is confluent.

Confluence

Lemma 1.12:
If $\rightarrow$ is confluent, then every element has at most one normal form.

Corollary 1.13:
If $\rightarrow$ is normalizing and confluent, then every element b has a unique normal form.

Proposition 1.14:
If $\rightarrow$ is normalizing and confluent, then $b \leftrightarrow^* c$ if and only if $b{\downarrow} = c{\downarrow}$.

Confluence and Local Confluence

Theorem 1.15 ("Newman's Lemma"):
If a terminating relation $\rightarrow$ is locally confluent, then it is confluent.

Part 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
- growing importance for discrete optimization problems (Automated Reasoning II)

2.1 Syntax

- propositional variables
- logical connectives
  $\Rightarrow$ Boolean connectives and constants

Propositional Variables

Let Σ be a set of propositional variables also called the signature of the (propositional) logic.

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

Propositional Formulas

PROP(Σ) is the set of propositional formulas over Σ inductively defined as follows:

$$\begin{array}{rcll}
\phi, \psi & ::= & \bot & \text{(falsum)} \\
& \mid & \top & \text{(verum)} \\
& \mid & P, \quad P \in \Sigma & \text{(atomic formula)} \\
& \mid & (\neg \phi) & \text{(negation)} \\
& \mid & (\phi \wedge \psi) & \text{(conjunction)} \\
& \mid & (\phi \vee \psi) & \text{(disjunction)} \\
& \mid & (\phi \rightarrow \psi) & \text{(implication)} \\
& \mid & (\phi \leftrightarrow \psi) & \text{(equivalence)}
\end{array}$$

Notational Conventions

As a notational convention we assume that $\neg$ binds strongest, and we remove outermost parenthesis, so $\neg P \lor Q$ is actually a shorthand for $((\neg P) \lor Q)$. For all other logical connectives we will explicitly put parenthesis when needed. From the semantics we will see that $\land$ and $\lor$ are associative and commutative. Therefore instead of $((P \land Q) \land R)$ we simply write $P \land Q \land R$.

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

Formula Manipulation

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

$$
\begin{aligned}
\mathsf{pos}(\phi) &:= \{\epsilon\} \text{ if } \phi \in \{\top, \bot\} \text{ or } \phi \in \Sigma \\
\mathsf{pos}(\neg\phi) &:= \{\epsilon\} \cup \{1p \mid p \in \mathsf{pos}(\phi)\} \\
\mathsf{pos}(\phi \circ \psi) &:= \{\epsilon\} \cup \{1p \mid p \in \mathsf{pos}(\phi)\} \cup \{2p \mid p \in \mathsf{pos}(\psi)\}
\end{aligned}
$$

where $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

Formula Manipulation

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 $\leq$, i.e., $p < q$ if $p \leq q$ but not $q \leq p$. By $\parallel$ we denote incomparable positions, i.e., $p \parallel q$ if neither $p \leq q$, nor $q \leq p$. Then 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 $p \parallel q$.

Formula Manipulation

The size of a formula ϕ is given by the cardinality of $\mathsf{pos}(\phi)$:
$|\phi| := |\,\mathsf{pos}(\phi)|$.

The subformula of ϕ at position $p \in \mathsf{pos}(\phi)$ is recursively defined by

$$\begin{array}{rcl}
\phi|_{\epsilon} & := & \phi \\
\neg\phi|_{1p} & := & \phi|_{p} \\
(\phi_1 \circ \phi_2)|_{ip} & := & \phi_i|_{p} \text{ where } i \in \{1, 2\}
\end{array}$$

$\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

Formula Manipulation

Finally, the replacement of a subformula at position $p \in \mathsf{pos}(\phi)$ by a formula ψ is recursively defined by

$$\begin{array}{rcl}
\phi[\psi]_\epsilon & := & \psi \\
(\neg\phi)[\psi]_{1p} & := & \neg(\phi[\psi]_p) \\
(\phi_1 \circ \phi_2)[\psi]_{1p} & := & (\phi_1[\psi]_p \circ \phi_2) \\
(\phi_1 \circ \phi_2)[\psi]_{2p} & := & (\phi_1 \circ \phi_2[\psi]_p)
\end{array}$$

where $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

Formula Manipulation

Example 2.1:
The set of positions for the formula $\phi = (A \land B) \rightarrow (A \lor B)$ is $\mathrm{pos}(\phi) = \{\epsilon, 1, 11, 12, 2, 21, 22\}$. The subformula at position 22 is B, $\phi|_{22} = B$ and replacing this formula by $A \leftrightarrow B$ results in $\phi[A \leftrightarrow B]_{22} = (A \land B) \rightarrow (A \lor (A \leftrightarrow B))$.

Formula Manipulation

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

Formula Manipulation

The polarity of a subformula ψ of ϕ at position p, $i \in \{1,2\}$ is recursively defined by

$$\begin{aligned}
\mathrm{pol}(\phi, \epsilon) &:= 1 \\
\mathrm{pol}(\neg\phi, 1p) &:= -\,\mathrm{pol}(\phi, p) \\
\mathrm{pol}(\phi_1 \circ \phi_2, ip) &:= \mathrm{pol}(\phi_i, p) \text{ if } \circ \in \{\wedge, \vee\} \\
\mathrm{pol}(\phi_1 \rightarrow \phi_2, 1p) &:= -\,\mathrm{pol}(\phi_1, p) \\
\mathrm{pol}(\phi_1 \rightarrow \phi_2, 2p) &:= \mathrm{pol}(\phi_2, p) \\
\mathrm{pol}(\phi_1 \leftrightarrow \phi_2, ip) &:= 0
\end{aligned}$$

Formula Manipulation

Example 2.2:

We reuse the formula $\phi = (A \land B) \rightarrow (A \lor B)$ Then $\text{pol}(\phi, 1) = \text{pol}(\phi, 11) = -1$ and $\text{pol}(\phi, 2) = \text{pol}(\phi, 22) = 1$. For the formula $\phi' = (A \land B) \leftrightarrow (A \lor B)$ we get $\text{pol}(\phi', \epsilon) = 1$ and $\text{pol}(\phi', p) = 0$ for all other $p \in \text{pos}(\phi')$, $p \neq \epsilon$.

2.2 Semantics

In classical logic (dating back to Aristoteles) 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 Σ-valuation is a map

$$\mathcal{A} : \Sigma \to \{0, 1\}.$$

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

Truth Value of a Formula in $\mathcal{A}$

Given a Σ-valuation $\mathcal{A}$, the function can be extended to $\mathcal{A} : \mathrm{PROP}(\Sigma) \to \{0,1\}$ by:

$$\begin{aligned}
\mathcal{A}(\bot) &= 0 \\
\mathcal{A}(\top) &= 1 \\
\mathcal{A}(\neg\phi) &= 1 - \mathcal{A}(\phi) \\
\mathcal{A}(\phi \wedge \psi) &= \mathrm{min}(\{\mathcal{A}(\phi), \mathcal{A}(\psi)\}) \\
\mathcal{A}(\phi \vee \psi) &= \mathrm{max}(\{\mathcal{A}(\phi), \mathcal{A}(\psi)\}) \\
\mathcal{A}(\phi \to \psi) &= \mathrm{max}(\{(1 - \mathcal{A}(\phi)), \mathcal{A}(\psi)\}) \\
\mathcal{A}(\phi \leftrightarrow \psi) &= \text{if } \mathcal{A}(\phi) = \mathcal{A}(\psi) \text{ then } 1 \text{ else } 0
\end{aligned}$$

2.3 Models, Validity, and Satisfiability

ϕ is valid in $\mathcal{A}$ ($\mathcal{A}$ is a model of ϕ; ϕ holds under $\mathcal{A}$):

$$\mathcal{A} \models \phi \;\;:\Leftrightarrow\;\; \mathcal{A}(\phi) = 1$$

ϕ is valid (or is a tautology):

$$\models \phi \;\;:\Leftrightarrow\;\; \mathcal{A} \models \phi \text{ for all } \Sigma\text{-valuations } \mathcal{A}$$

ϕ is called satisfiable if there exists an $\mathcal{A}$ such that $\mathcal{A} \models \phi$. Otherwise ϕ is called unsatisfiable (or contradictory).

Entailment and Equivalence

ϕ entails (implies) ψ (or ψ is a consequence of ϕ), written $\phi \models \psi$, if for all Σ-valuations $\mathcal{A}$ we have $\mathcal{A} \models \phi \;\Rightarrow\; \mathcal{A} \models \psi$.

ϕ and ψ are called equivalent, written $\phi \;\models\!\!\!\!\dashv\; \psi$, if for all Σ-valuations $\mathcal{A}$ we have $\mathcal{A} \models \phi \;\Leftrightarrow\; \mathcal{A} \models \psi$.

Proposition 2.3:
$\phi \models \psi$ if and only if $\models (\phi \rightarrow \psi)$.

Proposition 2.4:
$\phi \;\models\!\!\!\!\dashv\; \psi$ if and only if $\models (\phi \leftrightarrow \psi)$.

Entailment and Equivalence

Entailment is extended to sets of formulas N in the "natural way":

$N \models \phi$ if for all Σ-valuations $\mathcal{A}$:
  if $\mathcal{A} \models \psi$ for all $\psi \in N$, then $\mathcal{A} \models \phi$.

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:
ϕ is valid if and only if $\neg\phi$ is unsatisfiable.

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

Validity vs. Unsatisfiability

In a similar way, entailment $N \models \phi$ can be reduced to unsatisfiability:

Proposition 2.6:
$N \models \phi$ if and only if $N \cup \{\neg\phi\}$ is unsatisfiable.

Checking Unsatisfiability

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

If ϕ contains n distinct propositional variables, then it is sufficient to check 2^n valuations to see whether ϕ 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)

Truth Table

Let ϕ be a propositional formula over variables $P_1, \ldots, P_n$ and $k = |\,\mathrm{pos}(\phi)|$. Then a complete truth table for ϕ is a table with $n + k$ columns and $2^n + 1$ rows of the form

P_1	$\ldots$	P_n	$\phi\vert_{p_1}$	$\ldots$	$\phi\vert_{p_k}$
0	$\ldots$	0	$\mathcal{A}_1(\phi\vert_{p_1})$	$\ldots$	$\mathcal{A}_1(\phi\vert_{p_k})$
$\vdots$					
1	$\ldots$	1	$\mathcal{A}_{2^n}(\phi\vert_{p_1})$	$\ldots$	$\mathcal{A}_{2^n}(\phi\vert_{p_k})$

such that the $\mathcal{A}_i$ are exactly the 2^n different valuations for $P_1, \ldots, P_n$ and either $p_i \parallel p_{i+j}$ or $p_i \geq p_{i+j}$, in particular $p_k = \epsilon$ and $\phi|_{p_k} = \phi$ for all $i, j \geq 0$, $i + j \leq k$.

Truth Table

Truth tables can be used to check validity, satisfiability or unsatisfiability of a formula in a systematic way.

They have the nice property that if the rows are filled from left to right, then in order to compute $\mathcal{A}_i(\phi|_{p_j})$ the values for $\mathcal{A}_i$ of $\phi|_{p_j h}$ are already computed, $h \in \{1, 2\}$.

Substitution Theorem

Proposition 2.7:
Let ϕ_1 and ϕ_2 be equivalent formulas, and $\psi[\phi_1]_p$ be a formula in which ϕ_1 occurs as a subformula at position p.

Then $\psi[\phi_1]_p$ is equivalent to $\psi[\phi_2]_p$.

Equivalences

Proposition 2.8:

The following equivalences are valid for all formulas ϕ, ψ, χ:

$(\phi \wedge \phi) \leftrightarrow \phi$	Idempotency $\wedge$
$(\phi \vee \phi) \leftrightarrow \phi$	Idempotency $\vee$
$(\phi \wedge \psi) \leftrightarrow (\psi \wedge \phi)$	Commutativity $\wedge$
$(\phi \vee \psi) \leftrightarrow (\psi \vee \phi)$	Commutativity $\vee$
$(\phi \wedge (\psi \wedge \chi)) \leftrightarrow ((\phi \wedge \psi) \wedge \chi)$	Associativity $\wedge$
$(\phi \vee (\psi \vee \chi)) \leftrightarrow ((\phi \vee \psi) \vee \chi)$	Associativity $\vee$
$(\phi \wedge (\psi \vee \chi)) \leftrightarrow (\phi \wedge \psi) \vee (\phi \wedge \chi)$	Distributivity $\wedge\vee$
$(\phi \vee (\psi \wedge \chi)) \leftrightarrow (\phi \vee \psi) \wedge (\phi \vee \chi)$	Distributivity $\vee\wedge$

Equivalences

$(\phi \wedge (\phi \vee \psi)) \leftrightarrow \phi$	Absorption $\wedge\vee$
$(\phi \vee (\phi \wedge \psi)) \leftrightarrow \phi$	Absorption $\vee\wedge$
$(\phi \wedge \neg\phi) \leftrightarrow \bot$	Introduction $\bot$
$(\phi \vee \neg\phi) \leftrightarrow \top$	Introduction $\top$

Equivalences

$\neg(\phi \vee \psi) \leftrightarrow (\neg\phi \wedge \neg\psi)$	De Morgan $\neg\vee$
$\neg(\phi \wedge \psi) \leftrightarrow (\neg\phi \vee \neg\psi)$	De Morgan $\neg\wedge$
$\neg\top \leftrightarrow \bot$	Propagate $\neg\top$
$\neg\bot \leftrightarrow \top$	Propagate $\neg\bot$

Equivalences

$(\phi \wedge \top) \leftrightarrow \phi$	Absorption $\top\wedge$
$(\phi \vee \bot) \leftrightarrow \phi$	Absorption $\bot\vee$
$(\phi \rightarrow \bot) \leftrightarrow \neg\phi$	Eliminate $\bot \rightarrow$
$(\phi \leftrightarrow \bot) \leftrightarrow \neg\phi$	Eliminate $\bot \leftrightarrow$
$(\phi \leftrightarrow \top) \leftrightarrow \phi$	Eliminate $\top \leftrightarrow$
$(\phi \vee \top) \leftrightarrow \top$	Propagate $\top$
$(\phi \wedge \bot) \leftrightarrow \bot$	Propagate $\bot$

Equivalences

$(\phi \to \psi) \leftrightarrow (\neg\phi \lor \psi)$	Eliminate $\to$
$(\phi \leftrightarrow \psi) \leftrightarrow (\phi \to \psi) \land (\psi \to \phi)$	Eliminate1 $\leftrightarrow$
$(\phi \leftrightarrow \psi) \leftrightarrow (\phi \land \psi) \lor (\neg\phi \land \neg\psi)$	Eliminate2 $\leftrightarrow$

For simplification purposes the equivalences are typically applied as left to right rules.

2.4 Normal Forms

We define conjunctions of formulas as follows:

$$\bigwedge_{i=1}^{0} \phi_i = \top.$$

$$\bigwedge_{i=1}^{1} \phi_i = \phi_1.$$

$$\bigwedge_{i=1}^{n+1} \phi_i = \bigwedge_{i=1}^{n} \phi_i \wedge \phi_{n+1}.$$

and analogously disjunctions:

$$\bigvee_{i=1}^{0} \phi_i = \bot.$$

$$\bigvee_{i=1}^{1} \phi_i = \phi_1.$$

$$\bigvee_{i=1}^{n+1} \phi_i = \bigvee_{i=1}^{n} \phi_i \vee \phi_{n+1}.$$

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?

CNF and DNF

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 2.9:
For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).

Proof:
We consider the case of CNF and propose a naive algorithm.

Apply the following rules as long as possible (modulo associativity and commutativity of $\wedge$ and $\vee$):

Step 1: Eliminate equivalences:

$$\phi[(\psi_1 \leftrightarrow \psi_2)]_p \;\Rightarrow_{\mathrm{ECNF}}\; \phi[(\psi_1 \rightarrow \psi_2) \wedge (\psi_1 \rightarrow \psi_2)]_p$$

Conversion to CNF/DNF

Step 2: Eliminate implications:

$$\phi[(\psi_1 \rightarrow \psi_2)]_p \;\Rightarrow_{\mathsf{ECNF}}\; \phi[(\neg\psi_1 \vee \psi_2)]_p$$

Step 3: Push negations downward:

$$\phi[\neg(\psi_1 \vee \psi_2)]_p \;\Rightarrow_{\mathsf{ECNF}}\; \phi[(\neg\psi_1 \wedge \neg\psi_2)]_p$$
$$\phi[\neg(\psi_1 \wedge \psi_2)]_p \;\Rightarrow_{\mathsf{ECNF}}\; \phi[(\neg\psi_1 \vee \neg\psi_2)]_p$$

Step 4: Eliminate multiple negations:

$$\phi[\neg\neg\psi]_p \;\Rightarrow_{\mathsf{ECNF}}\; \phi[\psi]_p$$

Conversion to CNF/DNF

Step 5: Push disjunctions downward:

$$\phi[(\psi_1 \wedge \psi_2) \vee \chi]_p \ \Rightarrow_{\mathsf{ECNF}} \ \phi[(\psi_1 \vee \chi) \wedge (\psi_2 \vee \chi)]_p$$

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

$$\begin{aligned}
\phi[(\psi \wedge \top)]_p &\Rightarrow_{\mathsf{ECNF}} \phi[\psi]_p \\
\phi[(\psi \wedge \bot)]_p &\Rightarrow_{\mathsf{ECNF}} \phi[\bot]_p \\
\phi[(\psi \vee \top)]_p &\Rightarrow_{\mathsf{ECNF}} \phi[\top]_p \\
\phi[(\psi \vee \bot)]_p &\Rightarrow_{\mathsf{ECNF}} \phi[\psi]_p \\
\phi[\neg\bot]_p &\Rightarrow_{\mathsf{ECNF}} \phi[\top]_p \\
\phi[\neg\top]_p &\Rightarrow_{\mathsf{ECNF}} \phi[\bot]_p
\end{aligned}$$

Conversion to CNF/DNF

Proving termination is easy for steps 2, 4, and 6; steps 1, 3, and 5 are a bit more complicated.

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.

□

Complexity

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

Negation Normal Form (NNF)

The formula after application of Step 4 is said to be in Negation Normal Form, i.e., it does not contain $\rightarrow, \leftrightarrow$ and negation symbols only occur in front of propositional variables (atoms).

Satisfiability-preserving Transformations

The goal

“find a formula ψ in CNF such that $\phi \models\mid \psi$”

is unpractical.

But if we relax the requirement to

“find a formula ψ in CNF such that $\phi \models \bot \Leftrightarrow \psi \models \bot$”

we can get an efficient transformation.

Satisfiability-preserving Transformations

Idea: A formula $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \wedge (P \leftrightarrow \phi)$ is satisfiable where P is a new propositional variable that does not occur in ψ and works as an abbreviation for ϕ.

We can use this rule recursively for all subformulas in the original formula (this introduces a linear number of new propositional variables).

Conversion of the resulting formula to CNF increases the size only by an additional factor (each formula $P \leftrightarrow \phi$ gives rise to at most one application of the distributivity law).

Optimized Transformations

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

For example if $\psi[\phi_1 \leftrightarrow \phi_2]_p$ and $\mathrm{pol}(\psi, p) = -1$ then for CNF transformation do $\psi[(\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2)]_p$.

Optimized Transformations

Proposition 2.10:

Let P be a propositional variable not occurring in $\psi[\phi]_p$.

If $\text{pol}(\psi, p) = 1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \land (P \rightarrow \phi)$ is satisfiable.

If $\text{pol}(\psi, p) = -1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \land (\phi \rightarrow P)$ is satisfiable.

If $\text{pol}(\psi, p) = 0$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \land (P \leftrightarrow \phi)$ is satisfiable.

Proof:

Exercise. □

Optimized Transformations

The number of eventually generated clauses is a good indicator for useful CNF transformations:

ψ	$\nu(\psi)$	$\bar{\nu}(\psi)$
$\phi_1 \wedge \phi_2$	$\nu(\phi_1) + \nu(\phi_2)$	$\bar{\nu}(\phi_1)\bar{\nu}(\phi_2)$
$\phi_1 \vee \phi_2$	$\nu(\phi_1)\nu(\phi_2)$	$\bar{\nu}(\phi_1) + \bar{\nu}(\phi_2)$
$\phi_1 \rightarrow \phi_2$	$\bar{\nu}(\phi_1)\nu(\phi_2)$	$\nu(\phi_1) + \bar{\nu}(\phi_2)$
$\phi_1 \leftrightarrow \phi_2$	$\nu(\phi_1)\bar{\nu}(\phi_2) + \bar{\nu}(\phi_1)\nu(\phi_2)$	$\nu(\phi_1)\nu(\phi_2) + \bar{\nu}(\phi_1)\bar{\nu}(\phi_2)$
$\neg\phi_1$	$\bar{\nu}(\phi_1)$	$\nu(\phi_1)$
$P, \top, \bot$	1	1

Optimized CNF

Step 1: Exhaustively apply modulo C of $\leftrightarrow$, AC of $\land$, $\lor$:

$$
\begin{aligned}
\phi[(\psi \land \top)]_{\mathsf{p}} &\Rightarrow_{\mathsf{OCNF}} \phi[\psi]_{\mathsf{p}} \\
\phi[(\psi \lor \bot)]_{\mathsf{p}} &\Rightarrow_{\mathsf{OCNF}} \phi[\psi]_{\mathsf{p}} \\
\phi[(\psi \leftrightarrow \bot)]_{\mathsf{p}} &\Rightarrow_{\mathsf{OCNF}} \phi[\neg\psi]_{\mathsf{p}} \\
\phi[(\psi \leftrightarrow \top)]_{\mathsf{p}} &\Rightarrow_{\mathsf{OCNF}} \phi[\psi]_{\mathsf{p}} \\
\phi[(\psi \lor \top)]_{\mathsf{p}} &\Rightarrow_{\mathsf{OCNF}} \phi[\top]_{\mathsf{p}} \\
\phi[(\psi \land \bot)]_{\mathsf{p}} &\Rightarrow_{\mathsf{OCNF}} \phi[\bot]_{\mathsf{p}}
\end{aligned}
$$

Optimized CNF

$$
\begin{aligned}
\phi[(\psi \wedge \psi)]_p &\Rightarrow_{\mathsf{OCNF}} \phi[\psi]_p \\
\phi[(\psi \vee \psi)]_p &\Rightarrow_{\mathsf{OCNF}} \phi[\psi]_p \\
\phi[(\psi_1 \wedge (\psi_1 \vee \psi_2))]_p &\Rightarrow_{\mathsf{OCNF}} \phi[\psi_1]_p \\
\phi[(\psi_1 \vee (\psi_1 \wedge \psi_2))]_p &\Rightarrow_{\mathsf{OCNF}} \phi[\psi_1]_p \\
\phi[(\psi \wedge \neg\psi)]_p &\Rightarrow_{\mathsf{OCNF}} \phi[\bot]_p \\
\phi[(\psi \vee \neg\psi)]_p &\Rightarrow_{\mathsf{OCNF}} \phi[\top]_p \\
\phi[\neg\top]_p &\Rightarrow_{\mathsf{OCNF}} \phi[\bot]_p \\
\phi[\neg\bot]_p &\Rightarrow_{\mathsf{OCNF}} \phi[\top]_p
\end{aligned}
$$

Optimized CNF

$$\phi[(\psi \to \bot)]_p \Rightarrow_{\mathsf{OCNF}} \phi[\neg\psi]_p$$

$$\phi[(\psi \to \top)]_p \Rightarrow_{\mathsf{OCNF}} \phi[\top]_p$$

$$\phi[(\bot \to \psi)]_p \Rightarrow_{\mathsf{OCNF}} \phi[\top]_p$$

$$\phi[(\top \to \psi)]_p \Rightarrow_{\mathsf{OCNF}} \phi[\psi]_p$$

Optimized CNF

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

$$\psi[\phi]_p \;\Rightarrow_{\mathrm{OCNF}}\; \psi[P]_p \land \mathrm{def}(\psi, p, P)$$

where P is new to $\psi[\phi]_p$, $\mathrm{def}(\psi, p, P)$ is defined polarity dependent according to Proposition 2.10 and $\nu(\psi[\phi]_p) > \nu(\psi[P]_p \land \mathrm{def}(\psi, p, P))$.

Remark: Although computing ν is not practical in general, the test $\nu(\psi[\phi]_p) > \nu(\psi[P]_p \land \mathrm{def}(\psi, p, P))$ can be computed in constant time.

Optimized CNF

Step 3: Eliminate equivalences polarity dependent:

$$\phi[\psi_1 \leftrightarrow \psi_2]_p \;\Rightarrow_{\mathsf{OCNF}}\; \phi[(\psi_1 \rightarrow \psi_2) \wedge (\psi_2 \rightarrow \psi_1)]_p$$

if $\mathrm{pol}(\phi, p) = 1$ or $\mathrm{pol}(\phi, p) = 0$

$$\phi[\psi_1 \leftrightarrow \psi_2]_p \;\Rightarrow_{\mathsf{OCNF}}\; \phi[(\psi_1 \wedge \psi_2) \vee (\neg\psi_2 \wedge \neg\psi_1)]_p$$

if $\mathrm{pol}(\phi, p) = -1$

Optimized CNF

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

Remark: The $\Rightarrow_{\mathrm{OCNF}}$ algorithm is already close to a state of the art algorithm. Missing are further redundancy tests and simplification mechanisms we will discuss later on in this section.

2.5 Superposition for PROP(Σ)

Superposition for PROP(Σ) is:

- resolution (Robinson 1965) +
- ordering restrictions (Bachmair & Ganzinger 1990) +
- abstract redundancy criterion (B&G 1990) +
- partial model construction (B & G 1990) +
- partial-model based inference restriction (Weidenbach)

Resolution for PROP(Σ)

A calculus is a set of inference and reduction rules for a given logic (here PROP(Σ)).

We only consider calculi operating on a set of clauses N. Inference rules *add* new clauses to N whereas reduction rules *remove* clauses from N or *replace* clauses by “simpler” ones.

We are only interested in unsatisfiability, i.e., the considered calculi test whether a clause set N is unsatisfiable. So, in order to check validity of a formula ϕ we check unsatisfiability of the clauses generated from $\neg\phi$.

Resolution for PROP(Σ)

For clauses we switch between the notation as a disjunction, e.g., $P \vee Q \vee P \vee \neg R$, and the notation as a multiset, e.g., $\{P, Q, P, \neg R\}$. This makes no difference as we consider $\vee$ in the context of clauses always modulo AC. Note that $\bot$, the empty disjunction, corresponds to $\emptyset$, the empty multiset.

For literals we write L, possibly with subscript.. If $L = P$ then $\overline{L} = \neg P$ and if $L = \neg P$ then $\overline{L} = P$, so the bar flips the negation of a literal.

Clauses are typically denoted by letters C, D, possibly with subscript.

Resolution for PROP(Σ)

The resolution calculus consists of the inference rules resolution and factoring:

Resolution

$$\mathcal{I}\ \frac{C_1 \vee P \qquad C_2 \vee \neg P}{C_1 \vee C_2}$$

Factoring

$$\mathcal{I}\ \frac{C \vee L \vee L}{C \vee L}$$

where C_1, C_2, C always stand for clauses, all inference/reduction rules are applied with respect to AC of $\vee$. Given a clause set N the schema above the inference bar is mapped to N and the resulting clauses below the bar are then *added* to N.

Resolution for PROP(Σ)

and the reduction rules subsumption and tautology deletion:

$$\text{Subsumption} \qquad\qquad \text{Tautology Deletion}$$

$$\mathcal{R}\frac{C_1 \quad C_2}{C_1} \qquad\qquad \mathcal{R}\frac{C \vee P \vee \neg P}{}$$

where for subsumption we assume $C_1 \subseteq C_2$. Given a clause set N the schema above the reduction bar is mapped to N and the resulting clauses below the bar *replace* the clauses above the bar in N.

Clauses that can be removed are called redundant.

Resolution for PROP(Σ)

So, if we consider clause sets N as states, $\uplus$ is disjoint union, we get the rules

Resolution

$$(N \uplus \{C_1 \lor P, C_2 \lor \neg P\}) \quad \Rightarrow \quad (N \cup \{C_1 \lor P, C_2 \lor \neg P\} \cup \{C_1 \lor C_2\})$$

Factoring

$$(N \uplus \{C \lor L \lor L\}) \quad \Rightarrow \quad (N \cup \{C \lor L \lor L\} \cup \{C \lor L\})$$

Resolution for PROP(Σ)

Subsumption

$$(N \uplus \{C_1, C_2\}) \quad \Rightarrow \quad (N \cup \{C_1\})$$

provided $C_1 \subseteq C_2$

Tautology Deletion

$$(N \uplus \{C \vee P \vee \neg P\}) \quad \Rightarrow \quad (N)$$

We need more structure than just (N) in order to define a useful rewrite system. We fix this later on.

Resolution for PROP(Σ)

Theorem 2.11:

The resolution calculus is sound and complete:

$$N \text{ is unsatisfiable iff } N \Rightarrow^* \{\bot\}$$

Proof:

Will be a consequence of soundness and completeness of superposition. □

Ordering restrictions

Let $\prec$ be a total ordering on Σ.

We lift $\prec$ to a total ordering on literals by $\prec \subseteq \prec_L$ and $P \prec_L \neg P$ and $\neg P \prec_L Q$ for all $P \prec Q$.

We further lift $\prec_L$ to a total ordering on clauses $\prec_C$ by considering the multiset extension of $\prec_L$ for clauses.

Eventually, we overload $\prec$ with $\prec_L$ and $\prec_C$.

We define $N^{\prec C} = \{D \in N \mid D \prec C\}$.

Ordering restrictions

Eventually we will restrict inferences to maximal literals with respect to $\prec$.

Abstract Redundancy

A clause C is redundant with respect to a clause set N if $N^{\prec C} \models C$.

Tautologies are redundant. Subsumed clauses are redundant if $\subseteq$ is strict.

Remark: Note that for finite N, $N^{\prec C} \models C$ can be decided for PROP(Σ) but is as hard as testing unsatisfiability for a clause set N.

Partial Model Construction

Given a clause set N and an ordering $\prec$ we can construct a (partial) model $N_{\mathcal{I}}$ for N as follows:

$$N_C := \bigcup_{D \prec C} \delta_D$$

$$\delta_D := \begin{cases} \{P\} & \text{if } D = D' \lor P, P \text{ strictly maximal and } N_D \not\models D \\ \emptyset & \text{otherwise} \end{cases}$$

$$N_{\mathcal{I}} := \bigcup_{C \in N} \delta_C$$

Partial Model Construction

Clauses C with $\delta_C \neq \emptyset$ are called productive. Some properties of the partial model construction.

Proposition 2.12:

1. For every D with $(C \vee \neg P) \prec D$ we have $\delta_D \neq \{P\}$.
2. If $\delta_C = \{P\}$ then $N_C \cup \delta_C \models C$.
3. If $N_C \models D$ then for all C' with $C \prec C'$ we have $N_{C'} \models D$ and in particular $N_{\mathcal{I}} \models D$.

Notation: N, $N^{\prec C}$, $N_{\mathcal{I}}$, N_C

Please properly distinguish:

- N is a set of clauses interpreted as the conjunction of all clauses.
- $N^{\prec C}$ is of set of clauses from N strictly smaller than C with respect to $\prec$.
- $N_{\mathcal{I}}$, N_C are sets of atoms, often called Herbrand Interpretations. $N_{\mathcal{I}}$ is the overall (partial) model for N, whereas N_C is generated from all clauses from N strictly smaller than C.
- Validity is defined by $N_{\mathcal{I}} \models P$ if $P \in N_{\mathcal{I}}$ and $N_{\mathcal{I}} \models \neg P$ if $P \notin N_{\mathcal{I}}$, accordingly for N_C.

Superposition

The superposition calculus consists of the inference rules superposition left and factoring:

Superposition Left

$(N \uplus \{C_1 \vee P, C_2 \vee \neg P\}) \quad \Rightarrow \quad (N \cup \{C_1 \vee P, C_2 \vee \neg P\} \cup \{C_1 \vee C_2\})$

where P is strictly maximal in $C_1 \vee P$ and $\neg P$ is maximal in $C_2 \vee \neg P$

Factoring

$(N \uplus \{C \vee P \vee P\}) \quad \Rightarrow \quad (N \cup \{C \vee P \vee P\} \cup \{C \vee P\})$

where P is maximal in $C \vee P \vee P$

Superposition

examples for specific redundancy rules are

Subsumption

$$(N \uplus \{C_1, C_2\}) \quad \Rightarrow \quad (N \cup \{C_1\})$$

provided $C_1 \subset C_2$

Tautology Deletion

$$(N \uplus \{C \vee P \vee \neg P\}) \quad \Rightarrow \quad (N)$$

Subsumption Resolution

$$(N \uplus \{C_1 \vee L, C_2 \vee \overline{L}\}) \quad \Rightarrow \quad (N \cup \{C_1 \vee L, C_2\})$$

where $C_1 \subseteq C_2$

Superposition

Theorem 2.13:

If from a clause set N all possible superposition inferences are redundant and $\bot \notin N$ then N is satisfiable and $N_{\mathcal{I}} \models N$.

Superposition

So the proof actually tells us that at any point in time we need only to consider either a superposition left inference between a minimal false clause and a productive clause or a factoring inference on a minimal false clause.

A Superposition Theorem Prover *STP*

3 clause sets:

N(ew) containing new inferred clauses

U(sable) containing reduced new inferred clauses

clauses get into *W(orked) O(ff)* once their inferences have been computed

Strategy:

Inferences will only be computed when there are no possibilities for simplification

Rewrite Rules for *STP*

Tautology Deletion

$$(N \uplus \{C\}; U; WO) \quad \Rightarrow_{STP} \quad (N; U; WO)$$

if C is a tautology

Forward Subsumption

$$(N \uplus \{C\}; U; WO) \quad \Rightarrow_{STP} \quad (N; U; WO)$$

if some $D \in (U \cup WO)$ subsumes C

Backward Subsumption U

$$(N \uplus \{C\}; U \uplus \{D\}; WO) \quad \Rightarrow_{STP} \quad (N \cup \{C\}; U; WO)$$

if C strictly subsumes D ($C \subset D$)

Rewrite Rules for *STP*

Backward Subsumption *WO*

$(N \uplus \{C\}; U; WO \uplus \{D\}) \quad \Rightarrow_{STP} \quad (N \cup \{C\}; U; WO)$

if C strictly subsumes D ($C \subset D$)

Forward Subsumption Resolution

$(N \uplus \{C_1 \vee L\}; U; WO) \quad \Rightarrow_{STP} \quad (N \cup \{C_1\}; U; WO)$

if there exists $C_2 \vee \overline{L} \in (U \cup WO)$ such that $C_2 \subseteq C_1$

Backward Subsumption Resolution *U*

$(N \uplus \{C_1 \vee L\}; U \uplus \{C_2 \vee \overline{L}\}; WO) \quad \Rightarrow_{STP} \quad (N \cup \{C_1 \vee L\}; U \uplus \{C_2\}; WO)$

if $C_1 \subseteq C_2$

Rewrite Rules for *STP*

Backward Subsumption Resolution *WO*

$(N \uplus \{C_1 \vee L\}; U; WO \uplus \{C_2 \vee \overline{L}\}) \quad \Rightarrow_{STP} \quad (N \cup \{C_1 \vee L\}; U; WO \uplus \{C_2\})$

if $C_1 \subseteq C_2$

Clause Processing

$(N \uplus \{C\}; U; WO) \quad \Rightarrow_{STP} \quad (N; U \cup \{C\}; WO)$

Inference Computation

$(\emptyset; U \uplus \{C\}; WO) \quad \Rightarrow_{STP} \quad (N; U; WO \cup \{C\})$

where N is the set of clauses derived by superposition inferences from C and clauses in WO.

Soundness and Completeness

Theorem 2.14:

$$N \models \bot \quad \Leftrightarrow \quad (N;\emptyset;\emptyset) \quad \Rightarrow^{*}_{STP} \quad (N' \cup \{\bot\};U;WO)$$

Proof in L. Bachmair, H. Ganzinger: Resolution Theorem Proving appeared in the Handbook of Automated Reasoning, 2001

Termination

Theorem 2.15:
For finite N and a strategy where the reduction rules Tautology Deletion, the two Subsumption and two Subsumption Resolution rules are always exhaustively applied before Clause Processing and Inference Computation, the rewrite relation $\Rightarrow_{STP}$ is terminating on $(N; \emptyset; \emptyset)$.

Proof: think of it (more later on).

Fairness

Problem:

If N is inconsistent, then $(N; \emptyset; \emptyset) \Rightarrow^*_{STP} (N' \cup \{\bot\}; U; WO)$.

Does this imply that *every* derivation starting from an inconsistent set N eventually produces $\bot$?

No: a clause could be kept in U without ever being used for an inference.

Fairness

We need in addition a fairness condition:

If an inference is possible forever (that is, none of its premises is ever deleted), then it must be computed eventually.

One possible way to guarantee fairness: Implement U as a queue (there are other techniques to guarantee fairness).

With this additional requirement, we get a stronger result: If N is inconsistent, then every *fair* derivation will eventually produce $\bot$.

2.6 The CDCL 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).

Assumption:
Clauses contain neither duplicated literals nor complementary literals.

CDCL: Conflict Driven Clause Learning

Satisfiability of Clause Sets

$\mathcal{A} \models N$ if and only if $\mathcal{A} \models C$ for all clauses C in N.

$\mathcal{A} \models C$ if and only if $\mathcal{A} \models L$ for some literal $L \in C$.

Partial Valuations

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

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. Note that truth of a literal with respect to M is defined differently than for $N_{\mathcal{I}}$.

Partial Valuations

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

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ 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 $\mathcal{A}$ be a partial valuation and P a variable that is undefined under $\mathcal{A}$. 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 $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and assigns 1 (0) to P.

P is called a pure literal.

The Davis-Putnam-Logemann-Loveland Proc.

```
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 clause $P$) return DPLL($M \cup \{P\}$, $N$);
    elsif ($N$ contains unit clause $\neg P$) 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($M \cup \{\neg P\}$, $N$)) return true;
        else return DPLL($M \cup \{P\}$, $N$);
    }
}
```

The Davis-Putnam-Logemann-Loveland Proc.

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

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 branching variable is not chosen randomly.

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

CDCL = DPLL + Information is reused by learning + Restart + Specific Data Structures

Branching Heuristics

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, prefer variables from recent conflicts.

The Deduction Algorithm

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.

The Deduction Algorithm

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.

Conflict Analysis and Learning

Goal: Reuse information that is obtained in one branch in further branches.

Method: Learning:

> If a conflicting clause is found, derive a new clause from the conflict and add it to the current set of clauses.

Problem: This may produce a large number of new clauses; therefore it may become necessary to delete some of them afterwards to save space.

Backjumping

Related technique:

non-chronological backtracking (“backjumping”):

> If a conflict is independent of some earlier branch, try to skip over that backtrack level.

Restart

Runtimes of DPLL-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 adopted variable selection heuristics, but learned clauses are kept.

In particular, after learning a unit clause a restart is done.

Formalizing DPLL with Refinements

The DPLL procedure is modeled by a transition relation $\Rightarrow_{\mathsf{DPLL}}$ on a set of states.

States:

- *fail*
- $(M; N)$

where M is a *list of annotated literals* and N is a set of clauses. We use $+$ to right add a literal or a list of literals to M

Annotated literal:

- L: deduced literal, due to unit propagation.
- L^{d}: decision literal (guessed literal).

Formalizing DPLL with Refinements

Unit Propagate:

$$(M; N \cup \{C \vee L\}) \;\Rightarrow_{\mathrm{DPLL}}\; (M + L; N \cup \{C \vee L\})$$

if C is false under M and L is undefined under M.

Decide:

$$(M; N) \;\Rightarrow_{\mathrm{DPLL}}\; (M + L^{\mathrm{d}}; N)$$

if L is undefined under M and contained in N.

Fail:

$$(M; N \cup \{C\}) \;\Rightarrow_{\mathrm{DPLL}}\; \mathit{fail}$$

if C is false under M and M contains no decision literals.

Formalizing DPLL with Refinements

Backjump:

$$(M' + L^{\mathrm{d}} + M''; N) \;\Rightarrow_{\mathrm{DPLL}}\; (M' + L'; N)$$

if there is some "backjump clause" $C \vee L'$ such that

$N \models C \vee L'$,

C is false under M', and

L' is undefined under M'.

Formalizing DPLL with Refinements

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: $\overline{L_1} \vee \ldots \vee \overline{L_n}$, where the L_i are all the decision literals in $M + L^{\mathrm{d}} + M'$. (But usually there are better choices.)

Formalizing DPLL with Refinements

Lemma 2.16:
If we reach a state $(M; N)$ starting from $(\text{nil}; N)$, 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.

Formalizing DPLL with Refinements

Lemma 2.17:

Every derivation starting from (nil; N) terminates.

Formalizing DPLL with Refinements

Lemma 2.18:
Suppose that we reach a state $(M; N)$ starting from $(\text{nil}; N)$ 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.

Formalizing DPLL with Refinements

Theorem 2.19:
(1) If we reach a final state $(M; N)$ starting from $(\text{nil}; N)$, then N is satisfiable and M is a model of N.

(2) If we reach a final state *fail* starting from $(\text{nil}; N)$, then N is unsatisfiable.

Getting Better Backjump Clauses

Suppose that we have reached a state $(M; N)$ such that some clause $C \in N$ (or following from 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 a backjump clause.

(2) Otherwise, $C = C' \vee \overline{L}$, such that L is a deduced literal.

For every deduced literal L, there is a clause $D \vee L$, such that $N \models D \vee L$ and D is false under M.

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

Getting Better Backjump Clauses

By repeating this process, we will eventually obtain a clause that consists only of complements of decision literals and can be used in the “Backjump” rule.

Moreover, such a clause is a good candidate for learning.

Learning Clauses

The DPLL system can be extended by two rules to learn and to forget clauses:

Learn:

$$(M; N) \Rightarrow_{\mathrm{DPLL}} (M; N \cup \{C\})$$

if $N \models C$.

Forget:

$$(M; N \uplus \{C\}) \Rightarrow_{\mathrm{DPLL}} (M; N)$$

if $N \models C$.

Learning Clauses

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

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

Restart

Part of the CDCL system the restart rule:

Restart:

$$(M; N) \;\Rightarrow_{\mathrm{DPLL}}\; (\mathrm{nil}; N)$$

The restart rule is typically applied after a certain number of clauses have been learned or a unit is derived. It is closely coupled with the variable order heuristic.

If Restart is only applied finitely often, termination is guaranteed.

Variable Order Heuristic

For every propositional variable P_i there is a positive score k_i. At start k_i may for example be 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.

Variable Order Heuristic

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

The purpose of these mechanisms is to keep the search focused. Parameter b directs the search around the conflict, parameter j decides how many learned clauses are “sufficient” to move in “speed ” of parameter l away from this conflict.

Preprocessing

Before DPLL search, and computation of the variable order heuristics, a number of preprocessing steps are performed:

(i) Subsumption

Non-strict version.

(ii) Purity Deletion

Delete all clauses containing a literal L where $\overline{L}$ does not occur in the clause set.

(iii) Subsumption Resolution

Preprocessing

(iv) Tautology Deletion

(v) Literal Elimination

do all possible resolution steps on a literal L and then throw away all clauses containing L or $\overline{L}$; repeat this as long as $|N|$ does not grow.

Further Information

The ideas described so far have been implemented in all modern SAT solvers: zChaff, miniSAT,picoSAT. Because of clause learning the algorithm is now called CDCL: Conflict Driven Clause Learning.

It has been shown in 2009 that CDCL can polynomially simulate resolution, a long standing open question:

Knot Pipatsrisawat, Adnan Darwiche: On the Power of Clause-Learning SAT Solvers with Restarts. CP 2009, 654-668

Literature

Lintao Zhang and Sharad Malik: The Quest for Efficient Boolean Satisfiability Solvers; Proc. CADE-18, LNAI 2392, pp. 295–312, Springer, 2002.

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli: Solving SAT and SAT Modulo Theories; From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T), pp. 937–977, Journal of the ACM, 53(6), 2006.

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/`.

2.8 Example: Sudoku

	1	2	3	4	5	6	7	8	9
1								1	
2	4								
3		2							
4					5		4		7
5			8				3		
6			1		9				
7	3			4			2		
8		5		1					
9				8		6			

Idea: $p^{d}_{i,j}$=true iff the value of square i, j is d

For example:

$p^{8}_{3,5} = \mathit{true}$

Coding Sudoku by Propositional Clauses

- Concrete values result in units: $p^d_{i,j}$
- For every square (i, j) we generate $p^1_{i,j} \vee \ldots \vee p^9_{i,j}$
- For every square (i, j) and pair of values $d < d'$ we generate $\neg p^d_{i,j} \vee \neg p^{d'}_{i,j}$
- For every value d and column i we generate $p^d_{i,1} \vee \ldots \vee p^d_{i,9}$
  (Analogously for rows and 3×3 boxes)
- For every value d, column i, and pair of rows $j < j'$ we generate $\neg p^d_{i,j} \vee \neg p^d_{i,j'}$
  (Analogously for rows and 3×3 boxes)

Constraint Propagation is Unit Propagation

	1	2	3	4	5	6	7	8	9
1								1	
2	4								
3		2							
4					5		4		7
5			8				3		
6			1		9				
7	3			4	7		2		
8		5		1					
9				8		6			

From $\neg p_{1,7}^{3} \vee \neg p_{5,7}^{3}$ and $p_{1,7}^{3}$ we obtain by unit propagating $\neg p_{5,7}^{3}$ and further from $p_{5,7}^{1} \vee p_{5,7}^{2} \vee p_{5,7}^{3} \vee p_{5,7}^{4} \vee \ldots \vee p_{5,7}^{9}$ we get $p_{5,7}^{1} \vee p_{5,7}^{2} \vee p_{5,7}^{4} \vee \ldots \vee p_{5,7}^{9}$ (and finally $p_{5,7}^{7}$).

2.9 Other Calculi

OBDDs (Ordered Binary Decision Diagrams):

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

$\Rightarrow$ canonical representation of formulas.

see script of the Computational Logic course,

see Chapter 6.1/6.2 of Michael Huth and Mark Ryan: *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge Univ. Press, 2000.

Other Calculi

FRAIGs (Fully Reduced And-Inverter Graphs)

Minimized graph representation of boolean circuits.

$\Rightarrow$ semi-canonical representation of formulas.

Implementation needs DPLL (and OBDDs) as subroutines.

Other Calculi

Tableau calculus

Hilbert calculus

Sequent calculus

Natural deduction

2.10 Superposition Versus CDCL

We establish a relationship between Superposition and CDCL. For CDCL we assume eager propagation and false clause detection.

Superposition: Is based on an ordering $\prec$. It computes a model assumption $N_{\mathcal{I}}$. Either $N_{\mathcal{I}}$ is a model, N contains the empty clause, or there is an inference on the minimal false clause with respect to $\prec$.

CDCL: Is based on a variable selection heuristic. It computes a model assumption via decision variables and propagation. Either this assumption is a model of N, N contains the empty clause, or there is a backjump clause that is learned.

Superposition Versus CDCL

Proposition 2.20:
Let $(L_1 + L_2 + \ldots + L_k; N)$ be a CDCL state. Some of the L_i may be decision literals and the corresponding propositional variables are $P_1, \ldots, P_k$. Furthermore, let us assume that $L_1 + \ldots + L_{k-1}$ is a partial valuation that does not falsify any clause in N whereas $L_1 + L_2 + \ldots + L_k$ falsifies some clause $C \vee \overline{L_k} \in N$. Then

(a) L_k is a propagated literal.

(b) The resolvent between $C \vee \overline{L_k}$ and the clause propagating L_k is a superposition inference and the conclusion is not redundant with respect to the ordering $P_1 \prec P_2 \ldots \prec P_k$.

Superposition Versus CDCL

Proposition 2.21:
The 1UIP backjump clause is not redundant.

Proof:
By Proposition 2.20 a one resolution step 1UIP backjump clause has this property. The argument in the proof of Proposition 2.20 can be repeated until we reach the first decision literal L_m by resolving away $L_k, L_{k-1}, \dots, L_{m+1}$. □

Superposition Versus CDCL

Proposition 2.22:
Let $(L_1 + L_2 + \ldots + L_k; N)$ be a CDCL state. We assume that all decision literals among the L_i are negative and let the corresponding propositional variables be $P_1, \ldots, P_k$. Furthermore, let us assume that $L_1 + \ldots + L_k$ is a partial valuation that does not falsify any clause in N. Then $N_{\mathcal{I}}^{\prec P_{k+1}} = \{P_1, \ldots, P_k\} \cap \{L_1, \ldots, L_k\}$ with ordering $P_1 \prec P_2 \ldots \prec P_{k+1}$.

Part 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)
  $\Rightarrow$ terms, atomic formulas
- logical connectives (domain-independent)
  $\Rightarrow$ Boolean combinations, quantifiers

Signature

A signature $\Sigma = (\Omega, \Pi)$ 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.

Signature

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).

Signature

Refined concept for practical applications:
many-sorted signatures (corresponds to simple type systems in programming languages); not so interesting 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.

Context-Free Grammars

We define many of our notions on the bases of context-free grammars. Recall that a context-free grammar $G = (N, T, P, S)$ consists of:

- a set of non-terminal symbols N
- a set of terminal symbols T
- a set P of rules $A ::= w$ where $A \in N$ and $w \in (N \cup T)^*$
- a start symbol S where $S \in N$

For rules $A ::= w_1$, $A ::= w_2$ we write $A ::= w_1 \mid w_2$

Terms

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

$$\begin{array}{llll} s, t, u, v ::= & x & , x \in X & \text{(variable)} \\ \mid & f(s_1, \ldots, s_n) & , f/n \in \Omega & \text{(functional term)} \end{array}$$

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

Terms

In other words, terms are formal expressions with well-balanced brackets 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 Σ are formed according to this syntax:

$$\begin{array}{rcll} A, B & ::= & P(s_1, \ldots, s_m) \quad , P/m \in \Pi & \text{(non-equational atom)} \\ & \Big[\;\mid & (s \approx t) & \text{(equation)} \;\Big] \end{array}$$

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, (cf. exercises). But deductive systems where equality is treated specifically are much more efficient.

Literals

$$\begin{array}{lcll} L & ::= & A & \text{(positive literal)} \\ & \mid & \neg A & \text{(negative literal)} \end{array}$$

Clauses

$$\begin{array}{lcll} C, D & ::= & \bot & \text{(empty clause)} \\ & \mid & L_1 \vee \ldots \vee L_k, \quad k \geq 1 & \text{(non-empty clause)} \end{array}$$

General First-Order Formulas

$\mathsf{F}_{\Sigma}(X)$ is the set of first-order formulas over Σ defined as follows:

$$
\begin{array}{rcll}
\phi, \psi, \chi & ::= & \bot & \text{(falsum)} \\
 & \mid & \top & \text{(verum)} \\
 & \mid & A & \text{(atomic formula)} \\
 & \mid & \neg\phi & \text{(negation)} \\
 & \mid & (\phi \wedge \psi) & \text{(conjunction)} \\
 & \mid & (\phi \vee \psi) & \text{(disjunction)} \\
 & \mid & (\phi \rightarrow \psi) & \text{(implication)} \\
 & \mid & (\phi \leftrightarrow \psi) & \text{(equivalence)} \\
 & \mid & \forall x\, \phi & \text{(universal quantification)} \\
 & \mid & \exists x\, \phi & \text{(existential quantification)}
\end{array}
$$

Notational Conventions

We omit brackets according to the conventions for propositional logic.

Furthermore, $\forall x_1, \ldots, x_n\, \phi$ $(\exists x_1, \ldots, x_n\, \phi)$ abbreviates $\forall x_1 \ldots \forall x_n\, \phi$ $(\exists x_1 \ldots \exists x_n\, \phi)$.

Notational Conventions

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

Examples:

$$
\begin{array}{ccc}
s + t * u & \text{for} & +(s, *(t, u)) \\
s * u \leq t + v & \text{for} & \leq(*(s, u), +(t, v)) \\
-s & \text{for} & -(s) \\
0 & \text{for} & 0()
\end{array}
$$

Example: Peano Arithmetic

$$
\begin{aligned}
\Sigma_{PA} &= (\Omega_{PA},\ \Pi_{PA}) \\
\Omega_{PA} &= \{0/0,\ +/2,\ */2,\ s/1\} \\
\Pi_{PA} &= \{\leq/2,\ </2\}
\end{aligned}
$$

$+, *, <, \leq$ infix; $* \ >_p \ + \ >_p \ < \ >_p \ \leq$

Examples of formulas over this signature are:

$$
\begin{aligned}
&\forall x, y\,(x \leq y \leftrightarrow \exists z(x + z \approx y)) \\
&\exists x \forall y\,(x + y \approx y) \\
&\forall x, y\,(x * s(y) \approx x * y + x) \\
&\forall x, y\,(s(x) \approx s(y) \rightarrow x \approx y) \\
&\forall x \exists y\,(x < y \land \neg \exists z(x < z \land z < y))
\end{aligned}
$$

Remarks About the Example

We observe that the symbols $\leq$, $<$, 0, s are redundant as they can be defined in first-order logic with equality just with the help of $+$. The first formula defines $\leq$, while the second defines zero. The last formula, respectively, defines s.

Eliminating the existential quantifiers by Skolemization (cf. below) reintroduces the "redundant" symbols.

Consequently there is a *trade-off* between the complexity of the quantification structure and the complexity of the signature.

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 ϕ):

$$\begin{aligned}
&\mathrm{pos}(x) = \{\varepsilon\},\\
&\mathrm{pos}(f(s_1,\dots,s_n)) = \{\varepsilon\} \cup \textstyle\bigcup_{i=1}^{n} \{\, i\,p \mid p \in \mathrm{pos}(s_i)\,\}.
\end{aligned}$$

$$\begin{aligned}
&\mathrm{pos}(P(t_1,\dots,t_n)) = \{\varepsilon\} \cup \textstyle\bigcup_{i=1}^{n} \{\, i\,p \mid p \in \mathrm{pos}(t_i)\,\},\\
&\mathrm{pos}(\forall x\,\phi) = \{\varepsilon\} \cup \{\, 1\,p \mid p \in \mathrm{pos}(\phi)\,\},\\
&\mathrm{pos}(\exists x\,\phi) = \{\varepsilon\} \cup \{\, 1\,p \mid p \in \mathrm{pos}(\phi)\,\}.
\end{aligned}$$

Positions in Terms and Formulas

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

Bound and Free Variables

In $Qx\,\phi$, $Q \in \{\exists, \forall\}$, we call ϕ 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.

Bound and Free Variables

Example:

$$\forall y \quad \overbrace{(\forall \overbrace{x \quad P(x)}^{\textit{scope}} \quad \to \quad Q(x, y))}^{\textit{scope}}$$

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.

In general, substitutions are mappings

$$\sigma : X \to \mathsf{T}_\Sigma(X)$$

such that the domain of σ, that is, the set

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

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

Substitutions

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 σ 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 σ to a term t or formula ϕ by structural induction over the syntactic structure of t or ϕ 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 σ 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.

Why this definition of substitution is well-defined will be discussed below.

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

$$\begin{aligned}
f(s_1, \ldots, s_n)\sigma &= f(s_1\sigma, \ldots, s_n\sigma) \\
\bot\sigma &= \bot \\
\top\sigma &= \top \\
P(s_1, \ldots, s_n)\sigma &= P(s_1\sigma, \ldots, s_n\sigma) \\
(u \approx v)\sigma &= (u\sigma \approx v\sigma) \\
\neg\phi\sigma &= \neg(\phi\sigma) \\
(\phi\rho\psi)\sigma &= (\phi\sigma\, \rho\, \psi\sigma) \ ; \quad \text{for each binary connective } \rho \\
(Qx\,\phi)\sigma &= Qz\,(\phi\,\sigma[x \mapsto z]) \ ; \quad \text{with } z \text{ a fresh variable}
\end{aligned}$$

Structural Induction

Proposition 3.1:
Let $G = (N, T, P, S)$ be a context-free grammar (possibly infinite) and let q be a property of T^* (the words over the alphabet T of terminal symbols of G).

q holds for *all* words $w \in L(G)$, whenever one can prove the following two properties:

Structural Induction

1. (*base cases*)
   $q(w')$ holds for each $w' \in T^*$ such that $X ::= w'$ is a rule in P.

2. (*step cases*)
   If $X ::= w_0 X_0 w_1 \ldots w_n X_n w_{n+1}$ is in P with $X_i \in N$, $w_i \in T^*$, $n \geq 0$, then for all $w_i' \in L(G, X_i)$, whenever $q(w_i')$ holds for $0 \leq i \leq n$, then also $q(w_0 w_0' w_1 \ldots w_n w_n' w_{n+1})$ holds.

Here $L(G, X_i) \subseteq T^*$ denotes the language generated by the grammar G from the nonterminal X_i.

Structural Recursion

Proposition 3.2:
Let $G = (N, T, P, S)$ be a *unambiguous* (why?) context-free grammar. A function f is well-defined on $L(G)$ (that is, unambiguously defined) whenever these 2 properties are satisfied:

1. (base cases)
   f is well-defined on the words $w' \in T^*$ for each rule $X ::= w'$ in P.

2. (step cases)
   If $X ::= w_0 X_0 w_1 \ldots w_n X_n w_{n+1}$ is a rule in P then $f(w_0 w_0' w_1 \ldots w_n w_n' w_{n+1})$ is well-defined, assuming that each of the $f(w_i')$ is well-defined.

Substitution Revisited

Q: Does Proposition 3.2 justify that our homomorphic extension

$$\mathit{apply} : \mathsf{F}_\Sigma(X) \times (X \to \mathsf{T}_\Sigma(X)) \quad \to \quad \mathsf{F}_\Sigma(X),$$

with $\mathit{apply}(\phi, \sigma)$ denoted by $\phi\sigma$, of a substitution is well-defined?

A: We have two problems here. One is that *"fresh"* is (deliberately) left unspecified. That can be easily fixed by adding an extra variable counter argument to the apply function.

Substitution Revisited

The second problem is that Proposition 3.2 applies to unary functions only. The standard solution to this problem is to curryfy, that is, to consider the binary function as a unary function producing a unary (residual) function as a result:

$$\mathit{apply} : \mathsf{F}_{\Sigma}(X) \quad\rightarrow\quad ((X \rightarrow \mathsf{T}_{\Sigma}(X)) \rightarrow \mathsf{F}_{\Sigma}(X))$$

where we have denoted $(\mathit{apply}(\phi))(\sigma)$ as $\phi\sigma$.

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.

Structures

A Σ-algebra (also called Σ-interpretation or Σ-structure) is a triple

$$\mathcal{A} = (U_\mathcal{A},\ (f_\mathcal{A} : U_\mathcal{A}^n \to U_\mathcal{A})_{f/n \in \Omega},\ (P_\mathcal{A} \subseteq U_\mathcal{A}^m)_{P/m \in \Pi})$$

where $U_\mathcal{A} \neq \emptyset$ is a set, called the universe of $\mathcal{A}$.

By Σ-Alg we denote the class of all Σ-algebras.

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, also called a valuation (over a given Σ-algebra $\mathcal{A}$), is a map $\beta : X \to U_{\mathcal{A}}$.

Variable assignments are the semantic counterparts of substitutions.

Value of a Term in $\mathcal{A}$ with Respect to β

By structural induction we define

$$\mathcal{A}(\beta) : \mathsf{T}_\Sigma(X) \to U_\mathcal{A}$$

as follows:

$$\begin{aligned}
\mathcal{A}(\beta)(x) &= \beta(x), && x \in X \\
\mathcal{A}(\beta)(f(s_1, \dots, s_n)) &= f_\mathcal{A}(\mathcal{A}(\beta)(s_1), \dots, \mathcal{A}(\beta)(s_n)), && f/n \in \Omega
\end{aligned}$$

Value of a Term in $\mathcal{A}$ with Respect to β

In the scope of a quantifier we need to evaluate terms with respect to modified assignments. To that end, let $\beta[x \mapsto a] : X \to U_{\mathcal{A}}$, for $x \in X$ and $a \in \mathcal{A}$, 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 $\mathcal{A}$ with Respect to β

$\mathcal{A}(\beta) : \mathrm{F}_\Sigma(X) \to \{0,1\}$ is defined inductively as follows:

$$\begin{aligned}
\mathcal{A}(\beta)(\bot) &= 0 \\
\mathcal{A}(\beta)(\top) &= 1 \\
\mathcal{A}(\beta)(P(s_1,\ldots,s_n)) &= 1 \quad\Leftrightarrow\quad (\mathcal{A}(\beta)(s_1),\ldots,\mathcal{A}(\beta)(s_n)) \in P_{\mathcal{A}} \\
\mathcal{A}(\beta)(s \approx t) &= 1 \quad\Leftrightarrow\quad \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t) \\
\mathcal{A}(\beta)(\neg\phi) &= 1 \quad\Leftrightarrow\quad \mathcal{A}(\beta)(\phi) = 0 \\
\mathcal{A}(\beta)(\phi\rho\psi) &= \mathrm{B}_\rho(\mathcal{A}(\beta)(\phi), \mathcal{A}(\beta)(\psi)) \\
& \text{with } \mathrm{B}_\rho \text{ the Boolean function associated with } \rho \\
\mathcal{A}(\beta)(\forall x\phi) &= \min_{a\in U}\{\mathcal{A}(\beta[x \mapsto a])(\phi)\} \\
\mathcal{A}(\beta)(\exists x\phi) &= \max_{a\in U}\{\mathcal{A}(\beta[x \mapsto a])(\phi)\}
\end{aligned}$$

Example

The "Standard" Interpretation for Peano Arithmetic:

$$\begin{aligned}
U_{\mathbb{N}} &= \{0, 1, 2, \ldots\} \\
0_{\mathbb{N}} &= 0 \\
s_{\mathbb{N}} &: n \mapsto n + 1 \\
+_{\mathbb{N}} &: (n, m) \mapsto n + m \\
*_{\mathbb{N}} &: (n, m) \mapsto n * m \\
\leq_{\mathbb{N}} &= \{\, (n, m) \mid n \text{ less than or equal to } m \,\} \\
<_{\mathbb{N}} &= \{\, (n, m) \mid n \text{ less than } m \,\}
\end{aligned}$$

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

Example

Values over $\mathbb{N}$ for Sample Terms and Formulas:

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

$$\begin{aligned}
\mathbb{N}(\beta)(s(x) + s(0)) &= 3 \\
\mathbb{N}(\beta)(x + y \approx s(y)) &= 1 \\
\mathbb{N}(\beta)(\forall x, y(x + y \approx y + x)) &= 1 \\
\mathbb{N}(\beta)(\forall z\ z \leq y) &= 0 \\
\mathbb{N}(\beta)(\forall x \exists y\ x < y) &= 1
\end{aligned}$$

3.3 Models, Validity, and Satisfiability

ϕ is valid in $\mathcal{A}$ under assignment β:

$$\mathcal{A}, \beta \models \phi \quad :\Leftrightarrow \quad \mathcal{A}(\beta)(\phi) = 1$$

ϕ is valid in $\mathcal{A}$ ($\mathcal{A}$ is a model of ϕ):

$$\mathcal{A} \models \phi \quad :\Leftrightarrow \quad \mathcal{A}, \beta \models \phi, \text{ for all } \beta \in X \to U_{\mathcal{A}}$$

ϕ is valid (or is a tautology):

$$\models \phi \quad :\Leftrightarrow \quad \mathcal{A} \models \phi, \text{ for all } \mathcal{A} \in \Sigma\text{-Alg}$$

ϕ is called satisfiable iff there exist $\mathcal{A}$ and β such that $\mathcal{A}, \beta \models \phi$. Otherwise ϕ is called unsatisfiable.

Substitution Lemma

The following propositions, to be proved by structural induction, hold for all Σ-algebras $\mathcal{A}$, assignments β, and substitutions σ.

Lemma 3.3:
For any Σ-term t

$$\mathcal{A}(\beta)(t\sigma) = \mathcal{A}(\beta \circ \sigma)(t),$$

where $\beta \circ \sigma : X \to \mathcal{A}$ is the assignment $\beta \circ \sigma(x) = \mathcal{A}(\beta)(x\sigma)$.

Proposition 3.4:
For any Σ-formula ϕ, $\mathcal{A}(\beta)(\phi\sigma) = \mathcal{A}(\beta \circ \sigma)(\phi)$.

Substitution Lemma

Corollary 3.5:
$\mathcal{A}, \beta \models \phi\sigma \quad \Leftrightarrow \quad \mathcal{A}, \beta \circ \sigma \models \phi$

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

Entailment and Equivalence

ϕ entails (implies) ψ (or ψ is a consequence of ϕ), written $\phi \models \psi$, if for all $\mathcal{A} \in \Sigma$-Alg and $\beta \in X \to U_{\mathcal{A}}$, whenever $\mathcal{A}, \beta \models \phi$, then $\mathcal{A}, \beta \models \psi$.

ϕ and ψ are called equivalent, written $\phi \models\mid \psi$, if for all $\mathcal{A} \in \Sigma$-Alg and $\beta \in X \to U_{\mathcal{A}}$ we have $\mathcal{A}, \beta \models \phi \quad \Leftrightarrow \quad \mathcal{A}, \beta \models \psi$.

Entailment and Equivalence

Proposition 3.6:
ϕ entails ψ iff $(\phi \rightarrow \psi)$ is valid

Proposition 3.7:
ϕ and ψ are equivalent iff $(\phi \leftrightarrow \psi)$ is valid.

Extension to sets of formulas N in the "natural way", e. g.,
$N \models \phi$

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma$-Alg and $\beta \in X \rightarrow U_{\mathcal{A}}$: if $\mathcal{A}, \beta \models \psi$, for all $\psi \in N$, then $\mathcal{A}, \beta \models \phi$.

Validity vs. Unsatisfiability

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

Proposition 3.8:
Let ϕ and ψ be formulas, let N be a set of formulas. Then

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

(ii) $\phi \models \psi$ if and only if $\phi \wedge \neg\psi$ is unsatisfiable.

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

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

Theory of a Structure

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

$$Th(\mathcal{A}) = \{\, \psi \in \mathsf{F}_\Sigma(X) \mid \mathcal{A} \models \psi \,\}$$

Problem of axiomatizability:

For which structures $\mathcal{A}$ can one axiomatize $Th(\mathcal{A})$, that is, can one write down a formula ϕ (or a recursively enumerable set ϕ of formulas) such that

$$Th(\mathcal{A}) = \{\, \psi \mid \phi \models \psi \,\}?$$

Analogously for sets of structures.

Two Interesting Theories

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

Two Interesting Theories

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

Note: The choice of signature can make a big difference with regard to the computational complexity of theories.

3.4 Algorithmic Problems

Validity(ϕ): $\models \phi$?

Satisfiability(ϕ): ϕ satisfiable?

Entailment(ϕ,ψ): does ϕ entail ψ?

Model($\mathcal{A}$,ϕ): $\mathcal{A} \models \phi$?

Solve($\mathcal{A}$,ϕ): find an assignment β such that $\mathcal{A}, \beta \models \phi$.

Solve(ϕ): find a substitution σ such that $\models \phi\sigma$.

Abduce(ϕ): find ψ with “certain properties” such that $\psi \models \phi$.

Gödel's Famous Theorems

1. For most signatures Σ, validity is undecidable for Σ-formulas. (Later by Turing: Encode Turing machines as Σ-formulas.)
2. For each signature Σ, the set of valid Σ-formulas is recursively enumerable. (We will prove this by giving complete deduction systems.)
3. For $\Sigma = \Sigma_{PA}$ and $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *)$, the theory $Th(\mathbb{N}_*)$ is not recursively enumerable.

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

Q: Can you think of any fragments of first-order logic for which validity is decidable?

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 time polynomial in the size of the structure and the formula.

Plan

Lift superposition from propositional logic to first-order logic.

3.5 Normal Forms and Skolemization

Study of normal forms motivated by

- reduction of logical concepts,
- efficient data structures for theorem proving,
- satisfiability preserving transformations (renaming),
- Skolem's and Herbrand's theorem.

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

$$Q_1 x_1 \ldots Q_n x_n \; \phi,$$

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

Prenex Normal Form (Traditional)

Computing prenex normal form by the rewrite system $\Rightarrow_P$:

$$
\begin{aligned}
\phi[(\psi_1 \leftrightarrow \psi_2)]_p &\Rightarrow_P \phi[(\psi_1 \rightarrow \psi_2) \land (\psi_2 \rightarrow \psi_1)]_p \\
\phi[\neg Qx\psi_1]_p &\Rightarrow_P \phi[\overline{Q}x\neg\psi_1]_p \\
\phi[((Qx\psi_1)\ \rho\ \psi_2)]_p &\Rightarrow_P \phi[Qy(\psi_1\{x \mapsto y\}\ \rho\ \psi_2)]_p,\ \ \rho \in \{\land, \lor\} \\
\phi[((Qx\psi_1) \rightarrow \psi_2)]_p &\Rightarrow_P \phi[\overline{Q}y(\psi_1\{x \mapsto y\} \rightarrow \psi_2)]_p, \\
\phi[(\psi_1\ \rho\ (Qx\psi_2))]_p &\Rightarrow_P \phi[Qy(\psi_1\ \rho\ \psi_2\{x \mapsto y\})]_p,\ \ \rho \in \{\land, \lor, \rightarrow\}
\end{aligned}
$$

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):

$$\forall x_1, \ldots, x_n \exists y \, \phi \quad \Rightarrow_S \quad \forall x_1, \ldots, x_n \, \phi\{y \mapsto f(x_1, \ldots, x_n)\}$$

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

Skolemization

Together: $\phi \Rightarrow_P^* \underbrace{\psi}_{\text{prenex}} \Rightarrow_S^* \underbrace{\chi}_{\text{prenex, no } \exists}$

Theorem 3.9:

Let ϕ, ψ, and χ as defined above and closed. Then

(i) ϕ and ψ are equivalent.

(ii) $\chi \models \psi$ but the converse is not true in general.

(iii) ψ satisfiable (Σ-Alg) $\Leftrightarrow$ χ satisfiable (Σ'-Alg) where $\Sigma' = (\Omega \cup SKF, \Pi)$, if $\Sigma = (\Omega, \Pi)$.

The Complete Picture

$$
\begin{array}{llll}
\phi & \Rightarrow^*_P & Q_1 y_1 \ldots Q_n y_n \, \psi & (\psi \text{ quantifier-free}) \\[1em]
 & \Rightarrow^*_S & \forall x_1, \ldots, x_m \, \chi & (m \le n, \ \chi \text{ quantifier-free}) \\[1em]
 & \Rightarrow^*_{OCNF} & \underbrace{\underbrace{\forall x_1, \ldots, x_m}_{\text{leave out}} \ \bigwedge_{i=1}^{k} \ \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}}_{\phi'} &
\end{array}
$$

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

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

The Complete Picture

Theorem 3.10:
Let ϕ be closed. Then $\phi' \models \phi$. (The converse is not true in general.)

Theorem 3.11:
Let ϕ be closed. Then ϕ is satisfiable iff ϕ' 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.

- 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 Small Skolem Functions

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

- rename beneficial subformulas
- produce a negation normal form (NNF)
- apply miniscoping
- rename all variables
- skolemize

Formula renaming

We extend the machinery from propositional to first-order logic: $\nu(\forall x\, \phi) = \nu(\exists x\, \phi) = \nu(\phi)$ and $\bar{\nu}(\forall x\, \phi) = \bar{\nu}(\exists x\, \phi) = \bar{\nu}(\phi)$.

Introduce top-down fresh predicates for beneficial subformulas:

$$\psi[\phi]_p \;\Rightarrow_{\mathrm{OCNF}}\; \psi[P(x_1, \ldots, x_n)]_p \wedge \mathrm{def}(\psi, p, P)$$

where $\{x_1, \ldots, x_n\}$ are the free variables in ϕ, P/n is a predicate new to $\psi[\phi]_p$, $\nu(\psi[\phi]_p) > \nu(\psi[P]_p \wedge \mathrm{def}(\psi, p, P))$, and $\mathrm{def}(\psi, p, P)$ is defined polarity dependent analogous to the propositional case:

$$\mathrm{def}(\psi, p, P) := \forall x_1, \ldots, x_n\ [\psi|_p \circ P(x_1, \ldots, x_n)]$$

where $\circ \in \{\rightarrow, \leftrightarrow, \leftarrow\}$.

Negation Normal Form (NNF)

Apply the rewrite system $\Rightarrow_{\mathsf{NNF}}$:

$$\phi[\psi_1 \leftrightarrow \psi_2]_p \;\Rightarrow_{\mathsf{NNF}}\; \phi[(\psi_1 \rightarrow \psi_2) \land (\psi_2 \rightarrow \psi_1)]_p$$

if $\mathrm{pol}(\phi, p) = 1$ or $\mathrm{pol}(\phi, p) = 0$

$$\phi[\psi_1 \leftrightarrow \psi_2]_p \;\Rightarrow_{\mathsf{NNF}}\; \phi[(\psi_1 \land \psi_2) \lor (\neg\psi_2 \land \neg\psi_1)]_p$$

if $\mathrm{pol}(\phi, p) = -1$

Negation Normal Form (NNF)

$$\begin{aligned}
\phi[\neg Qx\ \psi]_p &\Rightarrow_{\mathsf{NNF}} \phi[\overline{Q}x\ \neg\psi]_p \\
\phi[\neg(\psi_1 \vee \psi_2)]_p &\Rightarrow_{\mathsf{NNF}} \phi[\neg\psi_1 \wedge \neg\psi_2]_p \\
\phi[\neg(\psi_1 \wedge \psi_2)]_p &\Rightarrow_{\mathsf{NNF}} \phi[\neg\psi_1 \vee \neg\psi_2]_p \\
\phi[\psi_1 \rightarrow \psi_2]_p &\Rightarrow_{\mathsf{NNF}} \phi[\neg\psi_1 \vee \psi_2]_p \\
\phi[\neg\neg\psi]_p &\Rightarrow_{\mathsf{NNF}} \phi[\psi]_p
\end{aligned}$$

Miniscoping

Apply the rewrite relation $\Rightarrow_{\mathsf{MS}}$. For the rules below we assume that x occurs freely in ψ_1, ψ_3, but x does not occur freely in ψ_2:

$$
\begin{array}{rcl}
\phi[Qx\,(\psi_1 \wedge \psi_2)]_p & \Rightarrow_{\mathsf{MS}} & \phi[(Qx\,\psi_1) \wedge \psi_2]_p \\
\phi[Qx\,(\psi_2 \vee \psi_2)]_p & \Rightarrow_{\mathsf{MS}} & \phi[(Qx\,\psi_1) \vee \psi_2]_p \\
\phi[\forall x\,(\psi_1 \wedge \psi_3)]_p & \Rightarrow_{\mathsf{MS}} & \phi[(\forall x\,\psi_1) \wedge (\forall x\,\psi_3)]_p \\
\phi[\exists x\,(\psi_1 \vee \psi_3)]_p & \Rightarrow_{\mathsf{MS}} & \phi[(\exists x\,\psi_1) \vee (\exists x\,\psi_3)]_p
\end{array}
$$

Variable Renaming

Rename all variables in ϕ such that there are no two different positions p, q with $\phi|_p = Qx\,\psi_1$ and $\phi|_q = Q'x\,\psi_2$.

Standard Skolemization

Apply the rewrite rule:

$$\phi[\exists x\, \psi]_p \quad \Rightarrow_{\mathsf{SK}} \quad \phi[\psi\{x \mapsto f(y_1, \dots, y_n)\}]_p$$

where p has minimal length,

$\{y_1, \dots, y_n\}$ are the free variables in $\exists x\, \psi$,

f/n is a new function symbol to ϕ

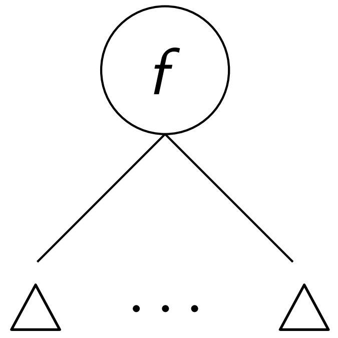
3.7 Herbrand Interpretations

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 $\mathcal{A}$ such that

- $U_{\mathcal{A}} = \mathsf{T}_{\Sigma}$ (= the set of ground terms over Σ)
- $f_{\mathcal{A}} : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n), \; f/n \in \Omega$

$$f_{\mathcal{A}}(\triangle, \ldots, \triangle) =$$



Herbrand Interpretations

In other words, *values are fixed* to be ground terms and *functions are fixed* to be the term constructors. Only predicate symbols $P/m \in \Pi$ may be freely interpreted as relations $P_{\mathcal{A}} \subseteq \mathsf{T}_{\Sigma}^{m}$.

Proposition 3.12:
Every set of ground atoms I uniquely determines a Herbrand interpretation $\mathcal{A}$ via

$$(s_1, \ldots, s_n) \in P_{\mathcal{A}} \quad :\Leftrightarrow \quad P(s_1, \ldots, s_n) \in I$$

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

Herbrand Interpretations

Example: $\Sigma_{Pres} = (\{0/0, s/1, +/2\},\ \{</2, \leq/2\})$

$\mathbb{N}$ as Herbrand interpretation over Σ_{Pres}:

$$
\begin{aligned}
I = \{ \quad & 0 \leq 0,\ 0 \leq s(0),\ 0 \leq s(s(0)),\ \ldots, \\
& 0 + 0 \leq 0,\ 0 + 0 \leq s(0),\ \ldots, \\
& \ldots,\ (s(0) + 0) + s(0) \leq s(0) + (s(0) + s(0)) \\
& \ldots \\
& s(0) + 0 < s(0) + 0 + 0 + s(0) \\
& \ldots\}
\end{aligned}
$$

Existence of Herbrand Models

A Herbrand interpretation I is called a Herbrand model of ϕ, if $I \models \phi$.

Theorem 3.13 (Herbrand):
Let N be a set of Σ-clauses.

$$\begin{array}{lcl} N \text{ satisfiable} & \Leftrightarrow & N \text{ has a Herbrand model (over } \Sigma) \\ & \Leftrightarrow & G_\Sigma(N) \text{ has a Herbrand model (over } \Sigma) \end{array}$$

where $G_\Sigma(N) = \{\, C\sigma \text{ ground clause} \mid C \in N,\ \sigma : X \to \mathsf{T}_\Sigma \,\}$ is the set of ground instances of N.

[The proof will be given below in the context of the completeness proof for superposition.]

Example of a G_Σ

For Σ_{Pres} one obtains for

$$C = (x < y) \vee (y \leq s(x))$$

the following ground instances:

$(0 < 0) \vee (0 \leq s(0))$
$(s(0) < 0) \vee (0 \leq s(s(0)))$
$\ldots$
$(s(0) + s(0) < s(0) + 0) \vee (s(0) + 0 \leq s(s(0) + s(0)))$
$\ldots$

3.8 Inference Systems and Proofs

Inference systems Γ (proof calculi) are sets of tuples

$$(\phi_1, \ldots, \phi_n, \phi_{n+1}),\ \ n \geq 0,$$

called inferences, and written

$$\underbrace{\frac{\overbrace{\phi_1 \ \ \ldots \ \ \phi_n}^{\text{premises}}}{\phi_{n+1}}}_{\text{conclusion}} \ .$$

Clausal inference system: premises and conclusions are clauses.

One also considers inference systems over other data structures.

Inference Systems

Inference systems Γ are short hands for rewrite systems over sets of formulas. If N is a set of formulas, then

$$\frac{\overbrace{\phi_1 \quad \ldots \quad \phi_n}^{\text{premises}}}{\underbrace{\phi_{n+1}}_{\text{conclusion}}} \quad \textit{side condition}$$

is a shorthand for

$$N \cup \{\phi_1 \quad \ldots \quad \phi_n\} \quad \Rightarrow_\Gamma \quad N \cup \{\phi_1 \quad \ldots \quad \phi_n\} \cup \{\phi_{n+1}\}$$

if *side condition*

Proofs

A proof in Γ of a formula ϕ from a a set of formulas N (called assumptions) is a sequence $\phi_1, \ldots, \phi_k$ of formulas where

(i) $\phi_k = \phi$,

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

$$\frac{\phi_{i_1} \quad \ldots \quad \phi_{i_{n_i}}}{\phi_i}$$

in Γ, such that $0 \leq i_j < i$, for $1 \leq j \leq n_i$.

Soundness and Completeness

Provability $\vdash_\Gamma$ of ϕ from N in Γ: $N \vdash_\Gamma \phi$ if there exists a proof Γ of ϕ from N.

Γ is called sound

$$\frac{\phi_1 \quad \ldots \quad \phi_n}{\phi} \in \Gamma \quad \text{implies } \phi_1, \ldots, \phi_n \models \phi$$

Γ is called complete

$$N \models \phi \quad \text{implies } N \vdash_\Gamma \phi$$

Γ is called refutationally complete

$$N \models \bot \quad \text{implies } N \vdash_\Gamma \bot$$

Soundness and Completeness

Proposition 3.14:

(i) Let Γ be sound. Then $N \vdash_\Gamma \phi$ implies $N \models \phi$

(ii) $N \vdash_\Gamma \phi$ implies there exist finitely many clauses $\phi_1, \ldots, \phi_n \in N$ such that $\phi_1, \ldots, \phi_n \vdash_\Gamma \phi$

Proofs as Trees

markings $\widehat{=}$ formulas

leaves $\widehat{=}$ assumptions and axioms

other nodes $\widehat{=}$ inferences: conclusion $\widehat{=}$ ancestor

premises $\widehat{=}$ direct descendants

$$\dfrac{P(f(c)) \qquad \dfrac{\dfrac{\dfrac{P(f(c)) \lor Q(b) \qquad \dfrac{\dfrac{P(f(c)) \lor Q(b) \quad \neg P(f(c)) \lor \neg P(f(c)) \lor Q(b)}{\neg P(f(c)) \lor Q(b) \lor Q(b)}}{\neg P(f(c)) \lor Q(b)}}{Q(b) \lor Q(b)}}{Q(b)} \qquad \neg P(f(c)) \lor \neg Q(b)}{\neg P(f(c))}}{\bot}$$

3.9 Ground Superposition

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 and recall partly the material from Section 2.5 for first-order ground clauses.

The Resolution Calculus *Res*

Resolution inference rule:

$$\frac{D \vee A \qquad \neg A \vee C}{D \vee C}$$

Terminology: $D \vee C$: resolvent; A: resolved atom

For Superposition (*Sup*): A strictly maximal, $\neg A$ maximal

(Positive) factorization inference rule:

$$\frac{C \vee A \vee A}{C \vee A}$$

For Superposition (*Sup*): A maximal

The Resolution Calculus *Res*

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 "$\vee$" as associative and commutative, hence A and $\neg A$ can occur anywhere in the clauses; moreover, when we write $C \vee A$, etc., this includes unit clauses, that is, $C = \bot$.

Sample Refutation

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

Soundness of Resolution

Theorem 3.15:

Propositional resolution is sound.

Proof:

Let $\mathcal{B} \in \Sigma$-Alg. To be shown:

(i) for resolution: $\mathcal{B} \models D \vee A,\ \mathcal{B} \models C \vee \neg A \ \Rightarrow\ \mathcal{B} \models D \vee C$

(ii) for factorization: $\mathcal{B} \models C \vee A \vee A \ \Rightarrow\ \mathcal{B} \models C \vee A$

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

If $\mathcal{B} \models A$, then $\mathcal{B} \models C$, hence $\mathcal{B} \models D \vee C$.

Otherwise, $\mathcal{B} \models \neg A$, then $\mathcal{B} \models D$, and again $\mathcal{B} \models D \vee C$.

(ii): even simpler. □

Soundness of Resolution

Note: In propositional logic (ground clauses) we have:

1. $\mathcal{B} \models L_1 \vee \ldots \vee L_n$ iff there exists i: $\mathcal{B} \models L_i$.

2. $\mathcal{B} \models A$ or $\mathcal{B} \models \neg A$.

This does not hold for formulas with variables!

Closure of Clause Sets under *Res*

$$\begin{aligned}
\mathit{Res}(N) &= \{\, C \mid C \text{ is conclusion of an inference in } \mathit{Res} \\
&\qquad\qquad \text{with premises in } N \,\} \\
\mathit{Res}^0(N) &= N \\
\mathit{Res}^{n+1}(N) &= \mathit{Res}(\mathit{Res}^n(N)) \cup \mathit{Res}^n(N), \ \text{ for } n \geq 0 \\
\mathit{Res}^*(N) &= \textstyle\bigcup_{n\geq 0} \mathit{Res}^n(N)
\end{aligned}$$

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

Closure of Clause Sets under *Res*

Proposition 3.16:

(i) $Res^*(N)$ is saturated.

(ii) Res is refutationally complete, iff for each set N of ground clauses:

$$N \models \bot \text{ iff } \bot \in Res^*(N)$$

Construction of Interpretations

Done the same way as for propositional logic: ground atoms play the rôle of propositional variables.

Model Existence Theorem

Theorem 3.17 (Bachmair & Ganzinger 1990):
Let $\succ$ be a clause ordering, let N be saturated w. r. t. *Res* (or *Sup*), and suppose that $\bot \notin N$. Then $N_{\mathcal{I}}^{\succ} \models N$.

Corollary 3.18:
Let N be saturated w. r. t. *Res*. Then $N \models \bot \Leftrightarrow \bot \in N$.

Compactness of Propositional Logic

Theorem 3.19 (Compactness):
Let N be a set of propositional (or first-order ground) formulas. Then N is unsatisfiable, if and only if some finite subset $M \subseteq N$ is unsatisfiable.

Proof:
"$\Leftarrow$": trivial. "$\Rightarrow$": Let N be unsatisfiable.
$\Rightarrow$ $\mathit{Res}^*(N)$ unsatisfiable
$\Rightarrow$ $\bot \in \mathit{Res}^*(N)$ by refutational completeness of resolution
$\Rightarrow$ $\exists n \geq 0 : \bot \in \mathit{Res}^n(N)$
$\Rightarrow$ $\bot$ has a finite resolution proof P;
choose M as the set of assumptions in P. □

3.10 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 DPLL procedure.

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

General Resolution through Instantiation

Idea: instantiate clauses appropriately:

$$\begin{tikzcd}
P(z',z') \vee \neg Q(z) \arrow[d, "{[a/z', f(a,b)/z]}"'] & & & \neg P(a,y) \arrow[dl, "{[a/y]}"'] \arrow[dr, "{[b/y]}"] & & & P(x',b) \vee Q(f(x',x)) \arrow[d, "{[a/x', b/x]}"] \\
P(a,a) \vee \neg Q(f(a,b)) \arrow[dr] & & \neg P(a,a) \arrow[dl] & & \neg P(a,b) \arrow[dr] & & P(a,b) \vee Q(f(a,b)) \arrow[dl] \\
 & \neg Q(f(a,b)) \arrow[d] & & & & Q(f(a,b)) \arrow[d] & \\
 & {} \arrow[drr] & & & & {} \arrow[dll] & \\
 & & & \perp & & &
\end{tikzcd}$$

General Resolution through Instantiation

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:

Do not instantiate more than necessary to get complementary literals.

General Resolution through Instantiation

$$\begin{tikzcd}
P(z', z') \vee \neg Q(z) \arrow[d, "{[a/z']}"'] & & & \neg P(a, y) \arrow[dl, "{[a/y]}"'] \arrow[dr, "{[b/y]}"] & & & P(x', b) \vee Q(f(x', x)) \arrow[d, "{[a/x']}"] \\
P(a, a) \vee \neg Q(z) \arrow[dr] & & \neg P(a, a) \arrow[dl] & & \neg P(a, b) \arrow[dr] & & P(a, b) \vee Q(f(a, x)) \arrow[dl] \\
& \neg Q(z) \arrow[d, "{[f(a, x)/z]}"'] & & & & Q(f(a, x)) \arrow[d] & \\
& \neg Q(f(a, x)) \arrow[drr] & & & & Q(f(a, x)) \arrow[dll] & \\
& & & \perp & & &
\end{tikzcd}$$

Lifting Principle

Problem: Make saturation of infinite sets of clauses as they arise from taking the (ground) instances of finitely many general clauses (with variables) effective and efficient.

Idea (Robinson 1965):

- Resolution for general clauses:
- *Equality* of ground atoms is generalized to *unifiability* of general atoms;
- Only compute *most general* (minimal) unifiers (mgu).

Lifting Principle

Significance: The advantage of the method in (Robinson 1965) compared with (Gilmore 1960) is that unification enumerates only those instances of clauses that participate in an inference. Moreover, clauses are not right away instantiated into ground clauses. Rather they are instantiated only as far as required for an inference. Inferences with non-ground clauses in general represent infinite sets of ground inferences which are computed simultaneously in a single step.

Resolution for General Clauses

General binary resolution *Res*:

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

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

Resolution for General Clauses

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.

Unification

Let $E = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ (s_i, t_i terms or atoms) a multiset of equality problems. A substitution σ 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.

Unification

A substitution σ is called more general than a substitution τ, denoted by $\sigma \leq \tau$, if there exists a substitution ρ such that $\rho \circ \sigma = \tau$, where $(\rho \circ \sigma)(x) := (x\sigma)\rho$ is the composition of σ and ρ 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 $\mathrm{mgu}(E)$.

Unification

Proposition 3.20:

(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 σ is called idempotent, if $\sigma \circ \sigma = \sigma$.

Proposition 3.21:
σ is idempotent iff $\mathit{dom}(\sigma) \cap \mathit{codom}(\sigma) = \emptyset$.

Rule-Based Naive Standard Unification

$$
\begin{array}{rcl}
t \doteq t, E & \Rightarrow_{SU} & E \\[1em]
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E & \Rightarrow_{SU} & s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\[1em]
f(\ldots) \doteq g(\ldots), E & \Rightarrow_{SU} & \bot \\[1em]
x \doteq t, E & \Rightarrow_{SU} & x \doteq t, E\{x \mapsto t\} \\
 & & \text{if } x \in var(E), x \notin var(t) \\[1em]
x \doteq t, E & \Rightarrow_{SU} & \bot \\
 & & \text{if } x \neq t, x \in var(t) \\[1em]
t \doteq x, E & \Rightarrow_{SU} & x \doteq t, E \\
 & & \text{if } t \notin X
\end{array}
$$

SU: Main Properties

If $E = x_1 \doteq u_1, \ldots, x_k \doteq u_k$, with x_i pairwise distinct, $x_i \notin var(u_j)$, 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 3.22:
If E is a solved form then σ_E is an mgu of E.

SU: Main Properties

Theorem 3.23:

1. If $E \Rightarrow_{SU} E'$ then σ is a unifier of E iff σ is a unifier of E'
2. If $E \Rightarrow^{*}_{SU} \bot$ 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 σ 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 3.22. □

Main Unification Theorem

Theorem 3.24:
E is unifiable if and only if there is a most general unifier σ of E, such that σ is idempotent and $\mathit{dom}(\sigma) \cup \mathit{codom}(\sigma) \subseteq \mathit{var}(E)$.

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.

Rule-Based Polynomial Unification

$$
\begin{aligned}
t \doteq t, E \quad &\Rightarrow_{PU} \quad E \\
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E \quad &\Rightarrow_{PU} \quad s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) \doteq g(\ldots), E \quad &\Rightarrow_{PU} \quad \bot \\
x \doteq y, E \quad &\Rightarrow_{PU} \quad x \doteq y, E\{x \mapsto y\} \\
& \qquad\quad \text{if } x \in var(E), x \neq y \\
x_1 \doteq t_1, \ldots, x_n \doteq t_n, E \quad &\Rightarrow_{PU} \quad \bot \\
& \qquad\quad \text{if there are positions } p_i \text{ with} \\
& \qquad\quad t_i/p_i = x_{i+1}, t_n/p_n = x_1 \\
& \qquad\quad \text{and some } p_i \neq \epsilon
\end{aligned}
$$

Rule-Based Polynomial Unification

$$
\begin{aligned}
x \doteq t, E \quad &\Rightarrow_{PU} \quad \bot \\
&\phantom{\Rightarrow_{PU}} \quad \text{if } x \neq t, x \in var(t) \\
t \doteq x, E \quad &\Rightarrow_{PU} \quad x \doteq t, E \\
&\phantom{\Rightarrow_{PU}} \quad \text{if } t \notin X \\
x \doteq t, x \doteq s, E \quad &\Rightarrow_{PU} \quad x \doteq t, t \doteq s, E \\
&\phantom{\Rightarrow_{PU}} \quad \text{if } t, s \notin X \text{ and } |t| \leq |s|
\end{aligned}
$$

Properties of PU

Theorem 3.25:

1. If $E \Rightarrow_{PU} E'$ then σ is a unifier of E iff σ 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 form 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 $\{x_1 \mapsto t_1\} \circ \cdots \circ \{x_k \mapsto t_k\}$.

Lifting Lemma

Lemma 3.26:

Let C and D be variable-disjoint clauses. If

$$\frac{\begin{tikzcd} D \arrow[d, "\sigma"] & C \arrow[d, "\rho"] \\ D\sigma & C\rho \end{tikzcd}}{C'}$$

[propositional resolution]

then there exists a substitution τ such that

$$\begin{tikzcd} \dfrac{D \qquad C}{C''} \arrow[d, "\tau"] \\ C' = C''\tau \end{tikzcd}$$

[general resolution]

Lifting Lemma

An analogous lifting lemma holds for factorization.

Saturation of Sets of General Clauses

Corollary 3.27:

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

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

Saturation of Sets of General Clauses

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 $Res(N)$ nor $G_\Sigma(N)$.)

Let $C' \in Res(G_\Sigma(N))$, meaning (i) there exist resolvable ground instances $D\sigma$ and $C\rho$ of N with resolvent C', or else (ii) C' is a factor of a ground instance $C\sigma$ of C.

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

Case (ii): Similar. □

Herbrand's Theorem

Lemma 3.28:
Let N be a set of Σ-clauses, let $\mathcal{A}$ be an interpretation. Then $\mathcal{A} \models N$ implies $\mathcal{A} \models G_\Sigma(N)$.

Lemma 3.29:
Let N be a set of Σ-clauses, let $\mathcal{A}$ be a *Herbrand* interpretation. Then $\mathcal{A} \models G_\Sigma(N)$ implies $\mathcal{A} \models N$.

Herbrand’s Theorem

Theorem 3.30 (Herbrand):
A set N of Σ-clauses is satisfiable if and only if it has a Herbrand model over Σ.

Proof:
The “$\Leftarrow$” part is trivial. For the “$\Rightarrow$” part let $N \not\models \bot$.

$$
\begin{aligned}
N \not\models \bot &\Rightarrow \bot \notin \mathit{Res}^*(N) && \text{(resolution is sound)}\\
&\Rightarrow \bot \notin G_\Sigma(\mathit{Res}^*(N)) &&\\
&\Rightarrow G_\Sigma(\mathit{Res}^*(N))_\mathcal{I} \models G_\Sigma(\mathit{Res}^*(N)) && \text{(Thm. 3.17; Cor. 3.27)}\\
&\Rightarrow G_\Sigma(\mathit{Res}^*(N))_\mathcal{I} \models \mathit{Res}^*(N) && \text{(Lemma 3.29)}\\
&\Rightarrow G_\Sigma(\mathit{Res}^*(N))_\mathcal{I} \models N && (N \subseteq \mathit{Res}^*(N)) \quad \Box
\end{aligned}
$$

The Theorem of Löwenheim-Skolem

Theorem 3.31 (Löwenheim–Skolem):
Let Σ be a countable signature and let S be a set of closed Σ-formulas. Then S is satisfiable iff S has a model over a countable universe.

Proof:
If both X and Σ are countable, then S can be at most countably infinite. Now generate, maintaining satisfiability, a set N of clauses from S. This extends Σ by at most countably many new Skolem functions to Σ'. As Σ' is countable, so is $T_{\Sigma'}$, the universe of Herbrand-interpretations over Σ'. Now apply Theorem 3.30. □

Refutational Completeness of General Resolution

Theorem 3.32:

Let N be a set of general clauses where $Res(N) \subseteq N$. Then

$$N \models \bot \Leftrightarrow \bot \in N.$$

Proof:

Let $Res(N) \subseteq N$. By Corollary 3.27: $Res(G_\Sigma(N)) \subseteq G_\Sigma(N)$

$$\begin{aligned} N \models \bot &\Leftrightarrow G_\Sigma(N) \models \bot && \text{(Lemma 3.28/3.29; Theorem 3.30)} \\ &\Leftrightarrow \bot \in G_\Sigma(N) && \text{(propositional resolution sound and complete)} \\ &\Leftrightarrow \bot \in N \quad \square \end{aligned}$$

Compactness of Predicate Logic

Theorem 3.33 (Compactness Theorem for First-Order Logic): Let S be a set of 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 $\mathit{Res}^*(N)$ is unsatisfiable. By Theorem 3.32, $\bot \in \mathit{Res}^*(N)$, and therefore $\bot \in \mathit{Res}^n(N)$ 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. $\square$

3.11 First-Order Superposition with Selection

Motivation: Search space for *Res* *very* large.

Ideas for improvement:

1. In the completeness proof (Model Existence Theorem 2.13) one only needs to resolve and factor maximal atoms
   $\Rightarrow$ if the calculus is restricted to inferences involving maximal atoms, the proof remains correct
   $\Rightarrow$ *ordering restrictions*

2. In the proof, it does not really matter with which negative literal an inference is performed
   $\Rightarrow$ choose a negative literal don't-care-nondeterministically
   $\Rightarrow$ *selection*

Selection Functions

A selection function is a mapping

$$\text{sel} : C \quad \mapsto \quad \text{set of occurrences of } \textit{negative} \text{ literals in } C$$

Example of selection with selected literals indicated as $\boxed{X}$:

$$\boxed{\neg A} \lor \neg A \lor B$$

$$\boxed{\neg B_0} \lor \boxed{\neg B_1} \lor A$$

Selection Functions

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.

Orderings for Terms, Atoms, Clauses

For first-order logic an ordering on the signature symbols is not sufficient to compare atoms, e.g., how to compare $P(a)$ and $P(b)$?

We propose the Knuth-Bendix Ordering for terms, atoms (with variables) which is then lifted as in the propositional case to literals and clauses.

The Knuth-Bendix Ordering (Simple)

Let $\Sigma = (\Omega, \Pi)$ be a finite signature, let $\succ$ be a total ordering ("precedence") on $\Omega \cup \Pi$, let $w : \Omega \cup \Pi \cup X \to \mathbb{R}^+$ be a weight function, satisfying $w(x) = w_0 \in \mathbb{R}^+$ for all variables $x \in X$ and $w(c) \geq w_0$ for all constants $c \in \Omega$.

The weight function w can be extended to terms (atoms) as follows:

$$w(f(t_1, \dots, t_n)) = w(f) + \sum_{1 \leq i \leq n} w(t_i)$$

$$w(P(t_1, \dots, t_n)) = w(P) + \sum_{1 \leq i \leq n} w(t_i)$$

The Knuth-Bendix Ordering (Simple)

The Knuth-Bendix ordering $\succ_{\mathrm{kbo}}$ on $\mathrm{T}_\Sigma(X)$ (atoms) induced by $\succ$ and w is defined by: $s \succ_{\mathrm{kbo}} t$ 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) $s = f(s_1,\dots,s_m)$, $t = g(t_1,\dots,t_n)$, and $f \succ g$, or

(b) $s = f(s_1,\dots,s_m)$, $t = f(t_1,\dots,t_m)$, and $(s_1,\dots,s_m)\ (\succ_{\mathrm{kbo}})_{\mathrm{lex}}\ (t_1,\dots,t_m)$.

where $\#(s,t) = |\{p \mid t|_p = s\}|$.

The Knuth-Bendix Ordering (Simple)

Proposition 3.34:

The Knuth-Bendix ordering $\succ_{\mathsf{kbo}}$ is

(1) a strict partial well-founded ordering on terms (atoms).

(2) stable under substitution: if $s \succ_{\mathsf{kbo}} t$ then $s\sigma \succ_{\mathsf{kbo}} t\sigma$ for any σ.

(3) total on ground terms (ground atoms).

Superposition Calculus $\mathit{Sup}^{\succ}_{\mathsf{sel}}$

The superposition calculus $\mathit{Sup}^{\succ}_{\mathsf{sel}}$ is parameterized by

- a selection function sel
- and a total and well-founded atom ordering $\succ$.

Superposition Calculus $Sup^{\succ}_{\mathrm{sel}}$

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:

A literal L is called [strictly] maximal in a clause C if and only if there exists a ground substitution σ such that $L\sigma$ is [strictly] maximal in $C\sigma$ (i.e., if for no other L' in C: $L\sigma \prec L'\sigma$ $[L\sigma \preceq L'\sigma]$).

Superposition Calculus $Sup_{\text{sel}}^{\succ}$

$$\frac{D \lor B \qquad C \lor \neg A}{(D \lor C)\sigma}$$ [Superposition Left with Selection]

if the following conditions are satisfied:

(i) $\sigma = \text{mgu}(A, B)$;

(ii) $B\sigma$ strictly maximal in $D\sigma \lor B\sigma$;

(iii) nothing is selected in $D \lor B$ by sel;

(iv) either $\neg A$ is selected, or else nothing is selected in $C \lor \neg A$ and $\neg A\sigma$ is maximal in $C\sigma \lor \neg A\sigma$.

Superposition Calculus $Sup^{\succ}_{\mathrm{sel}}$

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \qquad \text{[Factoring]}$$

if the following conditions are satisfied:

(i) $\sigma = \mathrm{mgu}(A, B)$;

(ii) $A\sigma$ is maximal in $C\sigma \vee A\sigma \vee B\sigma$;

(iii) nothing is selected in $C \vee A \vee B$ by sel.

Special Case: Propositional Logic

For ground clauses the superposition inference rule simplifies to

$$\frac{D \vee P \qquad C \vee \neg P}{D \vee C}$$

if the following conditions are satisfied:

(i) $P \succ D$;

(ii) nothing is selected in $D \vee P$ by sel;

(iii) $\neg P$ is selected in $C \vee \neg P$, or else nothing is selected in $C \vee \neg P$ and $\neg P \succeq \max(C)$.

Note: For positive literals, $P \succ D$ is the same as $P \succ \max(D)$.

Special Case: Propositional Logic

Analogously, the factoring rule simplifies to

$$\frac{C \vee P \vee P}{C \vee P}$$

if the following conditions are satisfied:

(i) P is the largest literal in $C \vee P \vee P$;

(ii) nothing is selected in $C \vee P \vee P$ by sel.

Search Spaces Become Smaller

1	$P \vee Q$	
2	$P \vee \boxed{\neg Q}$	
3	$\neg P \vee Q$	
4	$\neg P \vee \boxed{\neg Q}$	
5	$Q \vee Q$	Res 1, 3
6	Q	Fact 5
7	$\neg P$	Res 6, 4
8	P	Res 6, 2
9	$\bot$	Res 8, 7

we assume $P \succ Q$ and sel as indicated by $\boxed{X}$. The maximal literal in a clause is depicted in red.

With this ordering and selection function the refutation proceeds strictly deterministically in this example. Generally, proof search will still be non-deterministic but the search space will be much smaller than with unrestricted resolution.

Avoiding Rotation Redundancy

From

$$\dfrac{\dfrac{C_1 \lor P \qquad C_2 \lor \neg P \lor Q}{C_1 \lor C_2 \lor Q} \qquad C_3 \lor \neg Q}{C_1 \lor C_2 \lor C_3}$$

we can obtain by rotation

$$\dfrac{C_1 \lor P \qquad \dfrac{C_2 \lor \neg P \lor Q \qquad C_3 \lor \neg Q}{C_2 \lor \neg P \lor C_3}}{C_1 \lor C_2 \lor C_3}$$

another proof of the same clause. In large proofs many rotations are possible. However, if $P \succ Q$, then the second proof does not fulfill the orderings restrictions.

Avoiding Rotation Redundancy

Conclusion: In the presence of orderings restrictions (however one chooses $\succ$) no rotations are possible. In other words, orderings identify exactly one representant in any class of rotation-equivalent proofs.

Lifting Lemma for $Sup^{\succ}_{\mathrm{sel}}$

Lemma 3.35:

Let D and C be variable-disjoint clauses. If

$$\begin{tikzcd} D \arrow[d, "\sigma"] & C \arrow[d, "\rho"] \\ D\sigma & C\rho \end{tikzcd}$$

$$\frac{D\sigma \qquad C\rho}{C'}$$

[propositional inference in $Sup^{\succ}_{\mathrm{sel}}$]

and if $\mathrm{sel}(D\sigma) \simeq \mathrm{sel}(D)$, $\mathrm{sel}(C\rho) \simeq \mathrm{sel}(C)$ (that is, “corresponding” literals are selected), then there exists a substitution τ such that

$$\frac{D \qquad C}{C''}$$

[inference in $Sup^{\succ}_{\mathrm{sel}}$]

$$\begin{tikzcd} C'' \arrow[d, "\tau"] \\ C' = C''\tau \end{tikzcd}$$

Lifting Lemma for $\mathit{Sup}_{\mathrm{sel}}^{\succ}$

An analogous lifting lemma holds for factorization.

Saturation of General Clause Sets

Corollary 3.36:

Let N be a set of general clauses saturated under $\mathit{Sup}^{\succ}_{\mathrm{sel}}$, i. e., $\mathit{Sup}^{\succ}_{\mathrm{sel}}(N) \subseteq N$. Then there exists a selection function sel′ such that $\mathrm{sel}\,|_N = \mathrm{sel}'\,|_N$ and $G_\Sigma(N)$ is also saturated, i. e.,

$$\mathit{Sup}^{\succ}_{\mathrm{sel}'}(G_\Sigma(N)) \subseteq G_\Sigma(N).$$

Proof:

We first define the selection function sel′ such that $\mathrm{sel}'(C) = \mathrm{sel}(C)$ for all clauses $C \in G_\Sigma(N) \cap N$. For $C \in G_\Sigma(N) \setminus N$ we choose a fixed but arbitrary clause $D \in N$ with $C \in G_\Sigma(D)$ and define $\mathrm{sel}'(C)$ 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. 3.27 using the above lifting lemma. □

Soundness and Refutational Completeness

Theorem 3.37:

Let $\succ$ be an atom ordering and sel a selection function such that $\mathit{Sup}^{\succ}_{\mathrm{sel}}(N) \subseteq N$. Then

$$N \models \bot \Leftrightarrow \bot \in N$$

Proof:

The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part consider the propositional level: Construct a candidate interpretation $N_{\mathcal{I}}$ as for superposition without selection, except that clauses C in N that have selected literals are not productive, even when they are false in N_C and when their maximal atom occurs only once and positively. The result then follows by Corollary 3.36. □

Craig-Interpolation

A theoretical application of superposition is Craig-Interpolation:

Theorem 3.38 (Craig 1957):
Let ϕ and ψ be two propositional formulas such that $\phi \models \psi$.
Then there exists a formula χ (called the interpolant for $\phi \models \psi$),
such that χ contains only prop. variables occurring both in ϕ
and in ψ, and such that $\phi \models \chi$ and $\chi \models \psi$.

Craig-Interpolation

Proof:

Translate ϕ and $\neg\psi$ into CNF. let N and M, resp., denote the resulting clause set. Choose an atom ordering $\succ$ for which the prop. variables that occur in ϕ but not in ψ are maximal. Saturate N into N^* w. r. t. $\mathit{Sup}_{\text{sel}}^{\succ}$ with an empty selection function sel . Then saturate $N^* \cup M$ w. r. t. $\mathit{Sup}_{\text{sel}}^{\succ}$ 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 that also occur in ψ. The conjunction of these premises is an interpolant χ. The theorem also holds for first-order formulas. For universal formulas the above proof can be easily extended. In the general case, a proof based on superposition technology is more complicated because of Skolemization. $\square$

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? (Conjecture: e. g., if they are tautologies or if they are subsumed by other clauses.)

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

Recall: 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. $G_\Sigma(N)$.

Note: The same ordering $\prec$ is used for ordering restrictions and for redundancy (and for the completeness proof).

Examples of Redundancy

Proposition 3.39:

Recall the redundancy criteria:

- C tautology (i. e., $\models C$) $\;\Rightarrow\;$ C redundant w. r. t. any set N.
  Tautology Deletion
- $C\sigma \subset D \;\Rightarrow\; D$ redundant w. r. t. $N \cup \{C\}$.
  Subsumption
- $C\sigma \subseteq D \;\Rightarrow\; D \vee \overline{L}\sigma$ redundant w. r. t. $N \cup \{C \vee L,\ D\}$.
  Subsumption Resolution

Saturation up to Redundancy

N is called saturated up to redundancy (w. r. t. $\mathit{Sup}^{\succ}_{\mathrm{sel}}$)

$$:\Leftrightarrow \mathit{Sup}^{\succ}_{\mathrm{sel}}(N \setminus \mathit{Red}(N)) \subseteq N \cup \mathit{Red}(N)$$

Theorem 3.40:

Let N be saturated up to redundancy. Then

$$N \models \bot \Leftrightarrow \bot \in N$$

Saturation up to Redundancy

Proof (Sketch):

(i) Ground case:

- consider the construction of the candidate interpretation $N_{\mathcal{I}}^{\succ}$ for $\mathit{Sup}_{\mathrm{sel}}^{\succ}$
- redundant clauses are not productive
- redundant clauses in N are not minimal counterexamples for $N_{\mathcal{I}}^{\succ}$

The premises of “essential” inferences are either minimal counterexamples or productive.

(ii) Lifting: no additional problems over the proof of Theorem 3.37. □

Monotonicity Properties of Redundancy

Theorem 3.41:

(i) $N \subseteq M \Rightarrow \mathit{Red}(N) \subseteq \mathit{Red}(M)$

(ii) $M \subseteq \mathit{Red}(N) \Rightarrow \mathit{Red}(N) \subseteq \mathit{Red}(N \setminus M)$

We conclude that redundancy is preserved when, during a theorem proving process, one adds (derives) new clauses or deletes redundant clauses. Recall that $\mathit{Red}(N)$ may include clauses that are not in N.

A First-Order Superposition Theorem Prover

Straightfotward extension of the propositional *STP* prover.

3 clause sets:

N(ew) containing new inferred clauses

U(sable) containing reduced new inferred clauses

clauses get into *W(orked) O(ff)* once their inferences have been computed

Strategy:

Inferences will only be computed when there are no possibilities for simplification

Rewrite Rules for *FSTP*

Tautology Deletion

$$(N \uplus \{C\}; U; WO) \quad \Rightarrow_{FSTP} \quad (N; U; WO)$$

if C is a tautology

Forward Subsumption

$$(N \uplus \{C\}; U; WO) \quad \Rightarrow_{FSTP} \quad (N; U; WO)$$

if some $D \in (U \cup WO)$ subsumes C, $D\sigma \subseteq C$

Backward Subsumption U

$$(N \uplus \{C\}; U \uplus \{D\}; WO) \quad \Rightarrow_{FSTP} \quad (N \cup \{C\}; U; WO)$$

if C strictly subsumes D ($C\sigma \subset D$)

Rewrite Rules for *FSTP*

Backward Subsumption *WO*

$$(N \uplus \{C\}; U; WO \uplus \{D\}) \quad \Rightarrow_{FSTP} \quad (N \cup \{C\}; U; WO)$$

if C strictly subsumes D ($C\sigma \subset D$)

Forward Subsumption Resolution

$$(N \uplus \{C_1 \vee L\}; U; WO) \quad \Rightarrow_{FSTP} \quad (N \cup \{C_1\}; U; WO)$$

if $C_2 \vee L' \in (U \cup WO)$ such that $C_2\sigma \subseteq C_1$ and $L'\sigma = \overline{L}$

Backward Subsumption Resolution *U*

$$(N \uplus \{C_1 \vee L'\}; U \uplus \{C_2 \vee L\}; WO) \quad \Rightarrow_{FSTP} \quad (N \cup \{C_1 \vee L\}; U \uplus \{C_2\}; WO)$$

if $C_1\sigma \subseteq C_2$ and $L'\sigma = \overline{L}$

Rewrite Rules for *FSTP*

Backward Subsumption Resolution *WO*

$(N \uplus \{C_1 \vee L'\}; U; WO \uplus \{C_2 \vee L\}) \quad \Rightarrow_{FSTP} \quad (N \cup \{C_1 \vee L\}; U; WO \uplus \{C_2\})$

if $C_1\sigma \subseteq C_2$ and $L'\sigma = \overline{L}$

Clause Processing

$(N \uplus \{C\}; U; WO) \quad \Rightarrow_{FSTP} \quad (N; U \cup \{C\}; WO)$

Inference Computation

$(\emptyset; U \uplus \{C\}; WO) \quad \Rightarrow_{FSTP} \quad (N; U; WO \cup \{C\})$

where N is the set of clauses derived by first-order superposition inferences from C and clauses in WO.

Implementation

Although first-order and propositional subsumption just differ in the matcher σ, propositional subsumption between two clauses C and D can be decided in $O(n)$, $n = |C| + |D|$ whereas first-order subsumption is NP-complete.

Hyperresolution

There are *many* variants of resolution. (We refer to [Bachmair, Ganzinger: Resolution Theorem Proving] for further reading.)

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

Hyperresolution replaces these successive steps by a single inference. As for $\mathit{Sup}^{\succ}_{\mathrm{sel}}$, the calculus is parameterized by an atom ordering $\succ$ and a selection function sel.

Hyperresolution

$$\frac{D_1 \vee B_1 \quad \ldots \quad D_n \vee B_n \qquad C \vee \neg A_1 \vee \ldots \vee \neg A_n}{(D_1 \vee \ldots \vee D_n \vee C)\sigma}$$

with $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$, 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 sel, or else nothing is selected in the right premise and $n = 1$ and $\neg A_1\sigma$ is maximal in $C\sigma$.

Similarly to superposition (resolution), hyperresolution has to be complemented by a factorization inference.

Hyperresolution

As we have seen, hyperresolution can be simulated by iterated binary superposition.

However this yields intermediate clauses which HR might not derive, and many of them might not be extendable into a full HR inference.

3.12 Summary: Superposition Theorem Proving

- Superposition 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 $C \lor A$, $A \succ C$; inferences with those reduce counterexamples.

Summary: Superposition Theorem Proving

- *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" 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) or arithmetic
  $\Rightarrow$ further specialization of inference systems required.

Other Inference Systems

- Tableaux
- Instantiation-based methods

  Resolution-based instance generation

  Disconnection calculus

  . . .
- Natural deduction
- Sequent calculus/Gentzen calculus
- Hilbert calculus

Other Inference Systems

One major problem with all those calculi concerning automation is that they contain a rule either guessing instances or limiting the use of formulas. So the procedure has to guess instances and/or the number of copies of formulas. For example rules like:

Universal Quantification

$$S \cup \{\forall x\; \phi\} \quad \Rightarrow \quad S \cup \{\forall x\; \phi\} \cup \phi\{x \mapsto t\}$$

for some ground term $t \in \mathrm{T}_{\Sigma}$

Existential Quantification

$$S \cup \{\exists x\; \phi\} \quad \Rightarrow \quad S \cup \{\exists x\; \phi\} \cup \phi\{x \mapsto a\}$$

for some constant a new to ϕ

Part 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 ϕ be a closed first-order formula with equality. Let $\sim\, \notin \Pi$ be a new predicate symbol. The set $Eq(\Sigma)$ contains the formulas

$$\begin{gathered}
\forall x\,(x \sim x)\\
\forall x, y\,(x \sim y \rightarrow y \sim x)\\
\forall x, y, z\,(x \sim y \wedge y \sim z \rightarrow x \sim z)\\
\forall \vec{x}, \vec{y}\,(x_1 \sim y_1 \wedge \cdots \wedge x_n \sim y_n \rightarrow f(x_1, \dots, x_n) \sim f(y_1, \dots, y_n))\\
\forall \vec{x}, \vec{y}\,(x_1 \sim y_1 \wedge \cdots \wedge x_m \sim y_m \wedge P(x_1, \dots, x_m) \rightarrow P(y_1, \dots, y_m))
\end{gathered}$$

for every $f \in \Omega$ and $P \in \Pi$. Let $\tilde{\phi}$ be the formula that one obtains from ϕ if every occurrence of $\approx$ is replaced by $\sim$. Then ϕ is satisfiable if and only if $Eq(\Sigma) \cup \{\tilde{\phi}\}$ is satisfiable.

Handling Equality Naively

By giving the equality axioms explicitly, first-order problems with equality can in principle be solved by *SFTP*.

But this is unfortunately not efficient, mainly due to the transitivity axiom.

Handling Equality Naively

Equality is theoretically difficult: First-order functional programming is Turing-complete.

But: *SFTP* 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:

Term rewrite systems

Expressing semantic consequence syntactically

Knuth-Bendix-Completion

Entailment for equations

(Superposition for first-order clauses with equality)

4.2 Term Rewrite Systems

Let E be a set of (implicitly universally quantified) equations.

The rewrite relation $\rightarrow_E \subseteq \mathrm{T}_\Sigma(X) \times \mathrm{T}_\Sigma(X)$ is defined by

$$
\begin{aligned}
s \rightarrow_E t \quad \text{iff} \quad & \text{there exist } (l \approx r) \in E,\ p \in \mathrm{pos}(s), \\
& \text{and } \sigma : X \rightarrow \mathrm{T}_\Sigma(X), \\
& \text{such that } s|_p = l\sigma \text{ and } t = s[r\sigma]_p.
\end{aligned}
$$

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.

Term Rewrite Systems

An equation $l \approx r$ is also called a rewrite rule, if l is not a variable and $\mathrm{vars}(l) \supseteq \mathrm{vars}(r)$.

Notation: $l \to r$.

A set of rewrite rules is called a term rewrite system (TRS).

Term Rewrite Systems

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 (abstrac) rewrite systems).

Note: If E is terminating, then it is a TRS.

Rewrite Relations

Corollary 4.2:
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 4.3:
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.

Rewrite Relations

Problems:

Show local confluence of E.

Show termination of E.

Transform E into an equivalent set of equations that is locally confluent and terminating.

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:

$s \approx_E t$ if and only if $s \leftrightarrow_E^* t$.

E-Algebras

Let E be a set of equations over $\mathrm{T}_\Sigma(X)$. The following inference system allows to derive consequences of E:

E-Algebras

$$\mathcal{I}\frac{}{t \approx t} \qquad \text{(Reflexivity)}$$

$$\mathcal{I}\frac{t \approx t'}{t' \approx t} \qquad \text{(Symmetry)}$$

$$\mathcal{I}\frac{t \approx t' \qquad t' \approx t''}{t \approx t''} \qquad \text{(Transitivity)}$$

$$\mathcal{I}\frac{t_1 \approx t_1' \quad \dots \quad t_n \approx t_n'}{f(t_1, \dots, t_n) \approx f(t_1', \dots, t_n')} \quad \text{for any } f/n \qquad \text{(Congruence)}$$

$$\mathcal{I}\frac{t \approx t'}{t\sigma \approx t'\sigma} \quad \text{for any substitution } \sigma \qquad \text{(Instance)}$$

E-Algebras

Lemma 4.4:

The following properties are equivalent:

(i) $s \leftrightarrow_E^* t$

(ii) $E \Rightarrow^* s \approx t$ is derivable.

where $E \Rightarrow^* s \approx t$ is an abbreviation for $E \Rightarrow^* E'$ and $s \approx t \in E'$.

Recall that the before inference rules of the form $\mathcal{I}\,\dfrac{A_1 \;\ldots\; A_k}{B}$

are abbreviations for rewrite rules $E \uplus \{A_1, \ldots, A_k\} \Rightarrow E \cup \{A_1, \ldots A_k, B\}$.

E-Algebras

Constructing a quotient algebra:

Let X be a set of variables.

For $t \in \mathrm{T}_\Sigma(X)$ let $[t] = \{\, t' \in \mathrm{T}_\Sigma(X) \mid E \Rightarrow^* t \approx t' \,\}$ be the congruence class of t.

Define a Σ-algebra $\mathrm{T}_\Sigma(X)/E$ (abbreviated by $\mathcal{T}$) as follows:

$U_{\mathcal{T}} = \{\, [t] \mid t \in \mathrm{T}_\Sigma(X) \,\}$.

$f_{\mathcal{T}}([t_1], \dots, [t_n]) = [f(t_1, \dots, t_n)]$ for $f \in \Omega$.

E-Algebras

Lemma 4.5:
$f_{\mathcal{T}}$ is well-defined: If $[t_i] = [t'_i]$, then $[f(t_1, \dots, t_n)] = [f(t'_1, \dots, t'_n)]$.

Lemma 4.6:
$\mathcal{T} = \mathrm{T}_\Sigma(X)/E$ is an E-algebra.

Lemma 4.7:
Let X be a countably infinite set of variables; let $s, t \in \mathrm{T}_\Sigma(X)$.
If $\mathrm{T}_\Sigma(X)/E \models \forall \vec{x}(s \approx t)$, then $E \Rightarrow^* s \approx t$ is derivable.

E-Algebras

Theorem 4.8 (“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 $s, t \in \mathrm{T}_\Sigma(X)$:

(i) $s \leftrightarrow_E^* t$.

(ii) $E \Rightarrow^* s \approx t$ is derivable.

(iii) $s \approx_E t$, i. e., $E \models \forall \vec{x}(s \approx t)$.

(iv) $\mathrm{T}_\Sigma(X)/E \models \forall \vec{x}(s \approx t)$.

Universal Algebra

$\mathsf{T}_\Sigma(X)/E = \mathsf{T}_\Sigma(X)/{\approx_E} = \mathsf{T}_\Sigma(X)/{\leftrightarrow_E^*}$ 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 $\hat{\varphi} : \mathsf{T}_\Sigma(X)/E \to \mathcal{B}$.

$\mathsf{T}_\Sigma(\emptyset)/E = \mathsf{T}_\Sigma(\emptyset)/{\approx_E} = \mathsf{T}_\Sigma(\emptyset)/{\leftrightarrow_E^*}$ is called the initial E-algebra.

Universal Algebra

$\approx_E = \{\, (s,t) \mid E \models s \approx t \,\}$ is called the equational theory of E.

$\approx_E^I = \{\, (s,t) \mid \mathrm{T}_\Sigma(\emptyset)/E \models s \approx t \,\}$ 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 $x + y \approx_E^I y + x$, but $x + y \not\approx_E y + x$.

4.3 Critical Pairs

Showing local confluence (Sketch):

Problem: If $t_1 \;{}_E\!\leftarrow\; t_0 \rightarrow_E t_2$, does there exist a term s such that $t_1 \rightarrow_E^* s \;{}_E^*\!\leftarrow\; t_2$?

If the two rewrite steps happen in different subtrees (disjoint redexes): 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.

Critical Pairs

Showing local confluence (Sketch):

Question:
Are there rewrite rules $l_1 \to r_1$ and $l_2 \to r_2$ 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.o.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 σ of $l_1|_p$ and l_2.

Critical Pairs

Let $l_i \to r_i$ $(i = 1, 2)$ be two rewrite rules in a TRS R whose variables have been renamed such that $\text{vars}(l_1) \cap \text{vars}(l_2) = \emptyset$. (Remember that $\text{vars}(l_i) \supseteq \text{vars}(r_i)$.)

Let $p \in \text{pos}(l_1)$ be a position such that $l_1|_p$ is not a variable and σ is an mgu of $l_1|_p$ and l_2.

Then $r_1\sigma \leftarrow l_1\sigma \to (l_1\sigma)[r_2\sigma]_p$.

$\langle r_1\sigma, (l_1\sigma)[r_2\sigma]_p\rangle$ is called a critical pair of R.

The critical pair is joinable (or: converges), if $r_1\sigma \downarrow_R (l_1\sigma)[r_2\sigma]_p$.

Critical Pairs

Theorem 4.9 ("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.

Critical Pairs

“if”: Suppose s rewrites to t_1 and t_2 using rewrite rules $l_i \rightarrow r_i \in R$ at positions $p_i \in \mathrm{pos}(s)$, 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 ($p_1 \mid\mid p_2$), or one is a prefix of the other (w.o.l.o.g., $p_1 \leq p_2$).

Critical Pairs

Case 1: $p_1 \mid\mid p_2$.

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 $l_1 \rightarrow r_1$.

Critical Pairs

Case 2: $p_1 \leq p_2$.

Case 2.1: $p_2 = p_1\, q_1\, q_2$, 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 $p_1\, q'\, q_2$, 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 $p_1\, q\, q_2$, 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 θ', where $\theta' = \theta[x \mapsto (x\theta)[r_2\theta]_{q_2}]$.

Critical Pairs

Case 2.2: $p_2 = p_1\, p$, 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 θ is a unifier of l_2 and $l_1|_p$.

Let σ 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 $r_1\sigma \rightarrow_R^* v \leftarrow_R^* (l_1\sigma)[r_2\sigma]_p$.

Consequently, $t_1 = s[r_1\theta]_{p_1} = s[r_1\sigma\tau]_{p_1} \rightarrow_R^* s[v\tau]_{p_1}$ and $t_2 = s[r_2\theta]_{p_2} = s[(l_1\theta)[r_2\theta]_p]_{p_1} = s[(l_1\sigma\tau)[r_2\sigma\tau]_p]_{p_1} = s[((l_1\sigma)[r_2\sigma]_p)\tau]_{p_1} \rightarrow_R^* s[v\tau]_{p_1}$.

This completes the proof of the Critical Pair Theorem. □

Critical Pairs

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., $p = \varepsilon$).

Critical Pairs

Corollary 4.10:
A terminating TRS R is confluent if and only if all its critical pairs are joinable.

Corollary 4.11:
For a finite terminating TRS, confluence is decidable.

4.4 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?

Termination

Proposition 4.12:

Both termination problems for TRSs are undecidable in general.

Consequence:

Decidable criteria for termination are not complete.

Reduction Orderings

Goal:

Given a finite TRS R, show termination of R by looking at finitely many rules $l \to r \in R$, rather than at infinitely many possible replacement steps $s \to_R s'$.

Reduction Orderings

A binary relation $\sqsupset$ over $\mathrm{T}_\Sigma(X)$ is called compatible with Σ-operations, if $s \sqsupset s'$ implies $f(t_1, \dots, s, \dots, t_n) \sqsupset f(t_1, \dots, s', \dots, t_n)$ for all $f \in \Omega$ and $s, s', t_i \in \mathrm{T}_\Sigma(X)$.

Lemma 4.13:
The relation $\sqsupset$ is compatible with Σ-operations, if and only if $s \sqsupset s'$ implies $t[s]_p \sqsupset t[s']_p$ for all $s, s', t \in \mathrm{T}_\Sigma(X)$ and $p \in \mathrm{pos}(t)$.

Note: compatible with Σ-operations = compatible with contexts.

Reduction Orderings

A binary relation $\sqsupset$ over $\mathsf{T}_\Sigma(X)$ is called stable under substitutions, if $s \sqsupset s'$ implies $s\sigma \sqsupset s'\sigma$ for all $s, s' \in \mathsf{T}_\Sigma(X)$ and substitutions σ.

Reduction Orderings

A binary relation $\sqsupset$ is called a rewrite relation, if it is compatible with Σ-operations and stable under substitutions.

Example: If R is a TRS, then $\rightarrow_R$ is a rewrite relation.

A strict partial ordering over $\mathrm{T}_\Sigma(X)$ that is a rewrite relation is called rewrite ordering.

A well-founded rewrite ordering is called reduction ordering.

Reduction Orderings

Theorem 4.14:
A TRS R terminates if and only if there exists a reduction ordering $\succ$ such that $l \succ r$ for every rule $l \to r \in R$.

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.

The Interpretation Method

Proving termination by interpretation:

Let $\mathcal{A}$ be a Σ-algebra; let $\succ$ be a well-founded strict partial ordering on its universe.

Define the ordering $\succ_{\mathcal{A}}$ over $\mathrm{T}_{\Sigma}(X)$ by $s \succ_{\mathcal{A}} t$ iff $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(t)$ for all assignments $\beta : X \to U_{\mathcal{A}}$.

Is $\succ_{\mathcal{A}}$ a reduction ordering?

The Interpretation Method

Lemma 4.15:

$\succ_{\mathcal{A}}$ is stable under substitutions.

The Interpretation Method

A function $f : U_{\mathcal{A}}^n \to U_{\mathcal{A}}$ is called monotone (w. r. t. $\succ$), if $a \succ a'$ implies $f(b_1, \dots, a, \dots, b_n) \succ f(b_1, \dots, a', \dots, b_n)$ for all $a, a', b_i \in U_{\mathcal{A}}$.

Lemma 4.16:
If the interpretation $f_{\mathcal{A}}$ of every function symbol f is monotone w. r. t. $\succ$, then $\succ_{\mathcal{A}}$ is compatible with Σ-operations.

Theorem 4.17:
If the interpretation $f_{\mathcal{A}}$ of every function symbol f is monotone w. r. t. $\succ$, then $\succ_{\mathcal{A}}$ is a reduction ordering.

Polynomial Orderings

Polynomial orderings:

Instance of the interpretation method:

The carrier set $U_{\mathcal{A}}$ is $\mathbb{N}$ or some subset of $\mathbb{N}$.

To every function symbol f with arity n we associate a polynomial $P_f(X_1, \dots, X_n) \in \mathbb{N}[X_1, \dots, X_n]$ with coefficients in $\mathbb{N}$ and indeterminates $X_1, \dots, X_n$. Then we define $f_{\mathcal{A}}(a_1, \dots, a_n) = P_f(a_1, \dots, a_n)$ for $a_i \in U_{\mathcal{A}}$.

Polynomial Orderings

Requirement 1:

If $a_1, \ldots, a_n \in U_{\mathcal{A}}$, then $f_{\mathcal{A}}(a_1, \ldots, a_n) \in U_{\mathcal{A}}$. (Otherwise, $\mathcal{A}$ would not be a Σ-algebra.)

Polynomial Orderings

Requirement 2:

$f_{\mathcal{A}}$ must be monotone (w. r. t. $\succ$).

From now on:

$U_{\mathcal{A}} = \{ n \in \mathbb{N} \mid n \geq 1 \}$.

If arity$(f) = 0$, then P_f is a constant ≥ 1.

If arity$(f) = n \geq 1$, then P_f is a polynomial $P(X_1, \dots, X_n)$, such that every X_i occurs in some monomial with exponent at least 1 and non-zero coefficient.

$\Rightarrow$ Requirements 1 and 2 are satisfied.

Polynomial Orderings

The mapping from function symbols to polynomials can be extended to terms: A term t containing the variables $x_1, \dots, x_n$ yields a polynomial P_t with indeterminates $X_1, \dots, X_n$ (where X_i corresponds to $\beta(x_i)$).

Example:

$$\Omega = \{b/0,\ f/1,\ g/3\}$$

$$P_b = 3, \quad P_f(X_1) = X_1^2, \quad P_g(X_1, X_2, X_3) = X_1 + X_2X_3.$$

Let $t = g(f(b), f(x), y)$, then $P_t(X, Y) = 9 + X^2Y$.

Polynomial Orderings

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 $a_1, \ldots, a_n \in U_{\mathcal{A}}$.

Clearly, $l \succ_{\mathcal{A}} r$ iff $P_l > P_r$ iff $P_l - P_r > 0$.

Question: Can we check $P_l - P_r > 0$ automatically?

Polynomial Orderings

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 4.18:
Hilbert's 10th Problem is undecidable.

Proposition 4.19:
Given a polynomial interpretation and two terms l, r, it is undecidable whether $P_l > P_r$.

Proof:
By reduction of Hilbert's 10th Problem. □

Polynomial Orderings

One easy case:

If we restrict to linear polynomials, deciding whether $P_l - P_r > 0$ is trivial:

$\sum k_i a_i + k > 0$ for all $a_1, \ldots, a_n \geq 1$ if and only if

$k_i \geq 0$ for all $i \in \{1, \ldots, n\}$,

and $\sum k_i + k > 0$

Polynomial Orderings

Another possible solution:

Test whether $P_l(a_1, \ldots, a_n) > P_r(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in \{\, x \in \mathbb{R} \mid x \geq 1 \,\}$.

This is decidable (but hard). Since $U_{\mathcal{A}} \subseteq \{\, x \in \mathbb{R} \mid x \geq 1 \,\}$, it implies $P_l > P_r$.

Alternatively:

Use fast overapproximations.

Simplification Orderings

The proper subterm ordering $\rhd$ is defined by $s \rhd t$ if and only if $s|_p = t$ for some position $p \neq \varepsilon$ of s.

Simplification Orderings

A rewrite ordering $\succ$ over $\mathrm{T}_\Sigma(X)$ is called simplification ordering, if it has the subterm property: $s \rhd t$ implies $s \succ t$ for all $s, t \in \mathrm{T}_\Sigma(X)$.

Example:

Let R_{emb} be the rewrite system $R_{\text{emb}} = \{\, f(x_1, \dots, x_n) \to x_i \mid f \in \Omega,\ 1 \le i \le n = \text{arity}(f) \,\}$.

Define $\rhd_{\text{emb}} = \to^{+}_{R_{\text{emb}}}$ and $\unrhd_{\text{emb}} = \to^{*}_{R_{\text{emb}}}$ ("homeomorphic embedding relation").

$\rhd_{\text{emb}}$ is a simplification ordering.

Simplification Orderings

Lemma 4.20:

If $\succ$ is a simplification ordering, then $s \rhd_{\text{emb}} t$ implies $s \succ t$ and $s \trianglerighteq_{\text{emb}} t$ implies $s \succeq t$.

Simplification Orderings

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.

Simplification Orderings

Theorem 4.21 ("Kruskal's Theorem"):
Let Σ 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 $t_j \trianglerighteq_{\text{emb}} t_i$. ($\trianglerighteq_{\text{emb}}$ is called a well-partial-ordering (wpo).)

Proof:
See Baader and Nipkow, page 113–115. □

Simplification Orderings

Theorem 4.22 (Dershowitz):

If Σ is a finite signature, then every simplification ordering $\succ$ on $\mathrm{T}_\Sigma(X)$ is well-founded (and therefore a reduction ordering).

Simplification Orderings

There are reduction orderings that are not simplification orderings and terminating TRSs that are not contained in any simplification ordering.

Example:

Let $R = \{f(f(x)) \to f(g(f(x)))\}$.

R terminates and $\to_R^+$ is therefore a reduction ordering.

Assume that $\to_R$ were contained in a simplification ordering $\succ$. Then $f(f(x)) \to_R f(g(f(x)))$ implies $f(f(x)) \succ f(g(f(x)))$, and $f(g(f(x))) \trianglerighteq_{\text{emb}} f(f(x))$ 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 Ω.

The lexicographic path ordering $\succ_{\mathrm{lpo}}$ on $\mathrm{T}_\Sigma(X)$ induced by $\succ$ is defined by: $s \succ_{\mathrm{lpo}} t$ iff

(1) $t \in \mathrm{vars}(s)$ and $t \neq s$, or

(2) $s = f(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$, and

- (a) $s_i \succeq_{\mathrm{lpo}} t$ for some i, or
- (b) $f \succ g$ and $s \succ_{\mathrm{lpo}} t_j$ for all j, or
- (c) $f = g$, $s \succ_{\mathrm{lpo}} t_j$ for all j, and $(s_1, \dots, s_m)\ (\succ_{\mathrm{lpo}})_{\mathrm{lex}}\ (t_1, \dots, t_n)$.

Path Orderings

Lemma 4.23:
$s \succ_{\mathrm{lpo}} t$ implies $\mathrm{vars}(s) \supseteq \mathrm{vars}(t)$.

Theorem 4.24:
$\succ_{\mathrm{lpo}}$ is a simplification ordering on $\mathrm{T}_\Sigma(X)$.

Theorem 4.25:
If the precedence $\succ$ is total, then the lexicographic path ordering $\succ_{\mathrm{lpo}}$ is total on ground terms, i. e., for all $s, t \in \mathrm{T}_\Sigma(\emptyset)$: $s \succ_{\mathrm{lpo}} t \vee t \succ_{\mathrm{lpo}} s \vee s = t$.

Path Orderings

Recapitulation:

Let $\Sigma = (\Omega, \Pi)$ be a finite signature, let $\succ$ be a strict partial ordering ("precedence") on Ω. The lexicographic path ordering $\succ_{\mathrm{lpo}}$ on $\mathrm{T}_\Sigma(X)$ induced by $\succ$ is defined by: $s \succ_{\mathrm{lpo}} t$ iff

(1) $t \in \mathrm{vars}(s)$ and $t \neq s$, or

(2) $s = f(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$, and

- (a) $s_i \succeq_{\mathrm{lpo}} t$ for some i, or
- (b) $f \succ g$ and $s \succ_{\mathrm{lpo}} t_j$ for all j, or
- (c) $f = g$, $s \succ_{\mathrm{lpo}} t_j$ for all j, and $(s_1, \dots, s_m)$ $(\succ_{\mathrm{lpo}})_{\mathrm{lex}}$ $(t_1, \dots, t_n)$.

Path Orderings

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 π)
- compare multiset of subterms using the multiset extension ("multiset path ordering (mpo)", Dershowitz)
- to each function symbol f with arity$(n) \geq 1$ associate a status $\in \{\mathit{mul}\} \cup \{\mathit{lex}_\pi \mid \pi : \{1, \dots, n\} \to \{1, \dots, n\}\}$ 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 Ω, let $w : \Omega \cup X \to \mathbb{R}_0^+$ be a weight function, such that the following admissibility conditions are satisfied:

$w(x) = w_0 \in \mathbb{R}^+$ for all variables $x \in X$; $w(c) \geq w_0$ for all constants $c \in \Omega$.

If $w(f) = 0$ for some $f \in \Omega$ with $\text{arity}(f) = 1$, then $f \succeq g$ for all $g \in \Omega$.

The weight function w can be extended to terms as follows:

$$w(t) = \sum_{x \in \text{vars}(t)} w(x) \cdot \#(x, t) + \sum_{f \in \Omega} w(f) \cdot \#(f, t).$$

The Knuth-Bendix Ordering

The Knuth-Bendix ordering $\succ_{\text{kbo}}$ on $\mathrm{T}_\Sigma(X)$ 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) $s = f(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$, and $f \succ g$, or
- (c) $s = f(s_1, \dots, s_m)$, $t = f(t_1, \dots, t_m)$, and $(s_1, \dots, s_m)\ (\succ_{\text{kbo}})_{\text{lex}}\ (t_1, \dots, t_m)$.

The Knuth-Bendix Ordering

Theorem 4.26:

The Knuth-Bendix ordering induced by $\succ$ and w is a simplification ordering on $\mathrm{T}_\Sigma(X)$.

Proof:

Baader and Nipkow, pages 125–129. □

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.

Defining a weight $w(f) = 0$ for some unary function symbol f was in particular introduced for the application of KBO to equational systems defining groups.

4.5 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.)

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.

Knuth-Bendix Completion: Inference Rules

The completion procedure is itself presented as a set of rewrite rules working on a pair of equations E and rules R: $(E_0; R_0) \Rightarrow (E_1; R_1) \Rightarrow (E_2; R_2) \Rightarrow \ldots$

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 $(E; R) \Rightarrow (E'; R')$, the equational theories of $E \cup R$ and $E' \cup R'$ agree: $\approx_{E \cup R} = \approx_{E' \cup R'}$.

Knuth-Bendix Completion: Inference Rules

Notations:

The formula $s \mathrel{\dot{\approx}} t$ denotes either $s \approx t$ or $t \approx s$.

$\mathrm{CP}(R)$ denotes the set of all critical pairs between rules in R.

Knuth-Bendix Completion: Inference Rules

Orient

$$(E \uplus \{s \mathrel{\dot{\approx}} t\}; R) \quad \Rightarrow_{KBC} \quad (E; R \cup \{s \to t\})$$

if $s \succ t$

Note: There are equations $s \approx t$ that cannot be oriented, i. e., neither $s \succ t$ nor $t \succ s$.

Knuth-Bendix Completion: Inference Rules

Trivial equations cannot be oriented – but we don't need them anyway:

Delete

$$(E \uplus \{s \approx s\}; R) \quad \Rightarrow_{KBC} \quad (E; R)$$

Knuth-Bendix Completion: Inference Rules

Critical pairs between rules in R are turned into additional equations:

Deduce

$$(E; R) \quad \Rightarrow_{KBC} \quad (E \cup \{s \approx t\}; R)$$

if $\langle s, t \rangle \in \mathrm{CP}(R)$

Note: If $\langle s, t \rangle \in \mathrm{CP}(R)$ then $s \;{}_R{\leftarrow}\; u \rightarrow_R t$ and hence $R \models s \approx t$.

Knuth-Bendix Completion: Inference Rules

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

$$(E \uplus \{s \mathrel{\dot{\approx}} t\}; R) \quad \Rightarrow_{KBC} \quad (E \cup \{u \approx t\}; R)$$

if $s \to_R u$

Knuth-Bendix Completion: Inference Rules

Simplification of the right-hand side of a rule is unproblematic.

R-Simplify-Rule

$$(E; R \uplus \{s \to t\}) \quad \Rightarrow_{KBC} \quad (E; R \cup \{s \to u\})$$

if $t \to_R u$

Simplification of the left-hand side may influence orientability and orientation. Therefore, it yields an *equation*:

L-Simplify-Rule

$$(E; R \uplus \{s \to t\}) \quad \Rightarrow_{KBC} \quad (E \cup \{u \approx t\}; R$$

if $s \to_R u$ using a rule $l \to r \in R$ such that $s \sqsupset l$ (see next slide).

Knuth-Bendix Completion: Inference Rules

For technical reasons, the lhs of $s \to t$ may only be simplified using a rule $l \to r$, if $l \to r$ *cannot* be simplified using $s \to t$, that is, if $s \sqsupset l$, where the encompassment quasi-ordering $\sqsupseteq\!\!\!\!\raisebox{-1ex}{$\sim$}$ is defined by

$$s \underset{\sim}{\sqsupset} l \quad \text{if} \quad s|_p = l\sigma \text{ for some } p \text{ and } \sigma$$

and $\sqsupset \;=\; \underset{\sim}{\sqsupset} \setminus \underset{\sim}{\sqsubset}$ is the strict part of $\underset{\sim}{\sqsupset}$.

Lemma 4.27:

$\sqsupset$ is a well-founded strict partial ordering.

Knuth-Bendix Completion: Inference Rules

Lemma 4.28:

If $(E; R) \Rightarrow_{KBC} (E'; R')$, then $\approx_{E \cup R} = \approx_{E' \cup R'}$.

Lemma 4.29:

If $(E; R) \Rightarrow_{KBC} (E'; R')$ and $\rightarrow_R \subseteq \succ$, then $\rightarrow_{R'} \subseteq \succ$.

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.

Knuth-Bendix Completion: Correctness Proof

A (finite or infinite sequence) $(E_0; R_0) \Rightarrow_{KBC} (E_1; R_1) \Rightarrow_{KBC} (E_2; R_2) \Rightarrow_{KBC} \ldots$ 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 $R_* = R_n$.

Knuth-Bendix Completion: Correctness Proof

A run is called fair, if $CP(R_*) \subseteq E_\infty$ (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 $\approx_{E_0} = \approx_{E_\infty \cup R_\infty} = \leftrightarrow^*_{E_\infty \cup R_\infty} = {\downarrow_{R_*}}$.

Knuth-Bendix Completion: Correctness Proof

General assumptions from now on:

$(E_0; R_0) \Rightarrow_{KBC} (E_1; R_1) \Rightarrow_{KBC} (E_2; R_2) \Rightarrow_{KBC} \ldots$
is a fair run.

R_0 and E_* are empty.

Knuth-Bendix Completion: Correctness Proof

A proof of $s \approx t$ in $E_\infty \cup R_\infty$ is a finite sequence $(s_0, \dots, s_n)$ such that $s = s_0$, $t = s_n$, and for all $i \in \{1, \dots, n\}$:

(1) $s_{i-1} \leftrightarrow_{E_\infty} s_i$, or

(2) $s_{i-1} \rightarrow_{R_\infty} s_i$, or

(3) $s_{i-1} \;{}_{R_\infty}\!\!\leftarrow s_i$.

The pairs (s_{i-1}, s_i) are called proof steps.

A proof is called a rewrite proof in R_*, if there is a $k \in \{0, \dots, n\}$ such that $s_{i-1} \rightarrow_{R_*} s_i$ for $1 \leq i \leq k$ and $s_{i-1} \;{}_{R_*}\!\!\leftarrow s_i$ for $k + 1 \leq i \leq n$

Knuth-Bendix Completion: Correctness Proof

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_*.

Knuth-Bendix Completion: Correctness Proof

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 $s_{i-1} \rightarrow_{R_\infty} s_i$ using $l \rightarrow r$, then $c(s_{i-1}, s_i) = (\{s_{i-1}\}, l, s_i)$.

(3) If $s_{i-1} \; {}_{R_\infty}\!\leftarrow s_i$ using $l \rightarrow r$, 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 $\succ$.

Knuth-Bendix Completion: Correctness Proof

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 4.30:
$\succ_C$ is a well-founded ordering.

Knuth-Bendix Completion: Correctness Proof

Lemma 4.31:
Let P be a proof in $E_\infty \cup R_\infty$. If P is not a rewrite proof in R_*, 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_*, then it contains

(a) a proof step that is in E_∞, or
(b) a proof step that is in $R_\infty \setminus R_*$, or
(c) a subproof $s_{i-1} \;{}_{R_*}\!\!\leftarrow s_i \rightarrow_{R_*} s_{i+1}$ (peak).

We show that in all three cases the proof step or subproof can be replaced by a smaller subproof:

Knuth-Bendix Completion: Correctness Proof

Case (a): A proof step using an equation $s \mathrel{\dot{\approx}} t$ is in E_∞. This equation must be deleted during the run.

If $s \mathrel{\dot{\approx}} t$ is deleted using *Orient*:

$$\dots s_{i-1} \leftrightarrow_{E_\infty} s_i \dots \qquad \Longrightarrow \qquad \dots s_{i-1} \rightarrow_{R_\infty} s_i \dots$$

If $s \mathrel{\dot{\approx}} t$ is deleted using *Delete*:

$$\dots s_{i-1} \leftrightarrow_{E_\infty} s_{i-1} \dots \qquad \Longrightarrow \qquad \dots s_{i-1} \dots$$

If $s \mathrel{\dot{\approx}} t$ is deleted using *Simplify-Eq*:

$$\dots s_{i-1} \leftrightarrow_{E_\infty} s_i \dots \qquad \Longrightarrow \qquad \dots s_{i-1} \rightarrow_{R_\infty} s' \leftrightarrow_{E_\infty} s_i \dots$$

Knuth-Bendix Completion: Correctness Proof

Case (b): A proof step using a rule $s \rightarrow t$ is in $R_\infty \setminus R_*$. This rule must be deleted during the run.

If $s \rightarrow t$ is deleted using *R-Simplify-Rule*:

$$\dots s_{i-1} \rightarrow_{R_\infty} s_i \dots \quad\Longrightarrow\quad \dots s_{i-1} \rightarrow_{R_\infty} s' \;{}_{R_\infty}\!\!\leftarrow s_i \dots$$

If $s \rightarrow t$ is deleted using *L-Simplify-Rule*:

$$\dots s_{i-1} \rightarrow_{R_\infty} s_i \dots \quad\Longrightarrow\quad \dots s_{i-1} \rightarrow_{R_\infty} s' \leftrightarrow_{E_\infty} s_i \dots$$

Knuth-Bendix Completion: Correctness Proof

Case (c): A subproof has the form $s_{i-1} \; {}_{R_*}\!\!\leftarrow s_i \rightarrow_{R_*} s_{i+1}$.

If there is no overlap or a non-critical overlap:

$$\ldots s_{i-1} \; {}_{R_*}\!\!\leftarrow s_i \rightarrow_{R_*} s_{i+1} \ldots \implies \ldots s_{i-1} \rightarrow^*_{R_*} s' \; {}_{R_*}^{*}\!\!\leftarrow s_{i+1} \ldots$$

If there is a critical pair that has been added using *Deduce*:

$$\ldots s_{i-1} \; {}_{R_*}\!\!\leftarrow s_i \rightarrow_{R_*} s_{i+1} \ldots \implies \quad \ldots s_{i-1} \leftrightarrow_{E_\infty} s_{i+1} \ldots$$

In all cases, checking that the replacement subproof is smaller than the replaced subproof is routine. □

Knuth-Bendix Completion: Correctness Proof

Theorem 4.32:
Let $(E_0; R_0) \Rightarrow_{KBC} (E_1; R_1) \Rightarrow_{KBC} (E_2; R_2) \Rightarrow_{KBC} \ldots$ be a fair run and let R_0 and E_* be empty. Then

(1) every proof in $E_\infty \cup R_\infty$ is equivalent to a rewrite proof in R_*,

(2) R_* is equivalent to E_0, and

(3) R_* is convergent.

Knuth-Bendix Completion: Correctness Proof

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 $R_* \subseteq R_\infty$, we get $\approx_{R_*} \subseteq \approx_{E_\infty \cup R_\infty}$. On the other hand, by (1), $\approx_{E_\infty \cup R_\infty} \subseteq \approx_{R_*}$.

(3) Since $\rightarrow_{R_*} \subseteq \succ$, R_* is terminating. By (1), R_* is confluent.

□

4.6 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.

Unfailing Completion

Let E be a set of equations, let $\succ$ be a reduction ordering.

We define the relation $\rightarrow_{E^\succ}$ by

$$
\begin{array}{lll}
s \rightarrow_{E^\succ} t & \text{iff} & \text{there exist } (u \approx v) \in E \text{ or } (v \approx u) \in E, \\
 & & p \in \text{pos}(s), \text{ and } \sigma : X \rightarrow \mathsf{T}_\Sigma(X), \\
 & & \text{such that } s|_p = u\sigma \text{ and } t = s[v\sigma]_p \\
 & & \text{and } u\sigma \succ v\sigma.
\end{array}
$$

Note: $\rightarrow_{E^\succ}$ is terminating by construction.

Unfailing Completion

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 $s \leftrightarrow_E^* t$ there exists a ground term v such that $s \rightarrow_{E^\succ}^* v \;{}_{E^\succ}^{*}\!\!\leftarrow t$.

(Analogously for $E \cup R$.)

Unfailing Completion

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 $s \not\prec t$, we approximate $\succ$ on ground terms by $\not\prec$ on arbitrary terms.

Unfailing Completion

Let $u_i \mathrel{\dot{\approx}} v_i$ $(i = 1, 2)$ be equations in E whose variables have been renamed such that $\mathrm{vars}(u_1 \mathrel{\dot{\approx}} v_1) \cap \mathrm{vars}(u_2 \mathrel{\dot{\approx}} v_2) = \emptyset$. Let $p \in \mathrm{pos}(u_1)$ be a position such that $u_1|_p$ is not a variable, σ 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 $\mathrm{SP}_\succ(E)$.

Semi-critical pairs of $E \cup R$ are defined analogously. If $\rightarrow_R \subseteq \succ$, then $\mathrm{CP}(R)$ and $\mathrm{SP}_\succ(R)$ agree.

Unfailing Completion

Note: In contrast to critical pairs, it may be necessary to consider overlaps of a rule 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.

Unfailing Completion

The *Deduce* rule takes now the following form:

Deduce

$$(E; R) \quad \Rightarrow_{UKBC} \quad (E \cup \{s \approx t\}; R)$$

if $\langle s, t\rangle \in \mathrm{SP}_{\succ}(E \cup R)$

The other rules are inherited from $\Rightarrow_{KBC}$. The fairness criterion for runs is replaced by

$$\mathrm{SP}_{\succ}(E_* \cup R_*) \subseteq E_\infty$$

(i. e., if every semi-critical pair between persisting rules or equations is computed at some step of the derivation).

Unfailing Completion

Analogously to Thm. 4.32 we obtain now the following theorem:

Theorem 4.33:
Let $(E_0; R_0) \Rightarrow_{UKBC} (E_1; R_1) \Rightarrow_{UKBC} (E_2; R_2) \Rightarrow_{UKBC} \dots$ be a fair run; let $R_0 = \emptyset$. Then

(1) $E_* \cup R_*$ is equivalent to E_0, and

(2) $E_* \cup R_*$ is ground convergent.

Unfailing Completion

Moreover one can show that, whenever there exists a *reduced* convergent R such that $\approx_{E_0} = \downarrow_R$ and $\rightarrow_R \in \succ$, then for every fair *and simplifying* run $E_* = \emptyset$ and $R_* = R$ 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_* is reduced, and for all equations $u \approx v \in E_*$, u and v are incomparable w. r. t. $\succ$ and irreducible w. r. t. R_*.

Unfailing Completion

Unfailing completion is refutationally complete for equational theories:

Theorem 4.34:
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 $(E_0; \emptyset) \Rightarrow_{UKBC} (E_1; R_1) \Rightarrow_{UKBC} (E_2; R_2) \Rightarrow_{UKBC} \ldots$ be a fair run of unfailing completion, then $s \approx_E t$ iff some $E_i \cup R_i$ contains $true \approx false$.

Unfailing Completion

Outlook:

Combine non-equational superposition 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.

Part 5: 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 inferences, 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/eliminated.

Coping with Large Sets of Formulas

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.)

5.1 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 Main Loop

Consequently: split the set of clauses into two subsets.

- Wo = "Worked-off" (or "active") clauses: Have already been selected as "given clause". (So all inferences between these clauses have already been computed.)
- Us = "Usable" (or "passive") clauses: Have not yet been selected as "given clause".

The Main Loop

During each iteration of the main loop:

Select a new given clause C from Us; $\mathit{Us} := \mathit{Us} \setminus \{C\}$.

Find partner clauses D_i from Wo; $\mathit{New} = \mathit{Infer}(\{ D_i \mid i \in I \}, C)$; $\mathit{Us} = \mathit{Us} \cup \mathit{New}$; $\mathit{Wo} = \mathit{Wo} \cup \{C\}$

The Main Loop

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.

The Main Loop

Design decision: should one also simplify *Us* using *Wo*?

yes $\rightsquigarrow$ “Full Reduction”:
Advantage: simplifications of *Us* may be useful to derive the empty clause.

no $\rightsquigarrow$ “Lazy Reduction”:
Advantage: clauses in *Us* 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.)

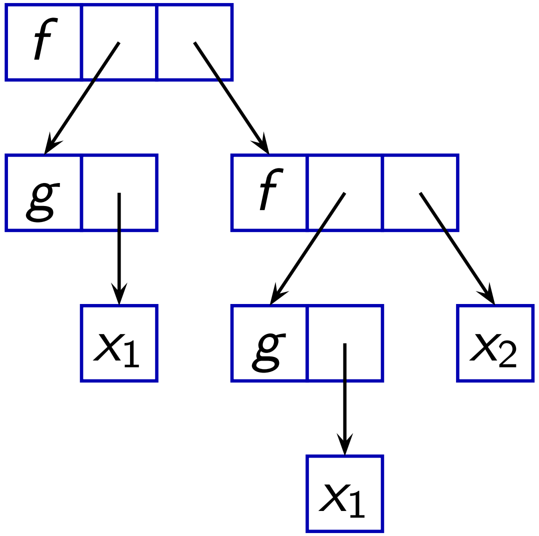
Main Loop Full Reduction

```
$Us = N$;
$Wo = \emptyset$;
while ($Us \neq \emptyset$ && $\bot \notin Us$) {
    Given = select clause from $Us$ and move it from $Us$ to $Wo$;
    $New$ = all inferences between Given and $Wo$;
    Reduce $New$ together with $Wo$ and $Us$;
    $Us = Us \cup New$;}
if ($\bot \in Us$)
    return "unsatisfiable";
else
    return "satisfiable";
```

5.2 Term Representations

The obvious data structure for terms: Trees

$f(g(x_1), f(g(x_1), x_2))$

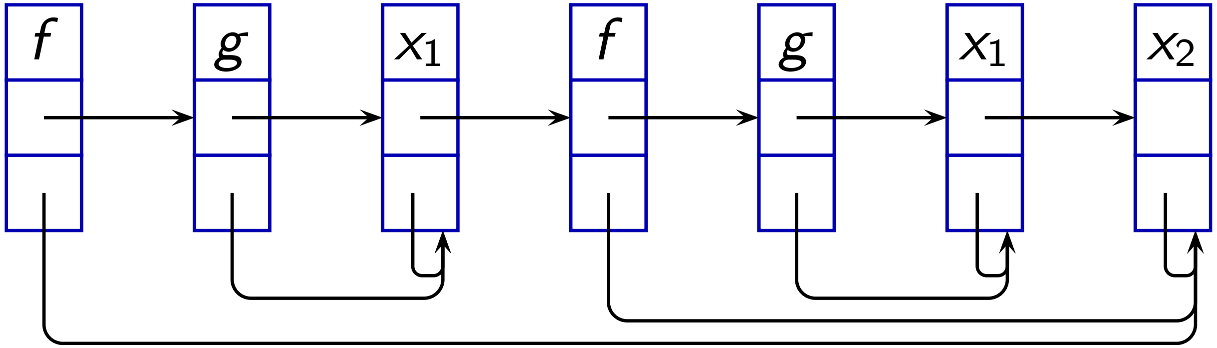


optionally: (full) sharing

Term Representations

An alternative: Flatterms

$f(g(x_1), f(g(x_1), x_2))$



need more memory;

but: better suited for preorder term traversal

and easier memory management.

5.3 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,
- ...

Index Data Structures

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).

Index Data Structures

Many different approaches:

- Path indexing
- Discrimination trees
- Substitution trees
- Context trees
- Feature vector indexing
- . . .

Index Data Structures

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 $f(g(*, b), *)$: $f.1.g.1.*$

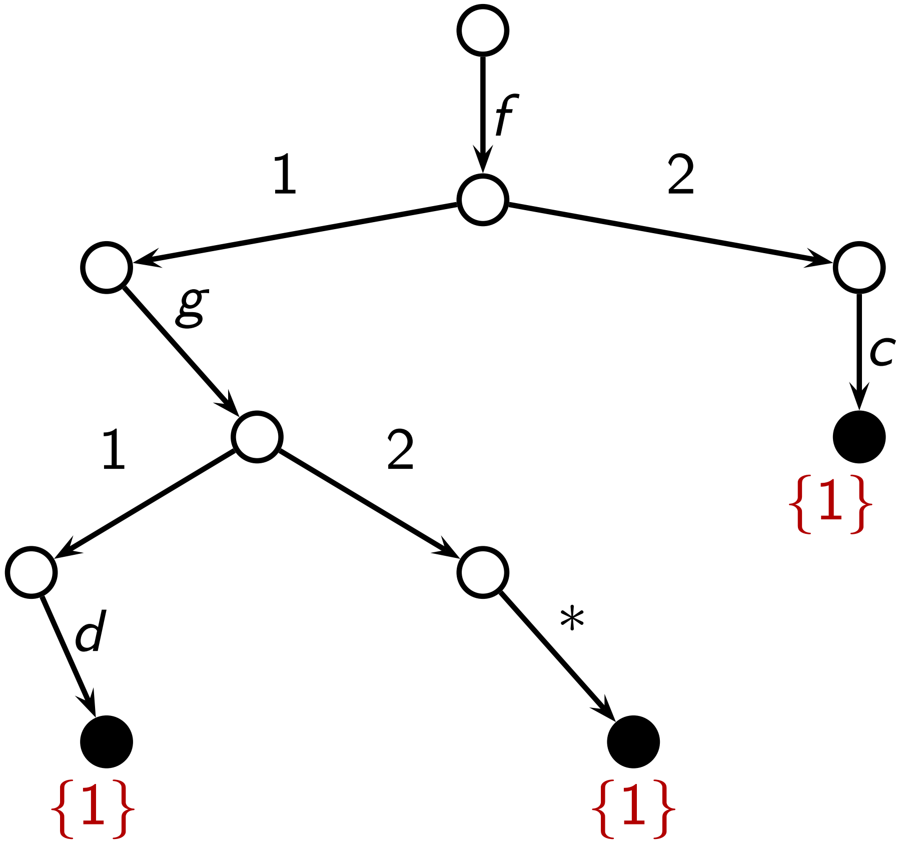
$f.1.g.2.b$

$f.2.*$

Each leaf of the trie contains the set of (pointers to) all terms that contain the respective path.

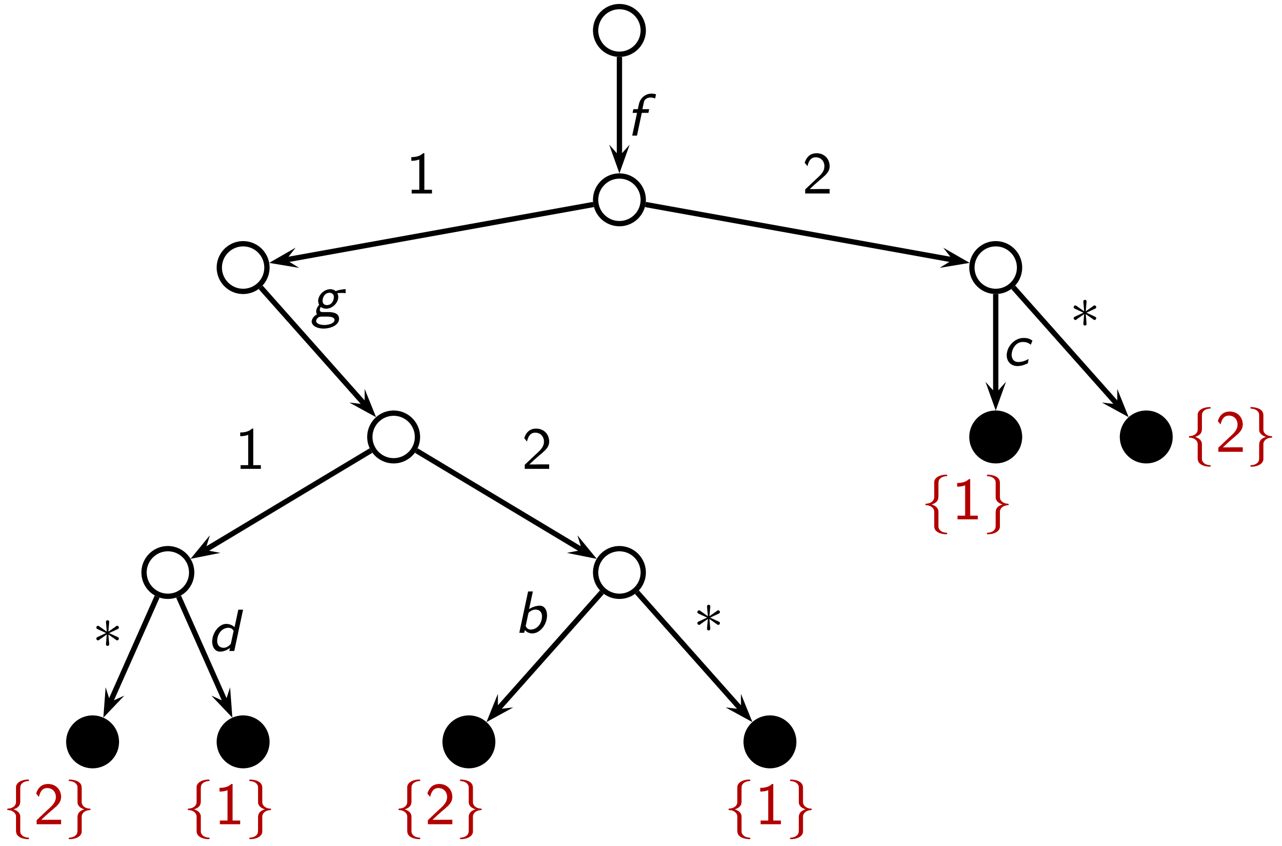
Path Indexing

Example: Path index for $\{f(g(d, *), c)\}$



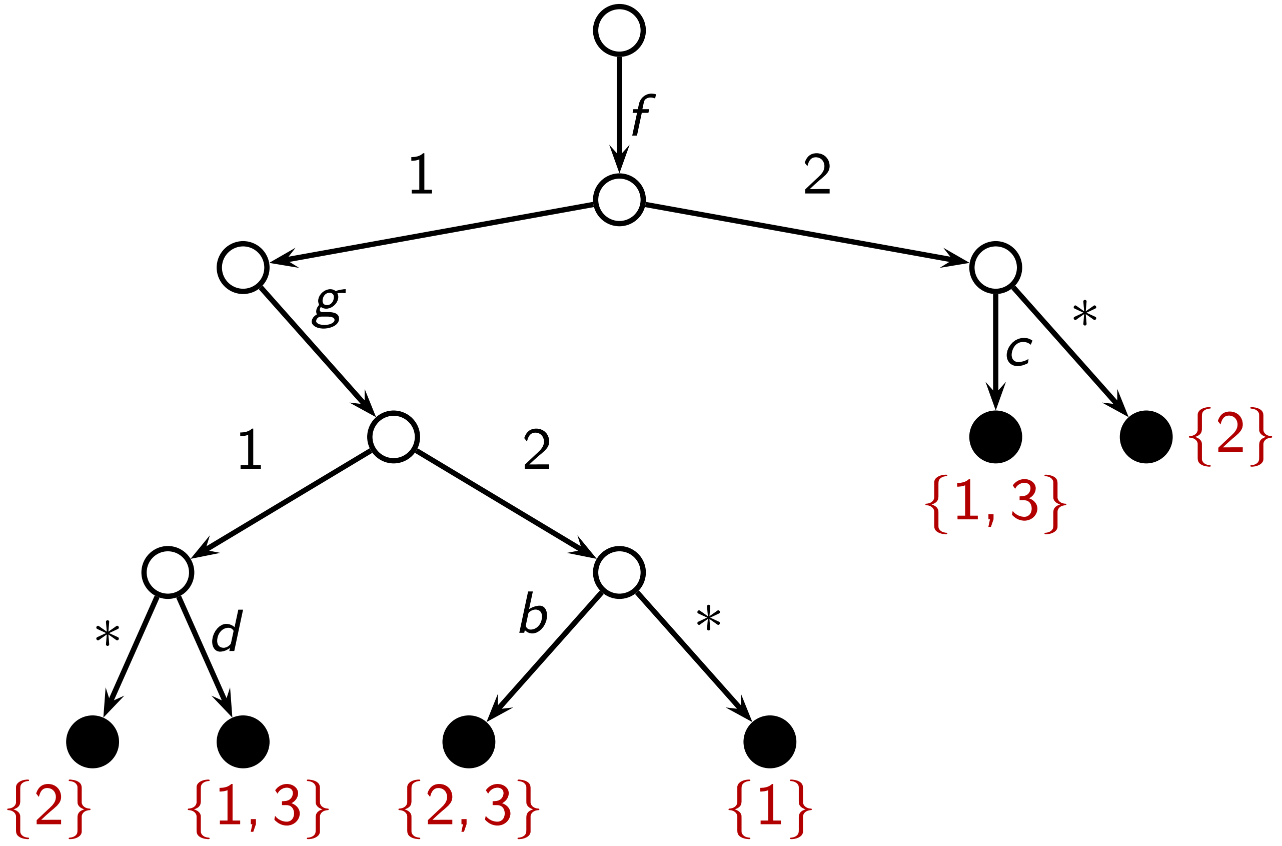
Path Indexing

Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*)\}$



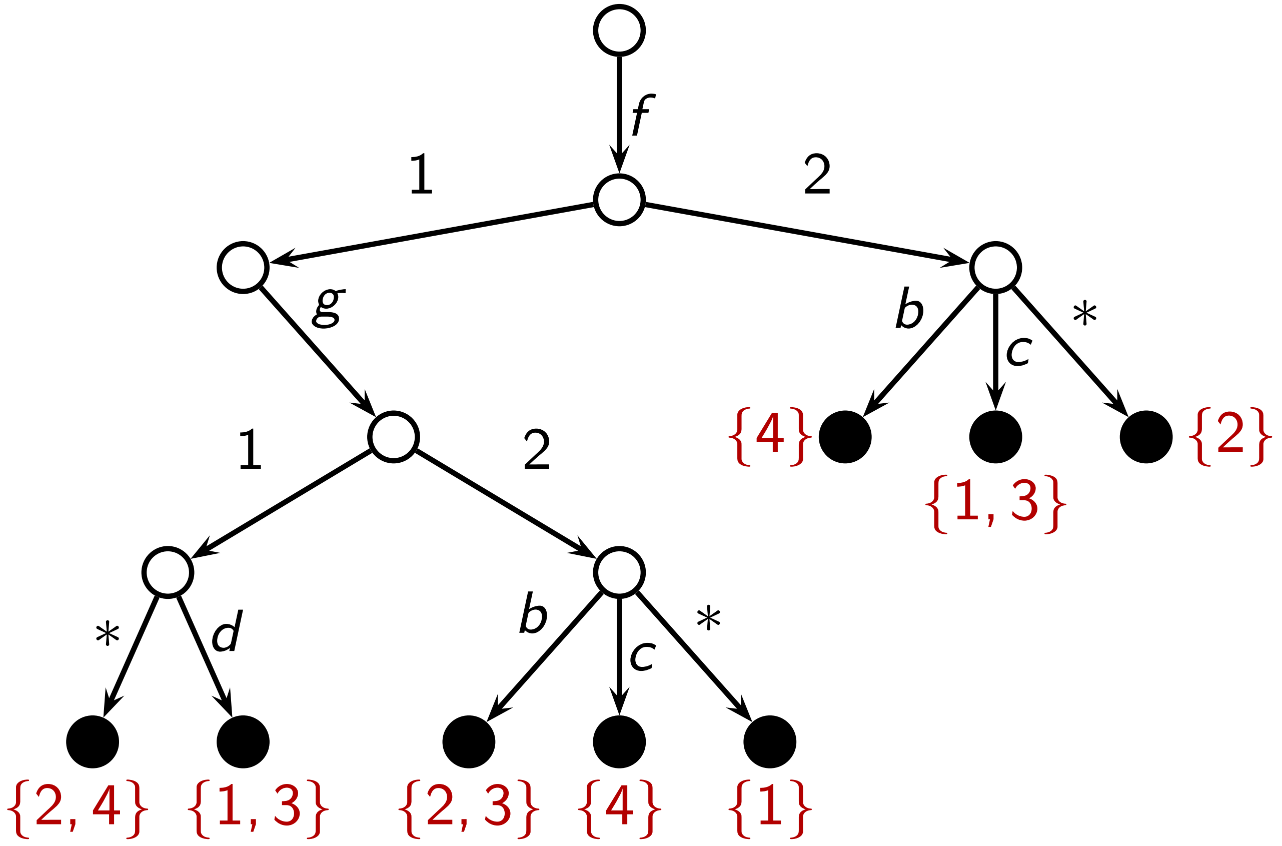
Path Indexing

Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*),$
$f(g(d,b),c)\}$



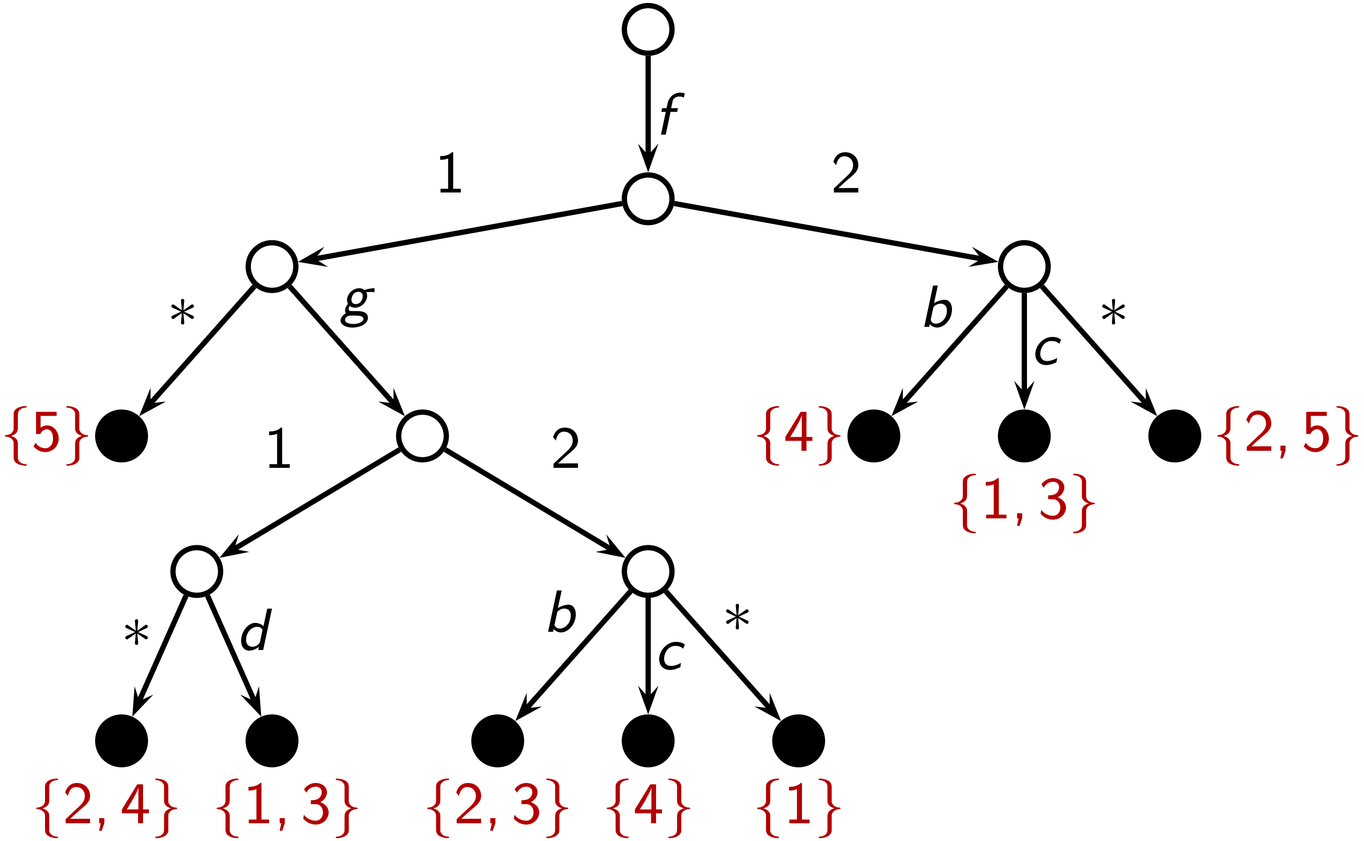
Path Indexing

Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b)\}$



Path Indexing

Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b),\ f(*,*)\}$



Path Indexing

Advantages:

Uses little space.

No backtracking for retrieval.

Efficient insertion and deletion.

Good for finding instances.

Disadvantages:

Retrieval requires combining intermediate results for subterms.

Discrimination Trees

Discrimination trees:

Preorder traversals of terms are encoded in a trie.

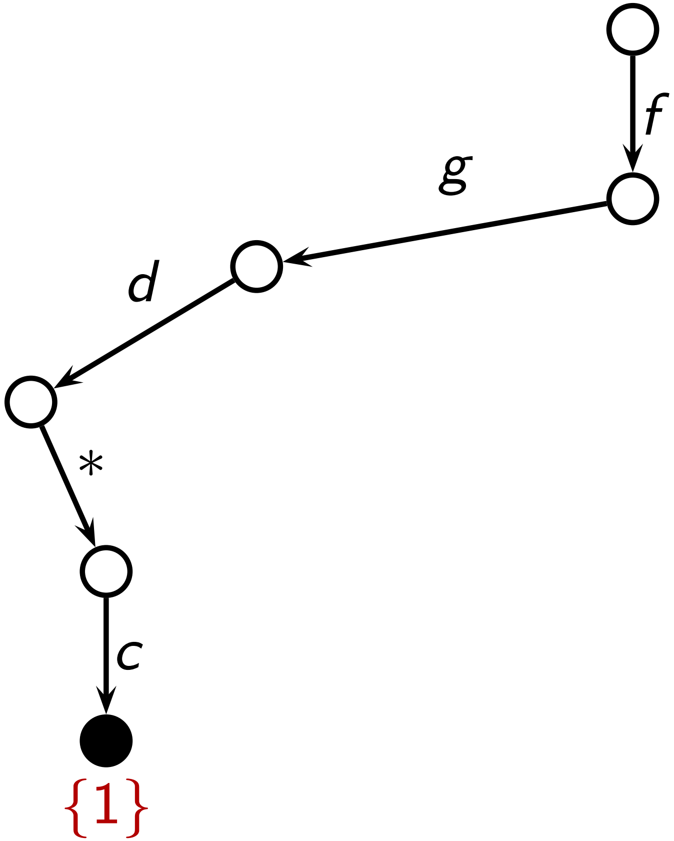
A star $*$ represents arbitrary variables.

Example: String of $f(g(*, b), *)$: $\quad f.g.*.b.*$

Each leaf of the trie contains (a pointer to) the term that is represented by the path.

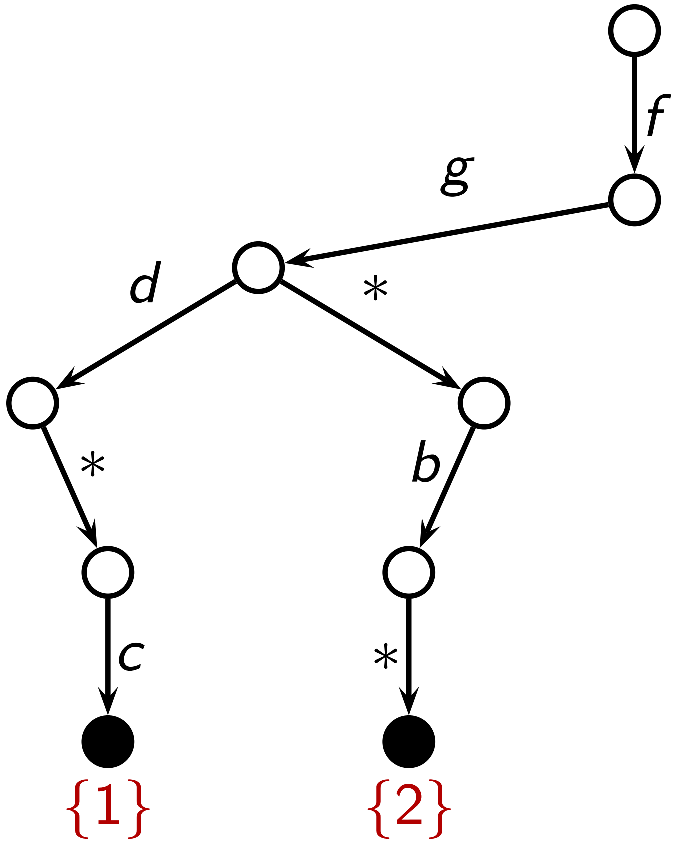
Discrimination Trees

Example: Discrimination tree for $\{f(g(d,*),c)\}$



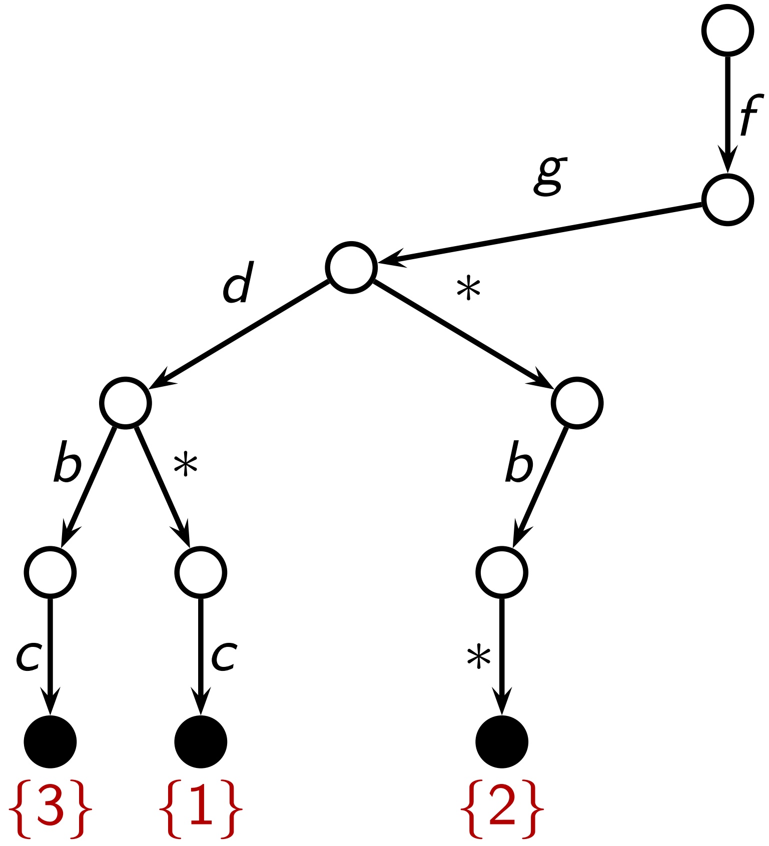
Discrimination Trees

Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*)\}$



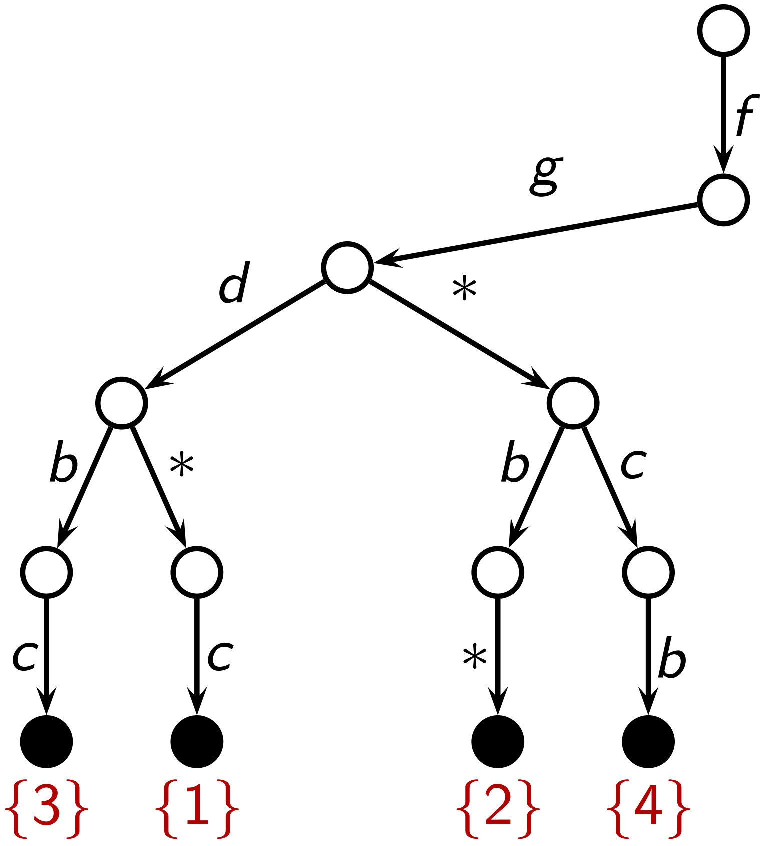
Discrimination Trees

Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c)\}$



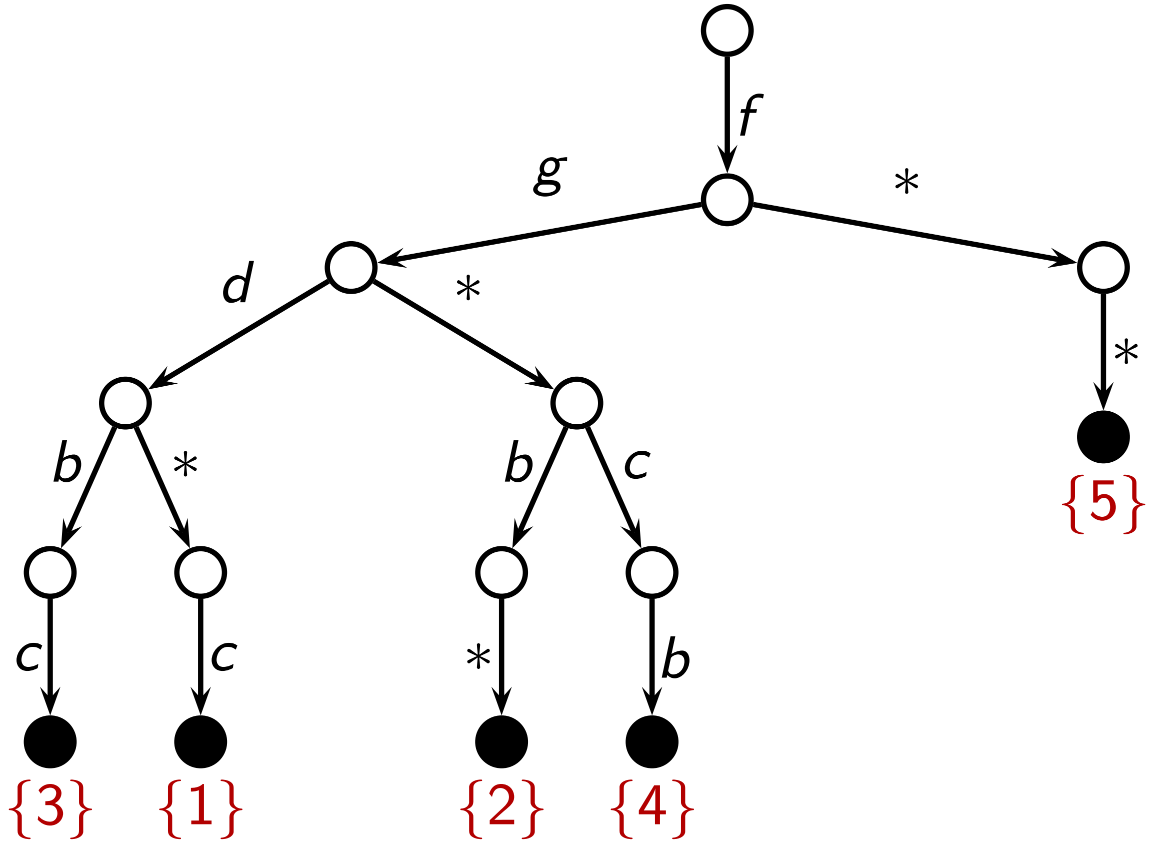
Discrimination Trees

Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b)\}$



Discrimination Trees

Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),$ $f(g(d,b),c),\ f(g(*,c),b),\ f(*,*)\}$



Discrimination Trees

Advantages:

Each leaf yields one term, hence retrieval does not require intersections of intermediate results for subterms.

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.

Backtracking required for retrieval.

Feature Vector Indexing

Goal:

C' is subsumed by C if $C' = C\sigma \vee D$.

Find all clauses C' for a given C or vice versa.

Feature Vector Indexing

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.
- $\ldots$

Feature Vector Indexing

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.

Feature Vector Indexing

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

Literature:

R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov: Term Indexing, Ch. 26 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. II*, Elsevier, 2001.

Christoph Weidenbach: Combining Superposition, Sorts and Splitting, Ch. 27 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. II*, Elsevier, 2001.

Part 6: 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.

Termination Revisited

Literature:

Nao Hirokawa and Aart Middeldorp: Dependency Pairs Revisited, RTA 2004, pp. 249-268 (in particular Sect. 1–4).

Thomas Arts and Jürgen Giesl: Termination of Term Rewriting Using Dependency Pairs, Theoretical Computer Science, 236:133-178, 2000.

6.1 Dependency Pairs

Invented by T. Arts and J. Giesl in 1996, many refinements since then.

Given: finite TRS R over $\Sigma = (\Omega, \emptyset)$.

$T_0 := \{\, t \in \mathrm{T}_\Sigma(X) \mid \exists$ infinite deriv. $t \rightarrow_R t_1 \rightarrow_R t_2 \rightarrow_R \ldots \,\}$.

$$\begin{aligned} T_\infty &:= \{\, t \in T_0 \mid \forall p > \varepsilon : t|_p \notin T_0 \,\} \\ &= \text{minimal elements of } T_0 \text{ w. r. t. } \rhd. \end{aligned}$$

$t \in T_0 \Rightarrow$ there exists a $t' \in T_\infty$ such that $t \trianglerighteq t'$.

R is non-terminating iff $T_0 \neq \emptyset$ iff $T_\infty \neq \emptyset$.

Dependency Pairs

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 $l \to r \in R$ must be applied at the root of t (possibly preceded by rewrite steps below the root):

$$t = f(t_1, \dots, t_n) \xrightarrow{>\varepsilon}{}^{*}_{R} f(s_1, \dots, s_n) = l\sigma \xrightarrow{\varepsilon}_{R} r\sigma.$$

In particular, $root(t) = root(l)$, so we see that the root symbol of any term in T_∞ must be contained in $D := \{\, root(l) \mid l \to r \in R \,\}$. D is called the set of defined symbols of R; $C := \Omega \setminus D$ is called the set of constructor symbols of R.

Dependency Pairs

The term $r\sigma$ is contained in T_0, so there exists a $v \in T_\infty$ such that $r\sigma \trianglerighteq v$.

If v occurred in $r\sigma$ at or below a variable position of r, then $x\sigma|_p = v$ for some $x \in var(r) \subseteq var(l)$, hence $s_i \trianglerighteq x\sigma$ 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 $root(u) = root(v) \in D$. Moreover, u cannot be a proper subterm of l, since otherwise again there would be an infinite derivation starting from some t_i.

Dependency Pairs

Putting everything together, we obtain

$$t = f(t_1, \ldots, t_n) \xrightarrow{>\varepsilon}{}_R^{*} f(s_1, \ldots, s_n) = l\sigma \xrightarrow{\varepsilon}{}_R r\sigma \trianglerighteq u\sigma$$

where $r \trianglerighteq u$, $root(u) \in D$, $l \not\triangleright u$, u is not a variable.

Since $u\sigma \in T_\infty$, we can continue this process and obtain an infinite sequence.

Dependency Pairs

If we define

$$S := \{\, l \to u \mid l \to r \in R,\ r \trianglerighteq u,\ root(u) \in D,\ l \not\triangleright u,\ u \notin X \,\},$$

we can combine the rewrite step at the root and the subterm step and obtain

$$t \xrightarrow{>\varepsilon}{}^{*}_{R} l\sigma \xrightarrow{\varepsilon}_{S} u\sigma.$$

Dependency Pairs

To get rid of the superscripts ε 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 $t = f(t_1, \dots, t_n)$ we define $t^\sharp := f^\sharp(t_1, \dots, t_n)$; 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

$$DP(R) := \{\, l^\sharp \to u^\sharp \mid l \to r \in R,\ r \trianglerighteq u,\ root(u) \in D,\ l \not\triangleright u,\ u \notin X \,\}.$$

Dependency Pairs

For $t \in T_\infty$, the sequence using the S-rule corresponds now to

$$t^\sharp \to_R^* l^\sharp\sigma \to_{DP(R)} u^\sharp\sigma$$

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 $DP(R)$ come from $\Omega^\sharp$, so rules from R can only be applied below the root and rules from $DP(R)$ can only be applied at the root.)

Dependency Pairs

Since $u^\sharp\sigma$ is again in $T^\sharp_\infty$, we can continue the process in the same way. We obtain: R is non-terminating iff there is an infinite sequence

$$t_1 \rightarrow_R^* t_2 \rightarrow_{DP(R)} t_3 \rightarrow_R^* t_4 \rightarrow_{DP(R)} \cdots$$

with $t_i \in T^\sharp_\infty$ 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 $s \to t \in DP(R)$

edges: from $s \to t$ to $u \to v$ if there are σ, τ such that $t\sigma \to_R^* u\tau$.

Dependency Graphs

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:

Dependency Graphs

The functions *cap* and *ren* are defined by:

$$\begin{aligned} cap(x) &= x \\ cap(f(t_1,\ldots,t_n)) &= \begin{cases} y & \text{if } f \in D \\ f(cap(t_1),\ldots,cap(t_n)) & \text{if } f \in C \cup D^\sharp \end{cases} \end{aligned}$$

$$\begin{aligned} ren(x) &= y, \quad y \text{ fresh} \\ ren(f(t_1,\ldots,t_n)) &= f(ren(t_1),\ldots,ren(t_n)) \end{aligned}$$

The overapproximated dependency graph contains an edge from $s \to t$ to $u \to v$ if $ren(cap(t))$ and u are unifiable.

Dependency Graphs

A cycle in the dependency graph is a non-empty subset $K \subseteq DP(R)$ such that there is a non-empty path 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

$$t_1 \to_R^* t_2 \to_K t_3 \to_R^* t_4 \to_K \dots$$

with $t_i \in T_\infty^\sharp$ 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.

6.2 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 π is a mapping $\pi : \Omega^\sharp \to \mathbb{N}$ such that $\pi(f^\sharp) = i \in \{1, \dots, \mathit{arity}(f^\sharp)\}$.

We define $\pi(f^\sharp(t_1, \dots, t_n)) = t_{\pi(f^\sharp)}$.

Subterm Criterion

Theorem 6.1 (Hirokawa and Middeldorp):
Let K be a cycle in $DG(R)$. If there is a simple projection π for K such that $\pi(l) \trianglerighteq \pi(r)$ for every $l \to r \in K$ and $\pi(l) \vartriangleright \pi(r)$ for some $l \to r \in K$, then there are no K-minimal sequences.

Subterm Criterion

Proof:

Suppose that

$$t_1 \rightarrow_R^* u_1 \rightarrow_K t_2 \rightarrow_R^* u_2 \rightarrow_K \dots$$

is a K-minimal infinite rewrite sequence. Apply π to every t_i:

Case 1: $u_i \rightarrow_K t_{i+1}$. 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, $\pi(l) \trianglerighteq \pi(r)$. 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: $t_i \rightarrow_R^* u_i$. Then $\pi(t_i) \to \pi(u_i)$.

Subterm Criterion

By applying π to every term in the K-minimal infinite rewrite sequence, we obtain an infinite $(\to_R \cup \rhd)$-sequence containing infinitely many $\rhd$-steps. Since $\rhd$ is well-founded, there must also exist infinitely many $\to_R$-steps (otherwise the infinite sequence would have an infinite tail consisting only of $\rhd$-steps, contradicting well-foundedness.)

Now note that $\rhd \circ \to_R \subseteq \to_R \circ \rhd$. Therefore we can commute $\rhd$-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 $t_1 \rhd \pi(t_1)$ and $t_1 \in T_\infty$, so there cannot be an infinite $\to_R$-sequence starting from $\pi(t_1)$. $\square$

Subterm Criterion

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.)[a]

Important property: Every cycle is contained in some SCC.

[a] There are several definitions of SCCs that differ in the treatment of edges from a node to itself.

Subterm Criterion

Idea: Search for a simple projection π such that $\pi(l) \trianglerighteq \pi(r)$ for all DPs $l \rightarrow r$ in the SCC. Delete all DPs in the SCC for which $\pi(l) \vartriangleright \pi(r)$ (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.

6.3 Reduction Pairs and Argument Filterings

Goal: Show the non-existence of K-minimal infinite rewrite sequences

$$t_1 \rightarrow_R^* u_1 \rightarrow_K t_2 \rightarrow_R^* u_2 \rightarrow_K \dots$$

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.

Reduction Pairs and Argument Filterings

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 $\succsim$.)

Reduction Pairs and Argument Filterings

Clearly, for any reduction ordering $\succ$, $(\succeq, \succ)$ is a reduction pair. More general reduction pairs can be obtained using argument filterings:

Argument filtering π:

$$\pi : \Omega \cup \Omega^\sharp \to \mathbb{N} \cup \text{list of } \mathbb{N}$$

$$\pi(f) = \begin{cases} i \in \{1, \dots, \mathit{arity}(f)\}, \text{ or} \\ [i_1, \dots, i_k], \text{ where } 1 \le i_1 < \cdots < i_k \le \mathit{arity}(f), \\ \qquad\qquad\qquad 0 \le k \le \mathit{arity}(f) \end{cases}$$

Reduction Pairs and Argument Filterings

Extension to terms:

$\pi(x) = x$

$\pi(f(t_1, \ldots, t_n)) = \pi(t_i)$, if $\pi(f) = i$

$\pi(f(t_1, \ldots, t_n)) = f'(\pi(t_{i_1}), \ldots, \pi(t_{i_k}))$, if $\pi(f) = [i_1, \ldots, i_k]$, where f'/k is a new function symbol.

Reduction Pairs and Argument Filterings

Let $\succ$ be a reduction ordering, let π be an argument filtering. Define $s \succ_\pi t$ iff $\pi(s) \succ \pi(t)$ and $s \succsim_\pi t$ iff $\pi(s) \succeq \pi(t)$.

Lemma 6.2:
$(\succsim_\pi, \succ_\pi)$ is a reduction pair.

Proof:
Follows from the following two properties:

$\pi(s\sigma) = \pi(s)\sigma_\pi$, where $\sigma_\pi(x) := \pi(\sigma(x))$.

$$\pi(s[u]_p) = \begin{cases} \pi(s), \text{if } p \text{ does not correspond to any position in } \pi(s) \\ \pi(s)[\pi(u)]_q, \text{if } p \text{ corresponds to } q \text{ in } \pi(s) \end{cases}$$

□

Reduction Pairs and Argument Filterings

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 and Argument Filterings

Reduction pairs by interpretation:

Let $\mathcal{A}$ be a Σ-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 $s \succsim_{\mathcal{A}} t$ iff $\mathcal{A}(\beta)(s) \succeq \mathcal{A}(\beta)(t)$ for all assignments $\beta : X \to U_{\mathcal{A}}$; define $s \succ_{\mathcal{A}} t$ iff $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(t)$ for all assignments $\beta : X \to U_{\mathcal{A}}$.

Then $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$ is a reduction pair.

Reduction Pairs and Argument Filterings

For polynomial orderings, this definition permits interpretations of function symbols where some variable does not occur at all (e. g., $P_f(X, Y) = 2X + 1$ for a *binary* function symbol). It is no longer required that *every* variable must occur with some positive coefficient.

Reduction Pairs and Argument Filterings

Theorem 6.3 (Arts and Giesl):
Let K be a cycle in the dependency graph of the TRS R. If there is a reduction pair $(\succsim, \succ)$ such that

- $l \succsim r$ for all $l \to r \in R$,
- $l \succsim r$ or $l \succ r$ for all $l \to r \in K$,
- $l \succ r$ for at least one $l \to r \in K$,

then there is no K-minimal infinite sequence.

Reduction Pairs and Argument Filterings

Proof:

Assume 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.

As $l \succsim r$ for all $l \to r \in R$, we obtain $t_i \succsim u_i$ by stability under substitutions, compatibility with contexts, reflexivity and transitivity.

As $l \succsim r$ or $l \succ r$ for all $l \to r \in K$, we obtain $u_i \; ({\succsim} \cup {\succ}) \; t_{i+1}$ by stability under substitutions.

So we get an infinite $({\succsim} \cup {\succ})$-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 $\succsim$ and $\succ$, we can transform this into an infinite $\succ$-sequence, contradicting well-foundedness. $\square$

Reduction Pairs and Argument Filterings

The idea can be extended to SCCs in the same way as for the subterm criterion:

Search for a reduction pair $(\succsim, \succ)$ such that $l \succsim r$ for all $l \to r \in R$ and $l \succsim r$ or $l \succ r$ for all DPs $l \to r$ 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.

Reduction Pairs and Argument Filterings

Example: Consider the following TRS R from [Arts and Giesl]:

$$minus(x, 0) \to x \tag{1}$$

$$minus(s(x), s(y)) \to minus(x, y) \tag{2}$$

$$quot(0, s(y)) \to 0 \tag{3}$$

$$quot(s(x), s(y)) \to s(quot(minus(x, y), s(y))) \tag{4}$$

(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 $s(x)$.)

Reduction Pairs and Argument Filterings

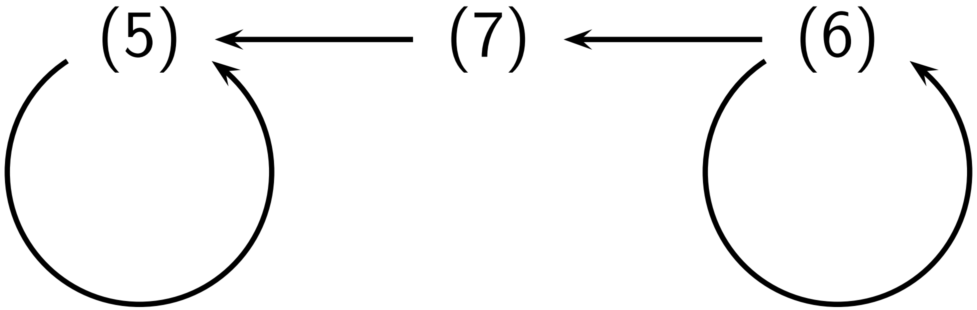
R has three dependency pairs:

$$minus^\sharp(s(x), s(y)) \to minus^\sharp(x, y) \tag{5}$$

$$quot^\sharp(s(x), s(y)) \to quot^\sharp(minus(x, y), s(y)) \tag{6}$$

$$quot^\sharp(s(x), s(y)) \to minus^\sharp(x, y) \tag{7}$$

The dependency graph of R is



Reduction Pairs and Argument Filterings

There are exactly two SCCs (and also two cycles). The cycle at (5) can be handled using the subterm criterion with $\pi(\mathit{minus}^\sharp) = 1$. For the cycle at (6) we can use an argument filtering π that maps *minus* to 1 and leaves all other function symbols unchanged (that is, $\pi(g) = [1, \ldots, \mathit{arity}(g)]$ for every g different from *minus*.) After applying the argument filtering, we compare left and right-hand sides using an LPO with precedence $\mathit{quot} > s$ (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

$$\frac{(G, R)}{(G_1, R_1), \quad \ldots, \quad (G_n, R_n)}$$

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).

DP Processors

Examples:

$$\frac{(G, R)}{(SCC_1, R), \;\; \ldots, \;\; (SCC_n, R)}$$

where $SCC_1, \ldots, SCC_n$ are the strongly conn. components of G.

$$\frac{(G, R)}{(G \setminus N, R)}$$

if there is an SCC of G and a simple projection π such that $\pi(l) \trianglerighteq \pi(r)$ for all DPs $l \to r$ in the SCC, and N is the set of DPs of the SCC for which $\pi(l) \triangleright \pi(r)$.

(and analogously for reduction pairs)

Innermost Termination

The dependency method can also be used for proving termination of innermost rewriting: $s \xrightarrow{\text{i}}_R t$ if $s \to_R t$ 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.)

6.4 Superposition

Goal:

Combine the ideas of superposition for first-order logic without equality (overlap maximal literals in a clause) and Knuth-Bendix completion (overlap maximal sides of equations) to get a calculus for equational clauses.

Observation

It is possible to encode an arbitrary predicate p using a function f_p and a new constant tt:

$$\begin{array}{rcr} P(t_1, \ldots, t_n) & \leadsto & f_P(t_1, \ldots, t_n) \approx tt \\ \neg\, P(t_1, \ldots, t_n) & \leadsto & \neg\, f_P(t_1, \ldots, t_n) \approx tt \end{array}$$

In equational logic it is therefore sufficient to consider the case that $\Pi = \emptyset$, i. e., equality is the only predicate symbol.

Abbreviation: $s \not\approx t$ instead of $\neg\, s \approx t$.

The Superposition Calculus – Informally

Conventions:

From now on: $\Pi = \emptyset$ (equality is the only predicate).

Inference rules are to be read modulo symmetry of the equality symbol.

We will first explain the ideas and motivations behind the superposition calculus and its completeness proof. Precise definitions will be given later.

The Superposition Calculus – Informally

Ground inference rules:

Superposition Right:

$$\frac{D' \lor t \approx t' \qquad C' \lor s[t] \approx s'}{D' \lor C' \lor s[t'] \approx s'}$$

Superposition Left:

$$\frac{D' \lor t \approx t' \qquad C' \lor s[t] \not\approx s'}{D' \lor C' \lor s[t'] \not\approx s'}$$

Equality Resolution:

$$\frac{C' \lor s \not\approx s}{C'}$$

(Note: We will need one further inference rule.)

The Superposition Calculus – Informally

Ordering restrictions:

Some considerations:

The literal ordering must depend primarily on the larger term of an equation.

As in the resolution case, negative literals must be a bit larger than the corresponding positive literals.

Additionally, we need the following property:
If $s \succ t \succ u$, then $s \not\approx u$ must be larger than $s \approx t$.
In other words, we must compare first the larger term, then the polarity, and finally the smaller term.

The Superposition Calculus – Informally

The following construction has the required properties:

Let $\succ$ be a *reduction ordering that is total on ground terms.*

To a positive literal $s \approx t$, we assign the multiset $\{s, t\}$, to a negative literal $s \not\approx t$ the multiset $\{s, s, t, t\}$. The literal ordering $\succ_L$ compares these multisets using the multiset extension of $\succ$.

The clause ordering $\succ_C$ compares clauses by comparing their multisets of literals using the multiset extension of $\succ_L$.

The Superposition Calculus – Informally

Ordering restrictions:

Ground inferences are necessary only if the following conditions are satisfied:

- In superposition inferences, the left premise is smaller than the right premise.
- The literals that are involved in the inferences are maximal in the respective clauses
(strictly maximal for positive literals in superposition inferences).
- In these literals, the lhs is greater than or equal to the rhs (in superposition inferences: greater than the rhs).

The Superposition Calculus – Informally

Model construction:

We want to use roughly the same ideas as in the completenes proof for superposition on first-order without equality.

But: a Herbrand interpretation does not work for equality: The equality symbol $\approx$ must be interpreted by equality in the interpretation.

The Superposition Calculus – Informally

Solution: Define a set E of ground equations and take $\mathrm{T}_\Sigma(\emptyset)/E = \mathrm{T}_\Sigma(\emptyset)/{\approx_E}$ as the universe.

Then two ground terms s and t are equal in the interpretation, if and only if $s \approx_E t$.

If E is a terminating and confluent rewrite system R, then two ground terms s and t are equal in the interpretation, if and only if $s \downarrow_R t$.

The Superposition Calculus – Informally

One problem:

In the completeness proof for the resolution calculus, the following property holds:

> If $C = C' \vee A$ with a strictly maximal and positive literal A is false in the current interpretation, then adding A to the current interpretation cannot make any literal of C' true.

This does not hold for superposition:

> Let $b \succ c \succ d$.
> Assume that the current rewrite system (representing the current interpretation) contains the rule $c \to d$.
> Now consider the clause $b \approx c \vee b \approx d$.

The Superposition Calculus – Informally

We need a further inference rule to deal with clauses of this kind, either the “Merging Paramodulation” rule of Bachmair and Ganzinger or the following “Equality Factoring” rule due to Nieuwenhuis:

Equality Factoring: $$\frac{C' \vee s \approx t' \vee s \approx t}{C' \vee t \not\approx t' \vee s \approx t'}$$

Note: This inference rule subsumes the usual factoring rule.

The Superposition Calculus – Informally

How do the non-ground versions of the inference rules for superposition look like?

Main idea as in non-equational first-order case:

Replace identity by unifiability.

Apply the mgu to the resulting clause.

In the ordering restrictions, replace $\succ$ by $\not\preceq$.

The Superposition Calculus – Informally

However:

As in Knuth-Bendix completion, we do not want to consider overlaps at or below a variable position.

Consequence: there are inferences between ground instances $D\theta$ and $C\theta$ of clauses D and C which are *not* ground instances of inferences between D and C.

Such inferences have to be treated in a special way in the completeness proof.

The Superposition Calculus – Formally

Until now, we have seen most of the ideas behind the superposition calculus and its completeness proof.

We will now start again from the beginning giving precise definitions and proofs.

Inference rules are applied with respect to the commutativity of equality $\approx$.

The Superposition Calculus – Formally

Inference rules (part 1):

Superposition Right:
$$\frac{D' \lor t \approx t' \qquad C' \lor s[u] \approx s'}{(D' \lor C' \lor s[t'] \approx s')\sigma}$$

where $\sigma = \text{mgu}(t, u)$ and
u is not a variable.

Superposition Left:
$$\frac{D' \lor t \approx t' \qquad C' \lor s[u] \not\approx s'}{(D' \lor C' \lor s[t'] \not\approx s')\sigma}$$

where $\sigma = \text{mgu}(t, u)$ and
u is not a variable.

The Superposition Calculus – Formally

Inference rules (part 2):

Equality Resolution:
$$\frac{C' \lor s \not\approx s'}{C'\sigma}$$

where $\sigma = \text{mgu}(s, s')$.

Equality Factoring:
$$\frac{C' \lor s' \approx t' \lor s \approx t}{(C' \lor t \not\approx t' \lor s \approx t')\sigma}$$

where $\sigma = \text{mgu}(s, s')$.

The Superposition Calculus – Formally

Theorem 6.4:
All inference rules of the superposition calculus are correct, i. e., for every rule

$$\frac{C_n, \ldots, C_1}{C_0}$$

we have $\{C_1, \ldots, C_n\} \models C_0$.

Proof:
Exercise. □

The Superposition Calculus – Formally

Orderings:

Let $\succ$ be a *reduction ordering that is total on ground terms.*

To a positive literal $s \approx t$, we assign the multiset $\{s, t\}$, to a negative literal $s \not\approx t$ the multiset $\{s, s, t, t\}$. The literal ordering $\succ_L$ compares these multisets using the multiset extension of $\succ$.

The clause ordering $\succ_C$ compares clauses by comparing their multisets of literals using the multiset extension of $\succ_L$.

The Superposition Calculus – Formally

Inferences have to be computed only if the following ordering restrictions are satisfied:

- In superposition inferences, after applying the unifier to both premises, the left premise is not greater than or equal to the right one.
- The last literal in each premise is maximal in the respective premise, i. e., there exists no greater literal
  (strictly maximal for positive literals in superposition inferences, i. e., there exists no greater or equal literal).
- In these literals, the lhs is not smaller than the rhs
  (in superposition inferences: neither smaller nor equal).

The Superposition Calculus – Formally

Superposition Left in Detail:

$$\frac{D' \lor t \approx t' \qquad C' \lor s[u] \not\approx s'}{(D' \lor C' \lor s[t'] \not\approx s')\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$,

u is not a variable,

$t\sigma \not\preceq t'\sigma$, $s\sigma \not\preceq s'\sigma$

$(t \approx t')\sigma$ strictly maximal in $(D' \lor t \approx t')\sigma$, nothing selected

$(s \not\approx s')\sigma$ maximal in $(C' \lor s \not\approx s')\sigma$ or selected

The Superposition Calculus – Formally

Superposition Right in Detail:

$$\frac{D' \vee t \approx t' \qquad C' \vee s[u] \approx s'}{(D' \vee C' \vee s[t'] \approx s')\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$,

u is not a variable,

$t\sigma \not\preceq t'\sigma$, $s\sigma \not\preceq s'\sigma$

$(t \approx t')\sigma$ strictly maximal in $(D' \vee t \approx t')\sigma$, nothing selected

$(s \approx s')\sigma$ strictly maximal in $(C' \vee s \approx s')\sigma$, nothing selected

The Superposition Calculus – Formally

Equality Resolution in Detail:

$$\frac{C' \vee s \not\approx s'}{C'\sigma}$$

where $\sigma = \mathrm{mgu}(s, s')$,

$(s \not\approx s')\sigma$ maximal in $(C' \vee s \approx s')\sigma$ or selected

The Superposition Calculus – Formally

Equality Factoring in Detail:

$$\frac{C' \lor s' \approx t' \lor s \approx t}{(C' \lor t \not\approx t' \lor s \approx t')\sigma}$$

where $\sigma = \mathrm{mgu}(s, s')$,

$s'\sigma \not\preceq t'\sigma$, $s\sigma \not\preceq t\sigma$

$(s \approx t)\sigma$ maximal in $(C' \lor s' \approx t' \lor s \approx t)\sigma$, nothing selected

The Superposition Calculus – Formally

A ground clause C is called redundant w. r. t. a set of ground clauses N, if it follows from clauses in N that are smaller than C.

A clause is redundant w. r. t. a set of clauses N, if all its ground instances are redundant w. r. t. $G_\Sigma(N)$.

The set of all clauses that are redundant w. r. t. N is denoted by $\mathit{Red}(N)$.

N is called saturated up to redundancy, if the conclusion of every inference from clauses in $N \setminus \mathit{Red}(N)$ is contained in $N \cup \mathit{Red}(N)$.

Superposition: Refutational Completeness

For a set E of ground equations, $\mathrm{T}_\Sigma(\emptyset)/E$ is an E-interpretation (or E-algebra) with universe $\{\, [t] \mid t \in \mathrm{T}_\Sigma(\emptyset) \,\}$.

One can show (similar to the proof of Birkhoff's Theorem) that for every *ground* equation $s \approx t$ we have $\mathrm{T}_\Sigma(\emptyset)/E \models s \approx t$ if and only if $s \leftrightarrow_E^* t$.

In particular, if E is a convergent set of rewrite rules R and $s \approx t$ is a ground equation, then $\mathrm{T}_\Sigma(\emptyset)/R \models s \approx t$ if and only if $s \downarrow_R t$. By abuse of terminology, we say that an equation or clause is valid (or true) in R if and only if it is true in $\mathrm{T}_\Sigma(\emptyset)/R$.

Superposition: Refutational Completeness

Construction of candidate interpretations
(Bachmair & Ganzinger 1990):

Let N be a set of clauses not containing $\bot$.
Using induction on the clause ordering we define sets of rewrite rules E_C and R_C for all $C \in G_\Sigma(N)$ as follows:

Assume that E_D has already been defined for all $D \in G_\Sigma(N)$ with $D \prec_C C$. Then $R_C = \bigcup_{D \prec_C C} E_D$.

Superposition: Refutational Completeness

The set E_C contains the rewrite rule $s \to t$, if

(a) $C = C' \vee s \approx t$.

(b) $s \approx t$ is strictly maximal in C.

(c) $s \succ t$.

(d) C is false in R_C.

(e) C' is false in $R_C \cup \{s \to t\}$.

(f) s is irreducible w. r. t. R_C.

(g) no negative literal is selected in C'

In this case, C is called productive. Otherwise $E_C = \emptyset$.

Finally, $R_\infty = \bigcup_{D \in G_\Sigma(N)} E_D$.

Superposition: Refutational Completeness

Lemma 6.5:

If $E_C = \{s \to t\}$ and $E_D = \{u \to v\}$, then $s \succ u$ if and only if $C \succ_C D$.

Superposition: Refutational Completeness

Corollary 6.6:
The rewrite systems R_C and R_∞ are convergent.

Proof:
Obviously, $s \succ t$ for all rules $s \to t$ in R_C and R_∞.

Furthermore, it is easy to check that there are no critical pairs between any two rules: Assume that there are rules $u \to v$ in E_D and $s \to t$ in E_C such that u is a subterm of s. As $\succ$ is a reduction ordering that is total on ground terms, we get $u \prec s$ and therefore $D \prec_C C$ and $E_D \subseteq R_C$. But then s would be reducible by R_C, contradicting condition (f). □

Superposition: Refutational Completeness

Lemma 6.7:
If $D \preceq_C C$ and $E_C = \{s \to t\}$, then $s \succ u$ for every term u occurring in a negative literal in D and $s \succeq v$ for every term v occurring in a positive literal in D.

Superposition: Refutational Completeness

Corollary 6.8:
If $D \in G_\Sigma(N)$ is true in R_D, then D is true in R_∞ and R_C for all $C \succ_C D$.

Proof:
If a positive literal of D is true in R_D, then this is obvious.

Otherwise, some negative literal $s \not\approx t$ of D must be true in R_D, hence $s \not\downarrow_{R_D} t$. As the rules in $R_\infty \setminus R_D$ have left-hand sides that are larger than s and t, they cannot be used in a rewrite proof of $s \downarrow t$, hence $s \not\downarrow_{R_C} t$ and $s \not\downarrow_{R_\infty} t$. □

Superposition: Refutational Completeness

Corollary 6.9:
If $D = D' \vee u \approx v$ is productive, then D' is false and D is true in R_∞ and R_C for all $C \succ_C D$.

Proof:
Obviously, D is true in R_∞ and R_C for all $C \succ_C D$.

Since all negative literals of D' are false in R_D, it is clear that they are false in R_∞ and R_C. For the positive literals $u' \approx v'$ of D', condition (e) ensures that they are false in $R_D \cup \{u \to v\}$. Since $u' \preceq u$ and $v' \preceq u$ and all rules in $R_\infty \setminus R_D$ have left-hand sides that are larger than u, these rules cannot be used in a rewrite proof of $u' \downarrow v'$, hence $u' \not\downarrow_{R_C} v'$ and $u' \not\downarrow_{R_\infty} v'$. $\square$

Superposition: Refutational Completeness

Lemma 6.10 ("Lifting Lemma"):
Let C be a clause and let θ be a substitution such that $C\theta$ is ground. Then every equality resolution or equality factoring inference from $C\theta$ is a ground instance of an inference from C.

Proof:
Exercise. □

Superposition: Refutational Completeness

Lemma 6.11 ("Lifting Lemma"):
Let $D = D' \vee u \approx v$ and $C = C' \vee [\neg]\, s \approx t$ be two clauses (without common variables) and let θ be a substitution such that $D\theta$ and $C\theta$ are ground.

If there is a superposition inference between $D\theta$ and $C\theta$ where $u\theta$ and some subterm of $s\theta$ are overlapped, and $u\theta$ does not occur in $s\theta$ at or below a variable position of s, then the inference is a ground instance of a superposition inference from D and C.

Proof:
Exercise. □

Superposition: Refutational Completeness

Theorem 6.12 ("Model Construction"):
Let N be a set of clauses that is saturated up to redundancy and does not contain the empty clause. Then we have for every ground clause $C\theta \in G_\Sigma(N)$:

(i) $E_{C\theta} = \emptyset$ if and only if $C\theta$ is true in $R_{C\theta}$.

(ii) If $C\theta$ is redundant w. r. t. $G_\Sigma(N)$, then it is true in $R_{C\theta}$.

(iii) $C\theta$ is true in R_∞ and in R_D for every $D \in G_\Sigma(N)$ with $D \succ_C C\theta$.

Superposition: Refutational Completeness

A Σ-interpretation $\mathcal{A}$ is called term-generated, if for every $b \in U_{\mathcal{A}}$ there is a ground term $t \in \mathsf{T}_{\Sigma}(\emptyset)$ such that $b = \mathcal{A}(\beta)(t)$.

Superposition: Refutational Completeness

Lemma 6.13:
Let N be a set of (universally quantified) Σ-clauses and let $\mathcal{A}$ be a term-generated Σ-interpretation. Then $\mathcal{A}$ is a model of $G_\Sigma(N)$ if and only if it is a model of N.

Proof:
($\Rightarrow$): Let $\mathcal{A} \models G_\Sigma(N)$; let $(\forall \vec{x} C) \in N$.
Then $\mathcal{A} \models \forall \vec{x} C$ iff $\mathcal{A}(\gamma[x_i \mapsto a_i])(C) = 1$ for all γ and a_i.
Choose ground terms t_i such that $\mathcal{A}(\gamma)(t_i) = a_i$; define θ such that $x_i\theta = t_i$, then $\mathcal{A}(\gamma[x_i \mapsto a_i])(C) = \mathcal{A}(\gamma \circ \theta)(C) = \mathcal{A}(\gamma)(C\theta) = 1$ since $C\theta \in G_\Sigma(N)$.

($\Leftarrow$): Let $\mathcal{A}$ be a model of N; let $C \in N$ and $C\theta \in G_\Sigma(N)$.
Then $\mathcal{A}(\gamma)(C\theta) = \mathcal{A}(\gamma \circ \theta)(C) = 1$ since $\mathcal{A} \models N$. □

Superposition: Refutational Completeness

Theorem 6.14 (Refutational Completeness: Static View):
Let N be a set of clauses that is saturated up to redundancy. Then N has a model if and only if N does not contain the empty clause.

Proof:
If $\bot \in N$, then obviously N does not have a model.
If $\bot \notin N$, then the interpretation R_∞ (that is, $\mathrm{T}_\Sigma(\emptyset)/R_\infty$) is a model of all ground instances in $G_\Sigma(N)$ according to part (iii) of the model construction theorem.
As $\mathrm{T}_\Sigma(\emptyset)/R_\infty$ is term generated, it is a model of N. □

Superposition: Refutational Completeness

So far, we have considered only inference rules that add new clauses to the current set of clauses (corresponding to the *Deduce* rule of Knuth-Bendix Completion).

In other words, we have derivations of the form $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$, where each N_{i+1} is obtained from N_i by adding the consequence of some inference from clauses in N_i.

Under which circumstances are we allowed to delete (or simplify) a clause during the derivation?

Superposition: Refutational Completeness

A run of the superposition calculus is a sequence
$N_0 \vdash N_1 \vdash N_2 \vdash \ldots$, such that
(i) $N_i \models N_{i+1}$, 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, $N_\infty = \bigcup_{i \geq 0} N_i$ and $N_* = \bigcup_{i \geq 0} \bigcap_{j \geq i} N_j$.
The set N_* of all persistent clauses is called the limit of the run.

Superposition: Refutational Completeness

Lemma 6.15:

If $N \subseteq N'$, then $\mathit{Red}(N) \subseteq \mathit{Red}(N')$.

Proof:

Obvious. $\square$

Superposition: Refutational Completeness

Lemma 6.16:
If $N' \subseteq \mathit{Red}(N)$, then $\mathit{Red}(N) \subseteq \mathit{Red}(N \setminus N')$.

Proof:
Follows from the compactness of first-order logic and the well-foundedness of the multiset extension of the clause ordering.

$\square$

Superposition: Refutational Completeness

Lemma 6.17:

Let $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$ be a run.

Then $\mathit{Red}(N_i) \subseteq \mathit{Red}(N_\infty)$ and $\mathit{Red}(N_i) \subseteq \mathit{Red}(N_*)$ for every i.

Proof:

Exercise. □

Superposition: Refutational Completeness

Corollary 6.18:
$N_i \subseteq N_* \cup Red(N_*)$ 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_*. $\square$

Superposition: Refutational Completeness

A run is called fair, if the conclusion of every inference from clauses in $N_* \setminus \mathit{Red}(N_*)$ is contained in some $N_i \cup \mathit{Red}(N_i)$.

Lemma 6.19:
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 $N_i \cup \mathit{Red}(N_i)$, and therefore contained in $N_* \cup \mathit{Red}(N_*)$.
Hence N_* is saturated up to redundancy. $\square$

Superposition: Refutational Completeness

Theorem 6.20 (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 term-generated 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 $N_0 \models N_*$. $\square$

Superposition: Extensions

Extensions and improvements:

simplification techniques,

selection functions (when, what),

redundancy for inferences,

constraint reasoning,

decidable first-order fragments.

Theory Reasoning

Superposition vs. resolution + equality axioms:

specialized inference rules,
thus no inferences with theory axioms,

computation modulo symmetry,

stronger ordering restrictions,

no variable overlaps,

stronger redundancy criterion.

Theory Reasoning

Similar techniques can be used for other theories:

transitive relations,

dense total orderings without endpoints,

commutativity,

associativity and commutativity,

abelian monoids,

abelian groups,

divisible torsion-free abelian groups.

Part 7: Outlook

Further topics in automated reasoning.

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.

Satisfiability Modulo Theories (SMT)

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.

Satisfiability Modulo Theories (SMT)

New CDCL rules:

T-Propagate:

$$M \parallel N \ \Rightarrow_{\text{CDCL(T)}} \ M\,L \parallel N$$

if $M \models_T L$ where L is undefined in M and L or $\overline{L}$ occurs in N.

T-Learn:

$$M \parallel N \ \Rightarrow_{\text{CDCL(T)}} \ M \parallel N \cup \{C\}$$

if $N \models_T C$ and each atom of C occurs in N or M.

Satisfiability Modulo Theories (SMT)

T-Backjump:

$$M\, L^{\mathrm{d}}\, M' \parallel N \cup \{C\} \;\Rightarrow_{\mathrm{CDCL(T)}}\; M\, L' \parallel N \cup \{C\}$$

if $M\, L^{\mathrm{d}}\, M' \models \neg C$

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 $M\, L^{\mathrm{d}}\, M'$.

7.2 Sorted Logics

So far, we have considered only unsorted first-order logic.

In practice, one often considers many-sorted logics:

$\mathit{read}/2$ becomes $\mathit{read} : \mathit{array} \times \mathit{nat} \rightarrow \mathit{data}$.

$\mathit{write}/3$ becomes $\mathit{write} : \mathit{array} \times \mathit{nat} \times \mathit{data} \rightarrow \mathit{array}$.

Variables: $x : \mathit{data}$

Only one declaration per function/predicate/variable symbol.

All terms, atoms, substitutions must be well-sorted.

Sorted Logics

Algebras:

Instead of universe $U_{\mathcal{A}}$, one set per sort: $\mathit{array}_{\mathcal{A}}$, $\mathit{nat}_{\mathcal{A}}$.

Interpretations of function and predicate symbols correspond to their declarations:

$\mathit{read}_{\mathcal{A}} : \mathit{array}_{\mathcal{A}} \times \mathit{nat}_{\mathcal{A}} \to \mathit{data}_{\mathcal{A}}$

Sorted Logics

Proof theory, calculi, etc.:

Essentially as in the unsorted case.

More difficult:

Subsorts

Overloading

Better treated via relativization:

$\forall x_S\ \phi \Rightarrow \forall y\ S(y) \rightarrow \phi\{x_S \mapsto y\}$

7.3 Splitting

Tableau-like rule within resolution to eliminate variable-disjoint (positive) disjunctions:

$$\frac{N \cup \{C_1 \vee C_2\}}{N \cup \{C_1\} \quad | \quad N \cup \{C_2\}}$$

if $var(C_1) \cap var(C_2) = \emptyset$.

Split clauses are smaller and more likely to be usable for simplification.

Splitting tree is explored using intelligent backtracking.

7.4 Integrating Theories into Superposition

Certain kinds of theories/axioms are

important in practice,

but difficult for theorem provers.

So far important case: equality

but also: transitivity, arithmetic...

Integrating Theories into Superposition

Idea: Combine Superposition and Constraint Reasoning.

Superposition Left Modulo Theories:

$$\frac{\Lambda_1 \parallel C_1 \vee t \approx t' \qquad \Lambda_2 \parallel C_2 \vee s[u] \not\approx s'}{(\Lambda_1, \Lambda_2 \parallel C_1 \vee C_2 \vee s[t'] \not\approx s')\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$,

$\ldots$

Advertisements

Interested in Bachelor/Master/PhD Thesis?

Automated Reasoning

contact Christoph Weidenbach

(MPI-INF, MPI-SWS Building, 6th floor)

Hybrid System Verification

contact Uwe Waldmann

Arithmetic Reasoning (Quantifier Elimination)

contact Thomas Sturm

Advertisements

Next semester:

Automated Reasoning II

Content: Integration of Theories (Arithmetic)

Lecture: Block Course

Tutorials: TBA