Automated Reasoning in First-Order Logic*

Peter Baumgartner†

2010

Automated Reasoning

- An application-oriented subfield of logic in computer science and artificial intelligence
- About algorithms and their implementation on computer for reasoning with mathematical logic formulas
- Considers a variety of logics and reasoning tasks
- Applications in logics in computer science

  Program verification, dynamic properties of reactive systems, databases
- Applications in logic-based artificial intelligence

  Mathematical theorem proving, planning, diagnosis, knowledge representation (description logics), logic programming, constraint solving

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

†Thanks to Christoph Weidenbach (MPI, Germany) who provided the largest part of this document.

Automated Reasoning in First-Order Logic

... First-Order Logic: Can express (mathematical) structures, e.g. groups

$$\forall x\ 1 \cdot x = x \qquad \forall x\ x \cdot 1 = x \tag{N}$$
$$\forall x\ x^{-1} \cdot x = 1 \qquad \forall x\ x \cdot x^{-1} = 1 \tag{I}$$
$$\forall x, y, z\ (x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{A}$$

... Reasoning ...:

- Object level: It follows $\forall x\ (x \cdot x) = 1 \rightarrow \forall x, y\ x \cdot y = y \cdot x$
- Meta-level: the word problem for groups is decidable

Automated ...: Computer program to provide the above conclusions *automatically*

Application Example: Compiler Validation

Prove equivalence of source (left) and target (right) program

```
1:  y := 1                  1: y := 1
2:  if z = x*x*x            2: R1 := x*x
3:    then y := x*x + y     3: R2 := R1*x
4:  endif                   4: jmpNE(z,R2,6)
                            5: y := R1+1
```

Formal proof obligation (indexes refer to line numbers above)

From $\quad y_1 = 1 \ \wedge\ z_0 = x_0 * x_0 * x_0 \ \wedge\ y_3 = x_0 * x_0 + y_1$

and $\quad y'_1 = 1 \ \wedge\ R1_2 = x'_0 * x'_0 \ \wedge\ R2_3 = R1_2 * x'_0 \ \wedge\ z'_0 = R2_3$
$\wedge\ y'_5 = R1_2 + 1 \ \wedge\ x_0 = x'_0 \ \wedge\ y_0 = y'_0 \ \wedge\ z_0 = z'_0$

it follows $\quad y_3 = y'_5$

These Lectures

- Automated theorem proving in *first-order* logic
  (The slides below on propositional logic are for reference only)
- Standard concepts

  Normal forms of logical formulas, unification, the *modern* resolution calculus

- Standard results

  Soundness and completeness of the resolution calculus with redundancy criteria

- Provide a basis for further studies
- "How to build a theorem prover"

"How to Build a Theorem Prover"

1. Fix an *input language* for mathematical formulas.
2. Fix a *semantics* to define what the formulas mean. Will be always "classical" here.
3. Determine the desired *services* from the theorem prover: the questions we would like the prover be able to answer.
4. Design a *calculus* for the logic and the services.

   Calculus: high-level description of the "logical analysis" algorithm. This includes improvements for search space pruning.
5. Prove the calculus is *correct* (sound and complete) wrt. the logic and the services, if possible.
6. Design and implement a *proof procedure* for the calculus.

1 Propositional Logic

Propositional logic

- Logic of truth values
- Decidable (but NP-complete)
- Can be used to describe functions over a finite domain
- Implemented automated reasoning systems for propositional logic ("SAT-solvers") have many applications in, e.g., systems verification, diagnosis, planning, constraint solving.

1.1 Syntax

- Propositional variables
- Logical symbols
  $\Rightarrow$ Boolean combinations

Propositional Variables

Let Π be a set of *propositional variables*.

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

Propositional Formulas

F_Π is the set of propositional formulas over Π defined as follows:

$$
\begin{array}{rcll}
F, G, H & ::= & \bot & \text{(falsum)} \\
& | & \top & \text{(verum)} \\
& | & P, \quad P \in \Pi & \text{(atomic formula)} \\
& | & \neg F & \text{(negation)} \\
& | & (F \wedge G) & \text{(conjunction)} \\
& | & (F \vee G) & \text{(disjunction)} \\
& | & (F \rightarrow G) & \text{(implication)} \\
& | & (F \leftrightarrow G) & \text{(equivalence)}
\end{array}
$$

Notational Conventions

- We omit brackets according to the following rules:
  - $\neg \quad >_p \quad \vee \quad >_p \quad \wedge \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$
    (binding precedences)
  - $\vee$ and $\wedge$ are associative
  - $\rightarrow$ is right-associative,
    i. e., $F \rightarrow G \rightarrow H$ means $F \rightarrow (G \rightarrow H)$.

1.2 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 Π-*valuation* is a map

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

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

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

Given a Π-valuation $\mathcal{A}$, the function $\mathcal{A}^* : \Sigma$-formulas $\to \{0, 1\}$ is defined inductively over the structure of F as follows:

$$\begin{aligned}
\mathcal{A}^*(\bot) &= 0 \\
\mathcal{A}^*(\top) &= 1 \\
\mathcal{A}^*(P) &= \mathcal{A}(P) \\
\mathcal{A}^*(\neg F) &= \mathsf{B}_\neg(\mathcal{A}^*(F)) \\
\mathcal{A}^*(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}^*(F), \mathcal{A}^*(G))
\end{aligned}$$

where B_ρ is the Boolean function associated with ρ defined by the usual truth table.

For simplicity, we write $\mathcal{A}$ instead of $\mathcal{A}^*$.

We also write ρ instead of B_ρ, i. e., we use the same notation for a logical symbol and for its meaning (but remember that formally these are different things.)

1.3 Models, Validity, and Satisfiability

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

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

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

$$\models F \;:\Leftrightarrow\; \mathcal{A} \models F \text{ for all } \Pi\text{-valuations } \mathcal{A}$$

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

Entailment and Equivalence

F *entails* (*implies*) G (or G *is a consequence of* F), written $F \models G$, if for all Π-valuations $\mathcal{A}$, whenever $\mathcal{A} \models F$ then $\mathcal{A} \models G$.

F and G are called *equivalent*, written $F \models\!\models G$, if for all Π-valuations $\mathcal{A}$ we have $\mathcal{A} \models F \Leftrightarrow \mathcal{A} \models G$.

Proposition 1.1 *$F \models G$ if and only if $\models (F \to G)$.(Proof follows)*

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

($\Leftarrow$) Suppose that F does not entail G. Then there exists a Π-valuation $\mathcal{A}$ such that $\mathcal{A} \models F$, but not $\mathcal{A} \models G$. Consequently, $\mathcal{A}(F \to G) = \mathsf{B}_{\to}(\mathcal{A}(F), \mathcal{A}(G)) = \mathsf{B}_{\to}(1, 0) = 0$, so $(F \to G)$ does not hold in $\mathcal{A}$. □

Proposition 1.2 *$F \models\!\models G$ if and only if $\models (F \leftrightarrow G)$.*

Proof. Follows from Prop. 1.1. □

Extension to sets of formulas N in the "natural way":

$N \models F$ if for all Π-valuations $\mathcal{A}$:
if $\mathcal{A} \models G$ for all $G \in N$, then $\mathcal{A} \models F$.

Validity vs. Unsatisfiability

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

Proposition 1.3 *F is valid if and only if $\neg F$ is unsatisfiable.(Proof follows)*

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

($\Leftarrow$) Analogously. □

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

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

Proposition 1.4 *$N \models F$ if and only if $N \cup \{\neg F\}$ is unsatisfiable.*

Checking Unsatisfiability

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

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

So the satisfiability problem is clearly deciadable (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. See lecture by Alwen Tiu on DPLL.

Substitution Theorem

Proposition 1.5 *Let F and G be equivalent formulas, let H be a formula in which F occurs as a subformula.*

Then H is equivalent to H' where H' is obtained from H by replacing the occurrence of the subformula F by G. (Notation: $H = H[F]$, $H' = H[G]$. Proof follows)

Proof. The proof proceeds by induction over the formula structure of H.

Each of the formulas $\bot$, $\top$, and P for $P \in \Pi$ contains only one subformula, namely itself. Hence, if $H = H[F]$ equals $\bot$, $\top$, or P, then $H = F$, $H' = G$, and H and H' are equivalent by assumption.

If $H = H_1 \wedge H_2$, then either F equals H (this case is treated as above), or F is a subformula of H_1 or H_2. Without loss of generality, assume that F is a subformula of H_1, so $H = H_1[F] \wedge H_2$. By the induction hypothesis, $H_1[F]$ and $H_1[G]$ are equivalent. Hence, for every valuation $\mathcal{A}$, $\mathcal{A}(H') = \mathcal{A}(H_1[G] \wedge H_2) = \mathcal{A}(H_1[G]) \wedge \mathcal{A}(H_2) = \mathcal{A}(H_1[F]) \wedge \mathcal{A}(H_2) = \mathcal{A}(H_1[F] \wedge H_2) = \mathcal{A}(H)$.

The other boolean connectives are handled analogously. □

Some Important Equivalences

Proposition 1.6 *The following equivalences are valid for all formulas F, G, H:*

$$
\begin{aligned}
(F \wedge F) &\leftrightarrow F \\
(F \vee F) &\leftrightarrow F && \textit{(Idempotency)} \\
(F \wedge G) &\leftrightarrow (G \wedge F) \\
(F \vee G) &\leftrightarrow (G \vee F) && \textit{(Commutativity)} \\
(F \wedge (G \wedge H)) &\leftrightarrow ((F \wedge G) \wedge H) \\
(F \vee (G \vee H)) &\leftrightarrow ((F \vee G) \vee H) && \textit{(Associativity)} \\
(F \wedge (G \vee H)) &\leftrightarrow ((F \wedge G) \vee (F \wedge H)) \\
(F \vee (G \wedge H)) &\leftrightarrow ((F \vee G) \wedge (F \vee H)) && \textit{(Distributivity)}
\end{aligned}
$$

$$(F \wedge (F \vee G)) \leftrightarrow F$$
$$(F \vee (F \wedge G)) \leftrightarrow F \qquad \textit{(Absorption)}$$
$$(\neg\neg F) \leftrightarrow F \qquad \textit{(Double Negation)}$$
$$\neg(F \wedge G) \leftrightarrow (\neg F \vee \neg G)$$
$$\neg(F \vee G) \leftrightarrow (\neg F \wedge \neg G) \qquad \textit{(De Morgan's Laws)}$$
$(F \wedge G) \leftrightarrow F$, *if* G *is a tautology*
$(F \vee G) \leftrightarrow \top$, *if* G *is a tautology*
$(F \wedge G) \leftrightarrow \bot$, *if* G *is unsatisfiable*
$(F \vee G) \leftrightarrow F$, *if* G *is unsatisfiable* *(Tautology Laws)*

$$(F \leftrightarrow G) \leftrightarrow ((F \rightarrow G) \wedge (G \rightarrow F)) \qquad \textit{(Equivalence)}$$
$$(F \rightarrow G) \leftrightarrow (\neg F \vee G) \qquad \textit{(Implication)}$$

1.4 Normal Forms

We define *conjunctions* of formulas as follows:

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

$\bigwedge_{i=1}^{1} F_i = F_1$.

$\bigwedge_{i=1}^{n+1} F_i = \bigwedge_{i=1}^{n} F_i \wedge F_{n+1}$.

and analogously *disjunctions:*

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

$\bigvee_{i=1}^{1} F_i = F_1$.

$\bigvee_{i=1}^{n+1} F_i = \bigvee_{i=1}^{n} F_i \vee F_{n+1}$.

Literals and Clauses

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

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

CNF and DNF

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

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

Warning: definitions in the literature differ:

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

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

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

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

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

Conversion to CNF/DNF

Proposition 1.7 *For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).*

Proof. We consider the case of CNF.

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

Step 1: Eliminate equivalences:

$$(F \leftrightarrow G) \;\Rightarrow_K\; (F \rightarrow G) \wedge (G \rightarrow F)$$

Step 2: Eliminate implications:

$$(F \rightarrow G) \;\Rightarrow_K\; (\neg F \vee G)$$

Step 3: Push negations downward:

$$\neg(F \vee G) \;\Rightarrow_K\; (\neg F \wedge \neg G)$$
$$\neg(F \wedge G) \;\Rightarrow_K\; (\neg F \vee \neg G)$$

Step 4: Eliminate multiple negations:

$$\neg\neg F \;\Rightarrow_K\; F$$

Step 5: Push disjunctions downward:

$$(F \wedge G) \vee H \;\Rightarrow_K\; (F \vee H) \wedge (G \vee H)$$

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

$$\begin{aligned}
(F \wedge \top) &\Rightarrow_K F \\
(F \wedge \bot) &\Rightarrow_K \bot \\
(F \vee \top) &\Rightarrow_K \top \\
(F \vee \bot) &\Rightarrow_K F \\
\neg\bot &\Rightarrow_K \top \\
\neg\top &\Rightarrow_K \bot
\end{aligned}$$

Proving termination is easy for most of the steps; only step 3 and step 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.

Satisfiability-preserving Transformations

The goal

"find a formula G in CNF such that $F \models\!\!\mid G$"

is unpractical.

But if we relax the requirement to

"find a formula G in CNF such that $F \models \bot \Leftrightarrow G \models \bot$"

we can get an efficient transformation.

Idea: A formula $F[F']$ is satisfiable if and only if $F[P] \wedge (P \leftrightarrow F')$ is satisfiable (where P is a new propositional variable that works as an abbreviation for F').

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 $P \leftrightarrow F'$ gives rise to at most one application of the distributivity law).

2 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*.

2.1 Syntax

Syntax:

- non-logical symbols (domain-specific)
  $\Rightarrow$ terms, atomic formulas
- logical symbols (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 $\mathsf{arity}(f) = n$,
- Π is a set of *predicate symbols* p with arity $m \geq 0$, written $\mathsf{arity}(p) = m$.

If $n = 0$ then f is also called a *constant (symbol)*.
If $m = 0$ then p is also called a *propositional variable*.
We use letters P, Q, R, S, to denote propositional variables.

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 for (the denotation of) *variables*.

Terms

Terms over Σ (resp., Σ-terms) are formed according to these syntactic rules:

$$\begin{array}{lll} s,t,u,v & ::= & x \quad , x \in X \qquad\qquad\qquad\qquad \text{(variable)} \\ & \mid & f(s_1,...,s_n) \quad , f \in \Omega, \mathsf{arity}(f) = n \quad \text{(functional term)} \end{array}$$

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.

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:

$$\begin{array}{rcll} A, B & ::= & p(s_1, ..., s_m) & , p \in \Pi, \mathsf{arity}(p) = m \\ \left[\right. & | & (s \approx t) & \text{(equation)} \qquad \left. \right] \end{array}$$

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, but deductive systems where equality is treated specifically can be much more efficient.

Literals

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

Clauses

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

General First-Order Formulas

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

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

Notational Conventions

We omit brackets according to the following rules:

- $\neg \quad >_p \quad \vee \quad >_p \quad \wedge \quad >_p \quad \to \quad >_p \quad \leftrightarrow$ (binding precedences)
- $\vee$ and $\wedge$ are associative and commutative
- $\to$ is right-associative

$Qx_1, \ldots, x_n\, F$ abbreviates $Qx_1 \ldots Qx_n\, F$.

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

Examples:

$s + t * u$	for	$+(s, *(t, u))$
$s * u \leq t + v$	for	$\leq (*(s, u), +(t, v))$
$-s$	for	$-(s)$
0	for	$0()$

Example: Peano Arithmetic

$\Sigma_{PA} \;=\; (\Omega_{PA},\ \Pi_{PA})$
$\Omega_{PA} \;=\; \{0/0,\ +/2,\ */2,\ s/1\}$
$\Pi_{PA} \;=\; \{\leq /2,\ < /2\}$
$+, *, <, \leq$ infix; $* \ >_p \ + \ >_p \ < \ >_p \ \leq$

Examples of formulas over this signature are:

$\forall x, y(x \leq y \leftrightarrow \exists z(x + z \approx y))$
$\exists x \forall y(x + y \approx y)$
$\forall x, y(x * s(y) \approx x * y + x)$
$\forall x, y(s(x) \approx s(y) \to x \approx y)$
$\forall x \exists y(x < y \wedge \neg \exists z(x < z \wedge z < y))$

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.

Bound and Free Variables

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

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

Formulas without variables are called *ground*.

Example:

$$\overbrace{\forall y \quad (\forall \overbrace{x \quad p(x)}^{\text{scope}} \quad \rightarrow \quad q(x,y))}^{\text{scope}}$$

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

$$\sigma : X \rightarrow \mathrm{T}_\Sigma(X)$$

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 are often written as $[s_1/x_1, \ldots, s_n/x_n]$, with x_i pairwise distinct, and then denote the mapping

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

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 F by structural induction over the syntactic structure of t or F by the equations depicted on the next page.

In the presence of quantification it is surprisingly complex: We need to make sure that the (free) variables in the codomain of 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.

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

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

It is instructive to evaluate $(\forall x\, p(x,y))\sigma$, where $\sigma = [a/x,\ x/y]$.

Structural Induction on Terms

Proposition 2.1 *Let q be a property of the elements of $T_\Sigma(X)$, the Σ-terms over X. Then, q holds for* all *$t \in T_\Sigma(X)$, whenever one can prove the following two properties:*

1. *(*base case*)*
   q holds for every $x \in X$.
2. *(*step case*)*
   for every $f \in \Omega$ with $\mathsf{arity}(f) = n$,
   for all terms $s_1, \ldots, s_n \in T_\Sigma(X)$,
   if q holds for every $s_1, \ldots, s_n$ then q also holds for $f(s_1, \ldots, s_n)$.

Analogously: structural induction on formulas

2.2 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

$$\mathcal{A} = (U_\mathcal{A},\ (f_\mathcal{A} : U^n \to U)_{f\in\Omega},\ (p_\mathcal{A} \subseteq U_\mathcal{A}^m)_{p\in\Pi})$$

where $\mathsf{arity}(f) = n$, $\mathsf{arity}(p) = m$, $U_\mathcal{A} \neq \emptyset$ 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

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

as follows:

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

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:

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

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.

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

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

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

2.3 Models, Validity, and Satisfiability

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

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

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

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

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

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

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

Entailment and Equivalence

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

F and G are called *equivalent*, written $F \models\!\!\models G$, if for all $\mathcal{A} \in \Sigma\text{-}\mathsf{Alg}$ und $\beta \in X \to U_{\mathcal{A}}$ we have $\mathcal{A}, \beta \models F \quad \Leftrightarrow \quad \mathcal{A}, \beta \models G$.

Proposition 2.2 *F entails G iff $(F \to G)$ is valid*

Proposition 2.3 *F and G are equivalent iff $(F \leftrightarrow G)$ is valid.*

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

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma\text{-}\mathsf{Alg}$ and $\beta \in X \to U_{\mathcal{A}}$: if $\mathcal{A}, \beta \models G$, for all $G \in N$, then $\mathcal{A}, \beta \models F$.

Validity vs. Unsatisfiability

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

Proposition 2.4 *Let F and G be formulas, let N be a set of formulas. Then*

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

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

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

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

Algorithmic Problems

This is a more comprehensive list of services an automated reasoning system might provide:

Validity(F): $\models F$?

Satisfiability(F): F satisfiable?

Entailment(F,G): does F entail G?

Model(A,F): $A \models F$?

Solve(A,F): find an assignment β such that $A, \beta \models F$.

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

Abduce(F): find G with "certain properties" such that $G \models F$.

2.4 Normal Forms and Skolemization (Traditional)

Study of normal forms motivated by

- reduction of logical concepts,
- efficient data structures for theorem proving.

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

Prenex Normal Form

Prenex formulas have the form

$$Q_1 x_1 \dots Q_n x_n \; F,$$

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

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

$$\begin{array}{rcll} (F \leftrightarrow G) & \Rightarrow_P & (F \to G) \wedge (G \to F) & \\ \neg Q x F & \Rightarrow_P & \overline{Q} x \neg F & (\neg Q) \\ (Q x F \;\rho\; G) & \Rightarrow_P & Qy(F[y/x] \;\rho\; G),\ y \text{ fresh},\ \rho \in \{\wedge, \vee\} & \\ (Q x F \to G) & \Rightarrow_P & \overline{Q} y(F[y/x] \to G),\ y \text{ fresh} & \\ (F \;\rho\; Q x G) & \Rightarrow_P & Qy(F \;\rho\; G[y/x]),\ y \text{ fresh},\ \rho \in \{\wedge, \vee, \to\} & \end{array}$$

Here $\overline{Q}$ 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):

$$\forall x_1, \dots, x_n \exists y F \;\Rightarrow_S\; \forall x_1, \dots, x_n F[f(x_1, \dots, x_n)/y]$$

where f, where $\mathsf{arity}(f) = n$, is a new function symbol (*Skolem function*).

Together: $F \stackrel{*}{\Rightarrow}_P \underbrace{G}_{\text{prenex}} \stackrel{*}{\Rightarrow}_S \underbrace{H}_{\text{prenex, no } \exists}$

Theorem 2.5 *Let F, G, and H as defined above and closed. Then*

(i) F and G are equivalent.

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

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

Clausal Normal Form (Conjunctive Normal Form)

$$
\begin{array}{rcl}
(F \leftrightarrow G) & \Rightarrow_K & (F \to G) \wedge (G \to F) \\
(F \to G) & \Rightarrow_K & (\neg F \vee G) \\
\neg(F \vee G) & \Rightarrow_K & (\neg F \wedge \neg G) \\
\neg(F \wedge G) & \Rightarrow_K & (\neg F \vee \neg G) \\
\neg\neg F & \Rightarrow_K & F \\
(F \wedge G) \vee H & \Rightarrow_K & (F \vee H) \wedge (G \vee H) \\
(F \wedge \top) & \Rightarrow_K & F \\
(F \wedge \bot) & \Rightarrow_K & \bot \\
(F \vee \top) & \Rightarrow_K & \top \\
(F \vee \bot) & \Rightarrow_K & F
\end{array}
$$

These rules are to be applied modulo associativity and commutativity of $\wedge$ and $\vee$. The first five rules, plus the rule $(\neg Q)$, compute the *negation normal form* (NNF) of a formula.

The Complete Picture

$$
\begin{array}{rcll}
F & \Rightarrow_P^* & Q_1 y_1 \ldots Q_n y_n \; G & (G \text{ quantifier-free}) \\
 & \Rightarrow_S^* & \forall x_1, \ldots, x_m \; H & (m \leq n,\ H \text{ quantifier-free}) \\
 & \Rightarrow_K^* & \underbrace{\underbrace{\forall x_1, \ldots, x_m}_{\text{leave out}} \bigwedge_{i=1}^{k} \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}}_{F'} &
\end{array}
$$

$N = \{C_1, \ldots, C_k\}$ is called the *clausal (normal) form* (CNF) of F.
Note: the variables in the clauses are implicitly universally quantified.

Theorem 2.6 *Let F be closed. Then $F' \models F$. (The converse is not true in general.)*

Theorem 2.7 *Let F be closed. Then F is satisfiable iff F' is satisfiable iff N is satisfiable*

Optimization

Here is lots of room for optimization since we only can preserve satisfiability anyway:

- size of the CNF exponential when done naively;
  but see the transformations we introduced for propositional logic
- want small arity of Skolem functions (not discussed here)

2.5 Herbrand Interpretations

From now an we shall consider PL without equality. Ω shall contains at least one constant symbol.

A *Herbrand interpretation* (over Σ) is a Σ-algebra $\mathcal{A}$ such that

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

$$f_{\mathcal{A}}(\triangle, \ldots, \triangle) = \quad \begin{array}{c} f \\ \triangle \ \cdots \ \triangle \end{array}$$

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

Proposition 2.8 *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.

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 F, if $I \models F$.

Theorem 2.9 (Herbrand) *Let N be a set of Σ-clauses.*

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

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

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

Example of a G_Σ

For Σ_{Pres} one obtains for

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

the following ground instances:

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

2.6 Inference Systems and Proofs

Inference systems Γ (proof calculi) are sets of tuples

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

called *inferences* or *inference rules*, and written

$$\frac{\overbrace{F_1 \ \ldots \ F_n}^{\text{premises}}}{\underbrace{F_{n+1}}_{\text{conclusion}}}\ .$$

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

Proofs

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

(i) $F_k = F$,

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

$$\frac{F_{i_1} \ \ldots \ F_{i_{n_i}}}{F_i}$$

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

Soundness and Completeness

Provability $\vdash_\Gamma$ of F from N in Γ: $N \vdash_\Gamma F \ :\Leftrightarrow$ there exists a proof Γ of F from N.

Γ is called *sound* $:\Leftrightarrow$

$$\frac{F_1 \ \ldots \ F_n}{F} \in \Gamma \quad \Rightarrow \quad F_1, \ldots, F_n \models F$$

Γ is called *complete* $:\Leftrightarrow$

$$N \models F \quad \Rightarrow \quad N \vdash_\Gamma F$$

Γ is called *refutationally complete* $:\Leftrightarrow$

$$N \models \bot \quad \Rightarrow \quad N \vdash_\Gamma \bot$$

Proposition 2.10

(i) Let Γ be sound. Then $N \vdash_\Gamma F \Rightarrow N \models F$

(ii) $N \vdash_\Gamma F \Rightarrow$ there exist $F_1, \ldots, F_n \in N$ s.t. $F_1, \ldots, F_n \vdash_\Gamma F$ (resembles compactness).

Proofs as Trees

markings	$\widehat{=}$	formulas		
leaves	$\widehat{=}$	assumptions and axioms		
other nodes	$\widehat{=}$	inferences: conclusion	$\widehat{=}$	ancestor
		premises	$\widehat{=}$	direct descendants

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

2.7 Propositional Resolution

We observe that propositional clauses and ground clauses are the same concept.

In this section we only deal with 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*

(Positive) factorisation inference rule:

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

These are *schematic inference rules*; for each substitution of the *schematic variables* C, D, and A, respectively, by ground clauses and ground atoms we obtain an inference rule.

As "$\vee$" is considered associative and commutative, we assume that A and $\neg A$ can occur anywhere in their respective clauses.

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.	$\bot$	(Res. 4. into 9.)

Resolution with Implicit Factorization RIF

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

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.	$Q(b) \vee Q(b) \vee Q(b)$	(Res. 2. into 5.)
7.	$\neg P(g(b,c))$	(Res. 6. into 3.)
8.	$\bot$	(Res. 4. into 7.)

Soundness of Resolution

Theorem 2.11 *Propositional resolution is sound.*

Proof. Let $I \in \Sigma\text{-}\mathsf{Alg}$. To be shown:

(i) for resolution: $I \models D \vee A,\ I \models C \vee \neg A \Rightarrow I \models D \vee C$

(ii) for factorization: $I \models C \vee A \vee A \Rightarrow I \models C \vee A$

(i): Assume premises are valid in I. Two cases need to be considered:
If $I \models A$, then $I \models C$, hence $I \models D \vee C$.
Otherwise, $I \models \neg A$, then $I \models D$, and again $I \models D \vee C$.
(ii): even simpler. □

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

1. $I \models L_1 \vee \ldots \vee L_n \Leftrightarrow$ there exists i: $I \models L_i$.
2. $I \models A$ or $I \models \neg A$.

This does not hold for formulas with variables!

2.8 Refutational Completeness of Resolution

How to show refutational completeness of propositional resolution:

- We have to show: $N \models \bot \;\Rightarrow\; N \vdash_{Res} \bot$, or equivalently: If $N \not\vdash_{Res} \bot$, then N has a model.
- Idea: Suppose that we have computed sufficiently many inferences (and not derived $\bot$).
- Now order the clauses in N according to some appropriate ordering, inspect the clauses in ascending order, and construct a series of Herbrand interpretations.
- The limit interpretation can be shown to be a model of N.

Multi-Sets

Let M be a set. A *multi-set* S over M is a mapping $S : M \to \mathbb{N}$. Hereby $S(m)$ specifies the number of occurrences of elements m of the base set M within the multi-set S.

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

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

$$\begin{aligned}(S_1 \cup S_2)(m) &= S_1(m) + S_2(m)\\ (S_1 \cap S_2)(m) &= \min\{S_1(m), S_2(m)\}\end{aligned}$$

A multi-set is called *finite*, if

$$|\{m \in M|\ s(m) > 0\}| < \infty,$$

for each m in M.

From now on we only consider finite multi-sets.

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

Let $(M, \succ)$ be a partial ordering. The *multi-set extension* of $\succ$ to multi-sets over M is defined by

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

Theorem 2.12
(a) $\succ_{\text{mul}}$ *is a partial ordering.*
(b) $\succ$ *well-founded* $\Rightarrow \succ_{\text{mul}}$ *well-founded.*
(c) $\succ$ *total* $\Rightarrow \succ_{\text{mul}}$ *total.*

Clause Orderings

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

2. Extend $\succ$ to an *ordering* $\succ_L$ *on ground literals:*

$$\begin{array}{lcll} [\neg]A & \succ_L & [\neg]B & \text{, if } A \succ B \\ \neg A & \succ_L & A & \end{array}$$

3. Extend $\succ_L$ to an *ordering* $\succ_C$ *on ground clauses:*
   $\succ_C = (\succ_L)_{\text{mul}}$, the multi-set extension of $\succ_L$.

   Notation: $\succ$ also for $\succ_L$ and $\succ_C$.

Example

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

$$\begin{array}{ll} & A_0 \vee A_1 \\ \prec & A_1 \vee A_2 \\ \prec & \neg A_1 \vee A_2 \\ \prec & \neg A_1 \vee A_4 \vee A_3 \\ \prec & \neg A_1 \vee \neg A_4 \vee A_3 \\ \prec & \neg A_5 \vee A_5 \end{array}$$

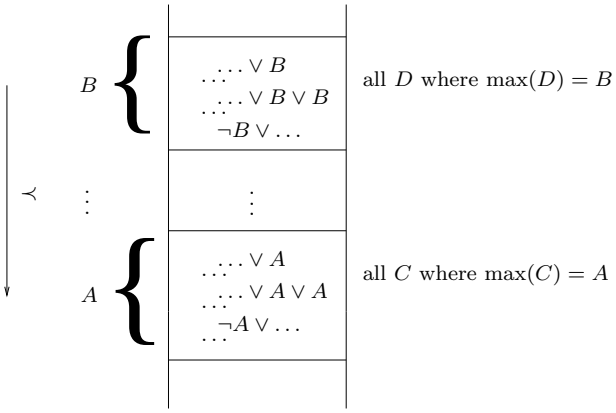
Properties of the Clause Ordering

Proposition 2.13

1. *The orderings on literals and clauses are total and well-founded.*
2. *Let C and D be clauses with $A = \max(C)$, $B = \max(D)$, where $\max(C)$ denotes the maximal atom in C.*
   (i) If $A \succ B$ then $C \succ D$.
   (ii) If $A = B$, A occurs negatively in C but only positively in D, then $C \succ D$.

Stratified Structure of Clause Sets

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



Closure of Clause Sets under Res

$$
\begin{aligned}
Res(N) &= \{C \mid C \text{ is concl. of a rule in } Res \text{ w/ premises in } N\} \\
Res^0(N) &= N \\
Res^{n+1}(N) &= Res(Res^n(N)) \cup Res^n(N), \text{ for } n \geq 0 \\
Res^*(N) &= \textstyle\bigcup_{n \geq 0} Res^n(N)
\end{aligned}
$$

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

Proposition 2.14

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

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

$$N \models \bot \;\Leftrightarrow\; \bot \in Res^*(N)$$

Construction of Interpretations

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

- "many" clauses from N are valid in I;
- *$I \models N$, if N is saturated and $\bot \notin N$.*

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

Example

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

	clauses C	I_C	Δ_C	Remarks
1	$\neg A_0$	$\emptyset$	$\emptyset$	true in I_C
2	$A_0 \vee A_1$	$\emptyset$	$\{A_1\}$	A_1 maximal
3	$A_1 \vee A_2$	$\{A_1\}$	$\emptyset$	true in I_C
4	$\neg A_1 \vee A_2$	$\{A_1\}$	$\{A_2\}$	A_2 maximal
5	$\neg A_1 \vee A_4 \vee A_3 \vee A_0$	$\{A_1, A_2\}$	$\{A_4\}$	A_4 maximal
6	$\neg A_1 \vee \neg A_4 \vee A_3$	$\{A_1, A_2, A_4\}$	$\emptyset$	A_3 not maximal; *min. counter-ex.*
7	$\neg A_1 \vee A_5$	$\{A_1, A_2, A_4\}$	$\{A_5\}$	

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

Main Ideas of the Construction

- Clauses are considered in the order given by $\prec$.
- When considering C, one already has a partial interpretation I_C (initially $I_C = \emptyset$) available.
- If C is true in the partial interpretation I_C, nothing is done. ($\Delta_C = \emptyset$).
- If C is false, one would like to change I_C such that C becomes true.

- Changes should, however, be *monotone*. One never deletes anything from I_C and the truth value of clauses smaller than C should be maintained the way it was in I_C.
- Hence, one chooses $\Delta_C = \{A\}$ if, and only if, C is false in I_C, if A occurs positively in C (*adding A will make C become true*) and if this occurrence in C is strictly maximal in the ordering on literals (*changing the truth value of A has no effect on smaller clauses*).

Resolution Reduces Counterexamples

$$\frac{\neg A_1 \vee A_4 \vee A_3 \vee A_0 \quad \neg A_1 \vee \neg A_4 \vee A_3}{\neg A_1 \vee \neg A_1 \vee A_3 \vee A_3 \vee A_0}$$

Construction of I for the extended clause set:

clauses C	I_C	Δ_C	Remarks
$\neg A_0$	$\emptyset$	$\emptyset$	
$A_0 \vee A_1$	$\emptyset$	$\{A_1\}$	
$A_1 \vee A_2$	$\{A_1\}$	$\emptyset$	
$\neg A_1 \vee A_2$	$\{A_1\}$	$\{A_2\}$	
$\neg A_1 \vee \neg A_1 \vee A_3 \vee A_3 \vee A_0$	$\{A_1, A_2\}$	$\emptyset$	A_3 occurs twice *minimal counter-ex.*
$\neg A_1 \vee A_4 \vee A_3 \vee A_0$	$\{A_1, A_2\}$	$\{A_4\}$	
$\neg A_1 \vee \neg A_4 \vee A_3$	$\{A_1, A_2, A_4\}$	$\emptyset$	counterexample
$\neg A_1 \vee A_5$	$\{A_1, A_2, A_4\}$	$\{A_5\}$	

The same I, but smaller counterexample, hence some progress was made.

Factorization Reduces Counterexamples

$$\frac{\neg A_1 \vee \neg A_1 \vee A_3 \vee A_3 \vee A_0}{\neg A_1 \vee \neg A_1 \vee A_3 \vee A_0}$$

Construction of I for the extended clause set:

clauses C	I_C	Δ_C	Remarks
$\neg A_0$	$\emptyset$	$\emptyset$	
$A_0 \vee A_1$	$\emptyset$	$\{A_1\}$	
$A_1 \vee A_2$	$\{A_1\}$	$\emptyset$	
$\neg A_1 \vee A_2$	$\{A_1\}$	$\{A_2\}$	
$\neg A_1 \vee \neg A_1 \vee A_3 \vee A_0$	$\{A_1, A_2\}$	$\{A_3\}$	
$\neg A_1 \vee \neg A_1 \vee A_3 \vee A_3 \vee A_0$	$\{A_1, A_2, A_3\}$	$\emptyset$	true in I_C
$\neg A_1 \vee A_4 \vee A_3 \vee A_0$	$\{A_1, A_2, A_3\}$	$\emptyset$	
$\neg A_1 \vee \neg A_4 \vee A_3$	$\{A_1, A_2, A_3\}$	$\emptyset$	true in I_C
$\neg A_3 \vee A_5$	$\{A_1, A_2, A_3\}$	$\{A_5\}$	

The resulting $I = \{A_1, A_2, A_3, A_5\}$ is a model of the clause set.

Construction of Candidate Interpretations

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

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

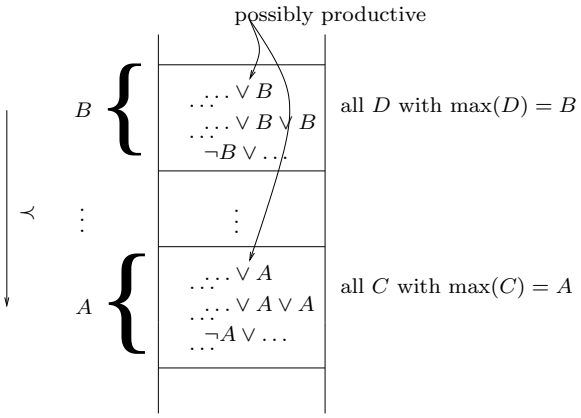
$$\Delta_C := \begin{cases} \{A\}, & \text{if } C \in N,\ C = C' \vee A,\ A \succ C',\ I_C \not\models C \\ \emptyset, & \text{otherwise} \end{cases}$$

We say that C *produces* A, if $\Delta_C = \{A\}$.

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

Structure of $N, \succ$

Let $A \succ B$; producing a new atom does not affect smaller clauses.



Some Properties of the Construction

Proposition 2.15

(i) $C = \neg A \vee C' \;\Rightarrow\;$ *no* $D \succeq C$ *produces* A.

(ii) C *productive* $\Rightarrow I_C \cup \Delta_C \models C$.

(iii) *Let* $D' \succ D \succeq C$. *Then*

$$I_D \cup \Delta_D \models C \Rightarrow I_{D'} \cup \Delta_{D'} \models C \text{ and } I_N \models C.$$

If, in addition, $C \in N$ *or* $\max(D) \succ \max(C)$*:*

$$I_D \cup \Delta_D \not\models C \Rightarrow I_{D'} \cup \Delta_{D'} \not\models C \text{ and } I_N \not\models C.$$

(iv) *Let* $D' \succ D \succ C$. *Then*

$$I_D \models C \Rightarrow I_{D'} \models C \text{ and } I_N \models C.$$

If, in addition, $C \in N$ *or* $\max(D) \succ \max(C)$*:*

$$I_D \not\models C \Rightarrow I_{D'} \not\models C \text{ and } I_N \not\models C.$$

(v) $D = C \vee A$ *produces* $A \Rightarrow I_N \not\models C$.

Model Existence Theorem

Theorem 2.16 (Bachmair & Ganzinger 1990) *Let $\succ$ be a clause ordering, let N be saturated w. r. t. Res, and suppose that $\bot \notin N$. Then $I_N^{\succ} \models N$.*

Corollary 2.17 *Let N be saturated w. r. t. Res. Then $N \models \bot \Leftrightarrow \bot \in N$.*

Proof of Theorem 2.16. Suppose $\bot \notin N$, but $I_N^{\succ} \not\models N$. Let $C \in N$ minimal (in $\succ$) such that $I_N^{\succ} \not\models C$. Since C is false in I_N, C is not productive. As $C \neq \bot$ there exists a maximal atom A in C.

Case 1: $C = \neg A \vee C'$ (i. e., the maximal atom occurs negatively)
$\Rightarrow I_N \models A$ and $I_N \not\models C'$
$\Rightarrow$ some $D = D' \vee A \in N$ produces A. As $\frac{D' \vee A \qquad \neg A \vee C'}{D' \vee C'}$, we infer that $D' \vee C' \in N$, and $C \succ D' \vee C'$ and $I_N \not\models D' \vee C'$
$\Rightarrow$ contradicts minimality of C.

Case 2: $C = C' \vee A \vee A$. Then $\frac{C' \vee A \vee A}{C' \vee A}$ yields a smaller counterexample $C' \vee A \in N$. $\Rightarrow$ contradicts minimality of C. □

Compactness of Propositional Logic

Theorem 2.18 (Compactness) *Let N be a set of propositional 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.
$\Rightarrow Res^*(N)$ unsatisfiable
$\Rightarrow \bot \in Res^*(N)$ by refutational completeness of resolution
$\Rightarrow \exists n \geq 0 : \bot \in Res^n(N)$
$\Rightarrow \bot$ has a finite resolution proof P;
choose M as the set of assumptions in P. □

2.9 General Resolution

Propositional resolution:

refutationally complete,

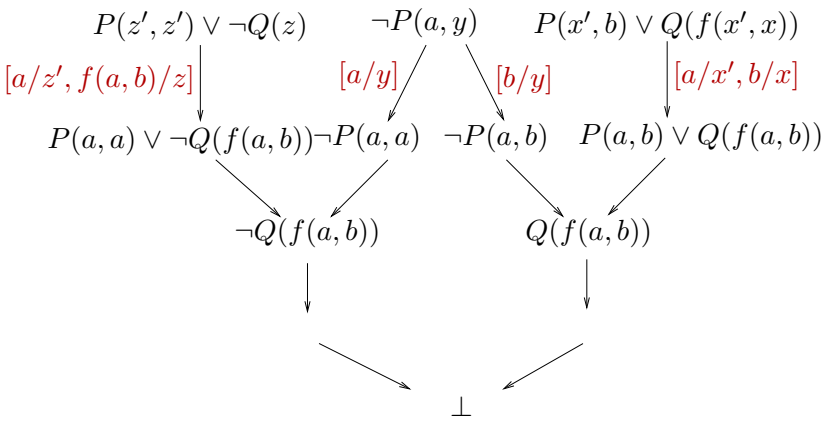
in its most naive version: not guaranteed to terminate for satisfiable sets of clauses, (improved versions do terminate, however)

in practice clearly inferior to the DPLL procedure (even with various improvements).

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

General Resolution through Instantiation

Idea: instantiate clauses appropriately:



Problems:

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

Even worse: There are infinitely many possible instances.

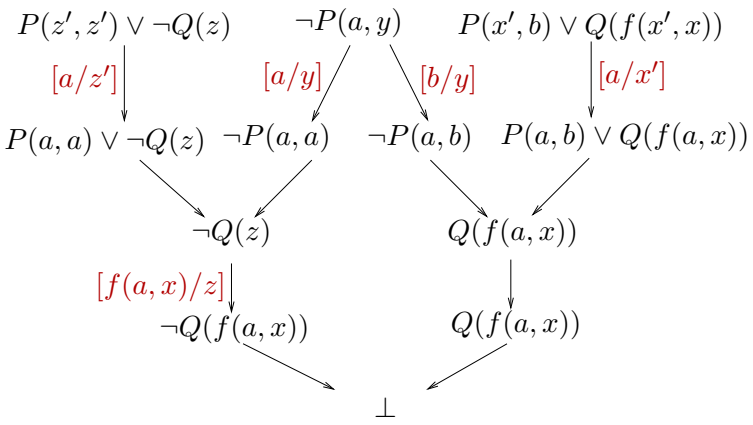
Observation:

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

Idea:

Do not instantiate more than necessary to get complementary literals.

Idea: do not instantiate more than necessary:



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.

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*:

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

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

General resolution RIF with implicit factorization:

$$\frac{D \vee B_1 \vee \ldots \vee B_n \qquad C \vee \neg A}{(D \vee C)\sigma} \quad \text{if } \sigma = \text{mgu}(A, B_1, \ldots, B_n)$$

[RIF]

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 multi-set 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*.

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 $\text{mgu}(E)$.

Proposition 2.19

(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 2.20 *σ is idempotent iff $dom(\sigma) \cap codom(\sigma) = \emptyset$.*

Rule Based Naive Standard Unification

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

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 $\sigma_E = [u_1/x_1, \ldots, u_k/x_k]$.

Proposition 2.21 *If E is a solved form then σ_E is an mgu of E.*

Theorem 2.22

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, $\sigma \circ [t/x] = \sigma[x \mapsto t\sigma] = \sigma[x \mapsto x\sigma] = \sigma$. Therefore, for any equation $u \doteq v$ in E: $u\sigma = v\sigma$, iff $u[t/x]\sigma = v[t/x]\sigma$. (2) and (3) follow by induction from (1) using Proposition 2.21. □

Main Unification Theorem

Theorem 2.23 *E is unifiable if and only if there is a most general unifier σ of E, such that σ is idempotent and $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$.*

Problem: *exponential growth* of terms possible

There are better, linear unification algorithms (not discussed here)

Proof of Theorem 2.23.

- $\Rightarrow_{SU}$ is Noetherian. A suitable lexicographic ordering on the multisets E (with $\bot$ minimal) shows this. Compare in this order:
  1. the number of defined variables (d.h. variables x in equations $x \doteq t$ with $x \notin var(t)$), which also occur outside their definition elsewhere in E;
  2. the multi-set ordering induced by (i) the size (number of symbols) in an equation; (ii) if sizes are equal consider $x \doteq t$ smaller than $t \doteq x$, if $t \notin X$. □
- A system E that is irreducible w. r. t. $\Rightarrow_{SU}$ is either $\bot$ or a solved form.
- Therefore, reducing any E by SU will end (no matter what reduction strategy we apply) in an irreducible E' having the same unifiers as E, and we can read off the mgu (or non-unifiability) of E from E' (Theorem 2.22, Proposition 2.21).
- σ is idempotent because of the substitution in rule 4. $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$, as no new variables are generated.

Lifting Lemma

Lemma 2.24 *Let C and D be variable-disjoint clauses. If*

$$\dfrac{\begin{array}{cc} D & C \\ \downarrow \sigma & \downarrow \rho \\ D\sigma & C\rho \end{array}}{C'} \quad \textit{[propositional resolution]}$$

then there exists a substitution τ such that

$$\frac{\begin{array}{cc} D & C \end{array}}{\begin{array}{c} C'' \\ \downarrow \tau \\ C' = C''\tau \end{array}} \qquad \textit{[general resolution]}$$

An analogous lifting lemma holds for factorization.

Saturation of Sets of General Clauses

Corollary 2.25 *Let N be a set of general clauses saturated under Res, i. e., $Res(N) \subseteq N$. Then also $G_\Sigma(N)$ is saturated, that is,*

$$Res(G_\Sigma(N)) \subseteq G_\Sigma(N).$$

Proof. W.l.o.g. we may assume that clauses in N are pairwise variable-disjoint. (Otherwise make them disjoint, and this renaming process changes neither $Res(N)$ 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 2.26 *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 2.27 *Let N be a set of Σ-clauses, let $\mathcal{A}$ be a* Herbrand *interpretation. Then $\mathcal{A} \models G_\Sigma(N)$ implies $\mathcal{A} \models N$.*

Theorem 2.28 (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 $N \not\models \bot$.

$$\begin{aligned}
N \not\models \bot &\Rightarrow \bot \notin Res^*(N) && \text{(resolution is sound)}\\
&\Rightarrow \bot \notin G_\Sigma(Res^*(N))\\
&\Rightarrow I_{G_\Sigma(Res^*(N))} \models G_\Sigma(Res^*(N)) && \text{(Thm. 2.16; Cor. 2.25)}\\
&\Rightarrow I_{G_\Sigma(Res^*(N))} \models Res^*(N) && \text{(Lemma 2.27)}\\
&\Rightarrow I_{G_\Sigma(Res^*(N))} \models N && (N \subseteq Res^*(N)) \quad \square
\end{aligned}$$

Refutational Completeness of General Resolution

Theorem 2.29 *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 2.25: $Res(G_\Sigma(N)) \subseteq G_\Sigma(N)$

$$\begin{aligned}
N \models \bot &\Leftrightarrow G_\Sigma(N) \models \bot && \text{(Lemma 2.26/2.27; Theorem 2.28)}\\
&\Leftrightarrow \bot \in G_\Sigma(N) && \text{(propositional resolution sound and complete)}\\
&\Leftrightarrow \bot \in N \quad \square
\end{aligned}$$

2.10 Ordered Resolution with Selection

Motivation: Search space for *Res very* large.

Ideas for improvement:

1. In the completeness proof (Model Existence Theorem 2.16) one only needs to resolve and factor maximal atoms
   $\Rightarrow$ if the calculus is restricted to inferences involving maximal atoms, the proof remains correct
   $\Rightarrow$ *order 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

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

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

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

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

Resolution Calculus $Res_S^{\succ}$

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:

Let $\succ$ be a total and well-founded ordering on ground atoms. A literal L is called *[strictly] maximal* in a clause C if and only if there exists a ground substitution σ such that for no other L' in C: $L\sigma \prec L'\sigma$ $[L\sigma \preceq L'\sigma]$.

Let $\succ$ be an atom ordering and S a selection function.

$$\frac{D \vee B \qquad C \vee \neg A}{(D \vee C)\sigma} \qquad [\textit{ordered resolution with selection}]$$

if $\sigma = \mathrm{mgu}(A, B)$ and

(i) $B\sigma$ strictly maximal w. r. t. $D\sigma$;

(ii) nothing is selected in D by S;

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

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \qquad [\textit{ordered factoring}]$$

if $\sigma = \mathrm{mgu}(A, B)$ and $A\sigma$ is maximal in $C\sigma$ and nothing is selected in C.

Special Case: Propositional Logic

For ground clauses the resolution inference simplifies to

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

if

(i) $A \succ D$;

(ii) nothing is selected in D by. S;

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

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

Search Spaces Become Smaller

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

we assume $A \succ B$ and S 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

$$\frac{\dfrac{C_1 \vee A \quad C_2 \vee \neg A \vee B}{C_1 \vee C_2 \vee B} \qquad C_3 \vee \neg B}{C_1 \vee C_2 \vee C_3}$$

we can obtain by *rotation*

$$\dfrac{C_1 \vee A \qquad \dfrac{C_2 \vee \neg A \vee B \quad C_3 \vee \neg B}{C_2 \vee \neg A \vee C_3}}{C_1 \vee C_2 \vee C_3}$$

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

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 of rotation-equivalent proofs.

Lifting Lemma for $Res_S^{\succ}$

Lemma 2.30 *Let D and C be variable-disjoint clauses. If*

$$\dfrac{\begin{array}{cc} D & C \\ \downarrow \sigma & \downarrow \rho \\ D\sigma & C\rho \end{array}}{C'} \qquad \textit{[propositional inference in } Res_S^{\succ}\textit{]}$$

and if $S(D\sigma) \simeq S(D)$, $S(C\rho) \simeq S(C)$ (that is, "corresponding" literals are selected), then there exists a substitution τ such that

$$\dfrac{\dfrac{D \qquad C}{C''}}{\begin{array}{c} \downarrow \tau \\ C' = C''\tau \end{array}} \qquad \textit{[inference in } Res_S^{\succ}\textit{]}$$

An analogous lifting lemma holds for factorization.

Saturation of General Clause Sets

Corollary 2.31 *Let N be a set of general clauses saturated under $Res_S^{\succ}$, i. e., $Res_S^{\succ}(N) \subseteq N$. Then there exists a selection function S' such that $S|_N = S'|_N$ and $G_\Sigma(N)$ is also saturated, i. e.,*

$$Res_{S'}^{\succ}(G_\Sigma(N)) \subseteq G_\Sigma(N).$$

Proof. We first define the selection function S' such that $S'(C) = S(C)$ 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 $S'(C)$ to be those occurrences of literals that are ground instances of the occurrences selected by S in D. Then proceed as in the proof of Corollary 2.25 using the above lifting lemma. □

Soundness and Refutational Completeness

Theorem 2.32 *Let $\succ$ be an atom ordering and S a selection function such that $Res^{\succ}_{S}(N) \subseteq N$. Then*

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

Proof. The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part consider first the propositional level: Construct a candidate interpretation I_N as for unrestricted resolution, except that clauses C in N that have selected literals are not productive, even when they are false in I_C and when their maximal atom occurs only once and positively. The result for general clauses follows using Corollary 2.31. □

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

Let N be a set of ground clauses and C a ground clause (not necessarily in N). C is called *redundant* w. r. t. N, if there exist $C_1, \ldots, C_n \in N$, $n \geq 0$, such that $C_i \prec C$ and $C_1, \ldots, C_n \models C$.

Redundancy for general clauses: C is called *redundant* w. r. t. N, if all ground instances $C\sigma$ of C are redundant w. r. t. $G_\Sigma(N)$.

Intuition: Redundant clauses are neither minimal counterexamples nor productive.

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

Examples of Redundancy

Proposition 2.33 *Some redundancy criteria:*

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

(Under certain conditions one may also use non-strict subsumption, but this requires a slightly more complicated definition of redundancy.)

Saturation up to Redundancy

N is called *saturated up to redundancy* (w. r. t. $Res_S^{\succ}$)

$$:\Leftrightarrow Res_S^{\succ}(N \setminus Red(N)) \subseteq N \cup Red(N)$$

Theorem 2.34 *Let* N *be saturated up to redundancy. Then*

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

Proof (Sketch). (i) Ground case:

- consider the construction of the candidate interpretation $I_N^{\succ}$ for $Res_S^{\succ}$
- redundant clauses are not productive
- redundant clauses in N are not minimal counterexamples for $I_N^{\succ}$

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

(ii) Lifting: no additional problems over the proof of Theorem 2.32. □

Monotonicity Properties of Redundancy

Theorem 2.35

(i) $N \subseteq M \Rightarrow Red(N) \subseteq Red(M)$

(ii) $M \subseteq Red(N) \Rightarrow Red(N) \subseteq Red(N \setminus M)$

Proof. Exercise. □

We conclude that redundancy is preserved when, during a theorem proving process, one adds (derives) new clauses or deletes redundant clauses.

A Resolution Prover

So far: static view on completeness of resolution:

Saturated sets are inconsistent if and only if they contain $\bot$.

We will now consider a dynamic view:

How can we get saturated sets in practice?

The theorems 2.34 and 2.35 are the basis for the completeness proof of our prover RP.

Rules for Simplifications and Deletion

We want to employ the following rules for simplification of prover states N:

- *Deletion of tautologies*

  $$N \cup \{C \vee A \vee \neg A\} \ \triangleright \ N$$

- *Deletion of subsumed clauses*

  $$N \cup \{C, D\} \ \triangleright \ N \cup \{C\}$$

  if $C\sigma \subseteq D$ (C *subsumes* D).

- *Reduction* (also called *subsumption resolution*)

  $$N \cup \{C \vee L, \ \ D \vee C\sigma \vee \overline{L}\sigma\} \ \triangleright \ N \cup \{C \vee L, D \vee C\sigma\}$$

Resolution Prover RP

3 clause sets: N(ew) containing new resolvents
P(rocessed) containing simplified resolvents
clauses get into O(ld) once their inferences have been computed

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

Transition Rules for RP (I)

Tautology elimination
$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$
if C is a tautology

Forward subsumption
$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$
if some $D \in \boldsymbol{P} \cup \boldsymbol{O}$ subsumes C

Backward subsumption
$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \cup \{D\} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$
$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{D\} \quad \triangleright \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$
if C strictly subsumes D

Transition Rules for RP (II)

Forward reduction
$\boldsymbol{N} \cup \{C \vee L\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$
if there exists $D \vee L' \in \boldsymbol{P} \cup \boldsymbol{O}$
such that $\overline{L} = L'\sigma$ and $D\sigma \subseteq C$

Backward reduction
$\boldsymbol{N} \mid \boldsymbol{P} \cup \{C \vee L\} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$
$\boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{C \vee L\} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$
if there exists $D \vee L' \in \boldsymbol{N}$
such that $\overline{L} = L'\sigma$ and $D\sigma \subseteq C$

Transition Rules for RP (III)

Clause processing
$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$

Inference computation
$\emptyset \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{C\}$,
with $\boldsymbol{N} = Res_S^{\succ}(\boldsymbol{O} \cup \{C\})$

Soundness and Completeness

Theorem 2.36

$$N \models \bot \quad \Leftrightarrow \quad N \mid \emptyset \mid \emptyset \quad \overset{*}{\triangleright} \quad N' \cup \{\bot\} \mid \_ \mid \_$$

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

Fairness

Problem:

If N is inconsistent, then $N \mid \emptyset \mid \emptyset \overset{*}{\triangleright} N' \cup \{\bot\} \mid \_ \mid \_$.

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

No: a clause could be kept in $\boldsymbol{P}$ without ever being used for an inference.

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 $\boldsymbol{P}$ 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 $\bot$.

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

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

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 S, or else nothing is selected in the right premise and $n = 1$ and $\neg A_1\sigma$ is maximal in $C\sigma$.

Similarly to resolution, hyperresolution has to be complemented by a factoring inference.

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

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

2.11 Summary: Resolution Theorem Proving

- Resolution is a machine calculus.
- Subtle interleaving of enumerating ground instances and proving inconsistency through the use of unification.
- Parameters: atom ordering $\succ$ and selection function S. On the non-ground level, ordering constraints can (only) be solved approximatively.
- Completeness proof by constructing candidate interpretations from productive clauses $C \vee A$, $A \succ C$; inferences with those reduce counterexamples.
- *Local* restrictions of inferences via $\succ$ and S
  $\Rightarrow$ 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)
  $\Rightarrow$ further specialization of inference systems required.

2.12 Other Inference Systems

Instantiation-based methods for FOL:

- (Analytic) Tableau;
- Resolution-based instance generation;
- Disconnection calculus;
- First-Order DPLL (Model Evolution)

Further (mainly propositional) proof systems:

- Hilbert calculus;
- Sequent calculus;
- Natural deduction.

Tableau

- Proof by refutation by analyzing the given formula's boolean structure, and by instantiating quantifed sub-formulas
- Hence, no normal form required and no "new" formulas generated (unlike Resolution)
- "Free variable" variant uses unification and can be used for first-order logic theorem proving
- Main applications, however, as decision procedures for
  - Description logics
  - (Propositional) modal logics

Constructing Tableau Proofs

Data structure: a proof is represented as a tableau — a binary tree, the nodes of which are labelled with formulas

Start: put the premises and the negated conclusion into the root of an otherwise empty tableau

Expansion: apply expansion rules to the formulas on the tree, thereby adding (instantiated, sub-)formulas and splitting branches

Closure: close (abandon) branches that are obviously contradictory

Refutation: a tableau expansion such that all branches are closed

Propositional Tableau Expansion Rules ("Smullyan-style")

ϕ, ϕ_1, ϕ_2 are propositional formulas

Alpha Rules

$$\frac{\phi_1 \wedge \phi_2}{\begin{array}{c}\phi_1\\ \phi_2\end{array}} \qquad \frac{\neg(\phi_1 \vee \phi_2)}{\begin{array}{c}\neg\phi_1\\ \neg\phi_2\end{array}} \qquad \frac{\neg(\phi_1 \rightarrow \phi_2)}{\begin{array}{c}\phi_1\\ \neg\phi_2\end{array}}$$

$\neg\neg$-Elimination

$$\frac{\neg\neg\phi}{\phi}$$

Beta Rules **Branch Closure**

$$\frac{\phi_1 \vee \phi_2}{\phi_1 \mid \phi_2} \qquad \frac{\neg(\phi_1 \wedge \phi_2)}{\neg\phi_1 \mid \neg\phi_2} \qquad \frac{\neg(\phi_1 \to \phi_2)}{\begin{array}{c}\neg\phi_1\\ \phi_2\end{array}} \qquad \frac{\begin{array}{c}\phi\\ \neg\phi\end{array}}{\times}$$

Example

(1) $\neg(A \vee (B \vee C)) \wedge \neg(B \wedge \neg C)$
(2) $\neg C$

(3) $\neg(A \vee (B \vee C))$
(4) $\neg(B \wedge \neg C)$

(5) $\neg B$	(6) $\neg\neg C$
(8) $\neg A$	(7) C
(9) $\neg(B \vee C)$	$\times$
(10) $\neg B$	
(11) $\neg C$	

(3) and (4) from (1); (5) and (6) from (4); (8) and (9) from (3); (10) and (11) from (9)

Smullyan's Uniform Notation

Formulas are of conjunctive (α) or disjunctive (β) type:

α	α_1	α_2
$\phi_1 \wedge \phi_2$	ϕ_1	ϕ_2
$\neg(\phi_1 \vee \phi_2)$	$\neg\phi_1$	$\neg\phi_2$
$\neg(\phi_1 \to \phi_2)$	ϕ_1	$\neg\phi_2$

β	β_1	β_2
$\phi_1 \vee \phi_2$	ϕ_1	ϕ_2
$\phi_1 \to \phi_2$	$\neg\phi_1$	ϕ_2
$\neg(\phi_1 \wedge \phi_2)$	$\neg\phi_1$	$\neg\phi_2$

The alpha and beta rules can be stated now as follows:

$$\frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta}{\beta_1 \mid \beta_2}$$

Important Properties (Informally)

A branch in a tableau is *satisfiable* iff the conjunction of its formulas is satisfiable. A tableau is *satisfiable* iff some of its branches is satisfiable.

A tableau expansion is *strict* iff every rule is applied at most once to every formula occurence in every branch.

Termination. Every strict tableau expansion is finite.

Soundness. No tableau expansion starting with a satisfiable set of formulas is a refutation.

Completeness. A tableau expansion is *fair* iff every rule aplicable to a formula occurence in a branch is applied eventually.

Every fair strict tableau expansion starting with an unsatisfiable set of formulas is a refutation.

Contrapositive: every open branch in a fair strict tableau expansion provides a model for the clause set in the root.

Tableaux for First-Order Logic

Formulas of universal (γ) or existential (δ) type:

γ	$\gamma_1(u)$
$\forall x\ \phi$	$\phi[u/x]$
$\neg\exists x\ \phi$	$\neg\phi[u/x]$

δ	$\delta_1(u)$
$\exists x\ \phi$	$\phi[u/x]$
$\neg\forall x\ \phi$	$\neg\phi[u/x]$

The gamma and delta rules can be stated now as follows:

$$\frac{\gamma}{\gamma_1(t)} \qquad \frac{\delta}{\delta_1(c)}$$

where

- t is an arbitrary ground term
- c is a constant symbol new to the branch

N.B: To obtain a complete calculus, the number of γ-rule applications to the same formula cannot be finitely bounded

Instantiation-Based Methods for FOL

Idea:

Overlaps of complementary literals produce instantiations (as in resolution);

However, contrary to resolution, clauses are not recombined.

Clauses are temporarily grounded – replace every variable by a constant – and checked for unsatisfiability; use an efficient propositional proof method, a "SAT-solver" for that.

Main variants: (ordered) semantic hyperlinking [Plaisted et al.], resolution-based instance generation (Inst-Gen) [Ganzinger and Korovin]

Resolution-Based Instance Generation

$$\frac{D \vee B \qquad C \vee \neg A}{(D \vee B)\sigma \qquad (C \vee \neg A)\sigma} \qquad [\textit{Inst-Gen}]$$

if $\sigma = \mathrm{mgu}(A, B)$ and at least one conclusion is a proper instance of its premise.

The instance-generation calculus saturates a given clause set under Inst-Gen and periodically passes the ground-instantiated version of the current clause set to a SAT-solver.

A refutation has been found if the SAT-solver determines unsatisfiability.

Other methods *do not* use a SAT-solver as a subroutine;

Instead, the *same* base calculus is used to generate new clause instances and test for unsatisfiability of grounded data structures.

Main variants: tableau variants, such as the disconnection calculus [Billon; Letz and Stenz], and a variant of the DPLL procedure for first-order logic [Baumgartner and Tinelli].

3 Implementation Issues

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.

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.)

3.1 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.

Consequently: split the set of clauses into two subsets.

- W = "Worked-off" (or "active") clauses: Have already been selected as "given clause". (So all inferences between these clauses have already been computed.)
- U = "Usable" (or "passive") clauses: Have not yet been selected as "given clause".

During each iteration of the main loop:

Select a new given clause C from U; $U := U \setminus \{C\}$.

Find partner clauses D_i from W; $New = Infer(\{ D_i \mid i \in I \}, C)$; $U = U \cup New$; $W = W \cup \{C\}$

Additionally:

Try to simplify C using W. (Skip the remainder of the iteration, if C can be eliminated.)

Try to simplify (or even eliminate) clauses from W using C.

Design decision: should one also simplify U using W ?

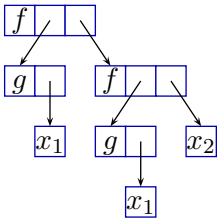
yes $\leadsto$ "Otter loop":
Advantage: simplifications of U may be useful to derive the empty clause.

no $\leadsto$ "Discount loop":
Advantage: clauses in U are really passive; only clauses in W have to be kept in index data structure. (Hence: can use index data structure for which retrieval is faster, even if update is slower and space consumption is higher.)

3.2 Term Representations

The obvious data structure for terms: Trees

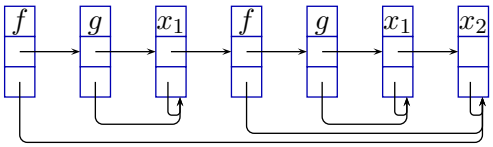
$f(g(x_1), f(g(x_1), x_2))$



optionally: (full) sharing

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.

3.3 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,
- ...

Requirements:

fast insertion,

fast deletion,

fast retrieval,

small memory consumption.

Note: In applications like functional or logic programming, the requirements are different (insertion and deletion are much less important).

Many different approaches:

- Path indexing
- Discrimination trees
- Substitution trees
- Context trees
- Feature vector indexing
- ...

Perfect filtering:

The indexing technique returns exactly those terms satisfying the query.

Imperfect filtering:

The indexing technique returns some superset of the set of all terms satisfying the query.

Retrieval operations must be followed by an additional check, but the index can often be implemented more efficiently.

Frequently: All occurrences of variables are treated as different variables.

Path Indexing

Path indexing:

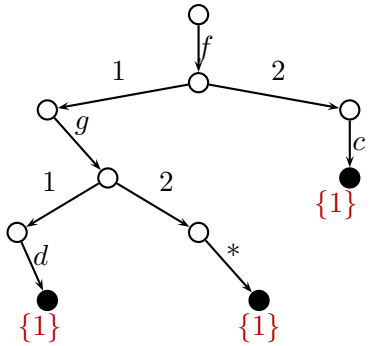
Paths of terms are encoded in a trie ("retrieval tree").

A star $*$ represents arbitrary variables.

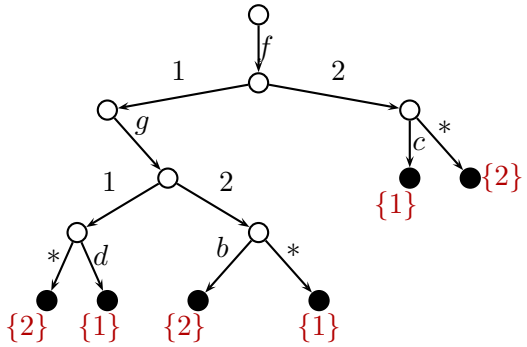
Example: Paths of $f(g(*, b), *)$: $f.1.g.1.*$
$f.1.g.2.b$
$f.2.*$

Each leaf of the trie contains the set of (pointers to) all terms that contain the respective path.

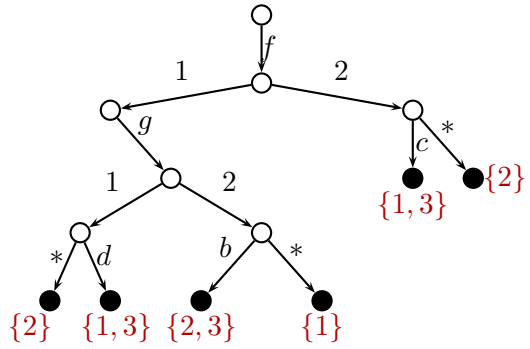
Example: Path index for $\{f(g(d, *), c)\}$



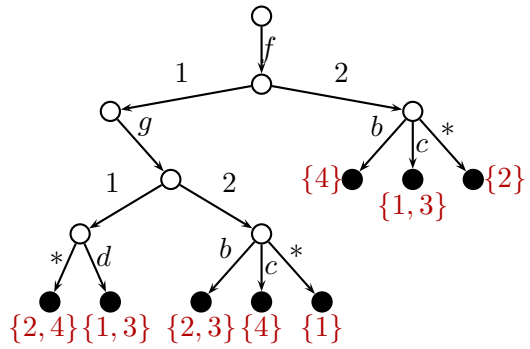
Example: Path index for $\{f(g(d, *), c),\ f(g(*, b), *)\}$



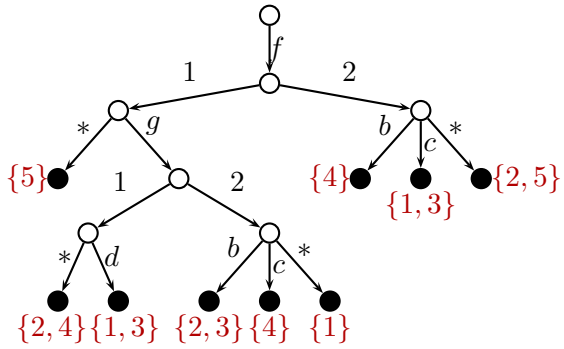
Example: Path index for $\{f(g(d, *), c),\ f(g(*, b), *),\ f(g(d, b), c)\}$



Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b)\}$



Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b),\ f(*,*)\}$



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:

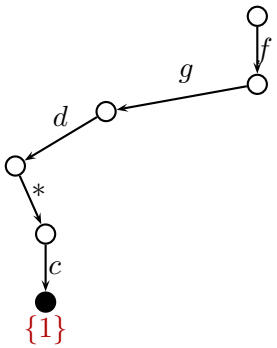
Preorder traversals of terms are encoded in a trie.

A star $*$ represents arbitrary variables.

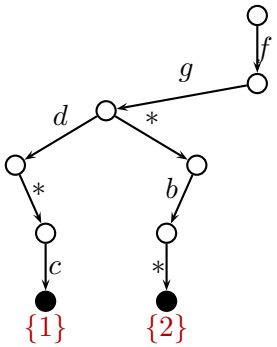
Example: String of $f(g(*, b), *)$: $f.g.*.b.*$

Each leaf of the trie contains (a pointer to) the term that is represented by the path.

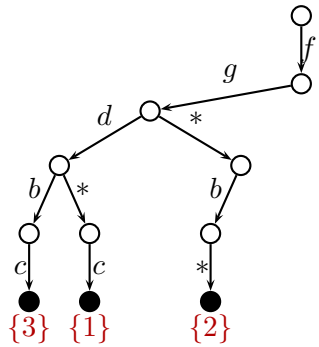
Example: Discrimination tree for $\{f(g(d, *), c)\}$



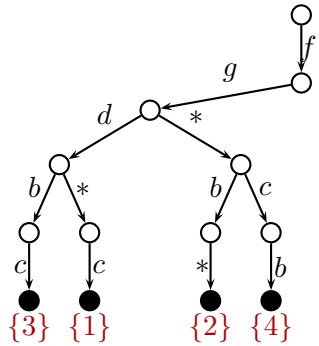
Example: Discrimination tree for $\{f(g(d, *), c),\ f(g(*, b), *)\}$



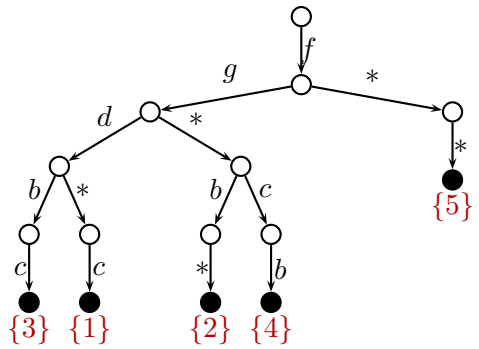
Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c)\}$



Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b)\}$



Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b),\ f(*,*)\}$



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.

Literature

Melvin Fitting: First-Order Logic and Automated Theorem Proving, Springer, 1996.

Leo Bachmair, and Harald Ganzinger: Resolution Theorem Proving, Ch. 2 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. I*, Elsevier, 2001.
Preprint: `http://www.mpi-inf.mpg.de/~hg/papers/reports/MPI-I-97-2-005.ps.gz`

The Wikipedia article on "Automated theorem proving":
`http://en.wikipedia.org/wiki/Automated_theorem_proving`

Further Reading

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.

The End