Logic for Computer Science

Harald Ganzinger

Summer Term 2002

Logic in Computer Science

computation is deduction

- logic programming, relational data bases
- operational semantics of PLs

proof theory

- mathematics on the computer
- constructive proofs and program synthesis

axiomatized domains

- modelling in logic
- knowledge representation
- specification and verification
- rapid prototyping

descriptive complexity theory

Emphasis in this Course

- introduction into logics and deductive services underlying important domains of application
  - first-order logic (universal calculus; theorem proving)
  - Horn clause logic (logic programming; goal solving)
  - temporal logic (verification of communicating programs; model checking)
  - [typed λ-calculus (constructive proofs; program extraction)]
- proof systems: soundness, completeness, complexity, implementation
- efficient algorithms for specific deduction problems
- exercises: implementation (in SML) of theoretical constructions

Literature

Schöning: Logik für Informatiker, Spektrum

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

Huth, Ryan: Logic in Computer Science: Modelling and reasoning about systems, Cambridge University Press

Gallier: Logic for Computer Science, Harper & Row

Reeves, Clarke: Logic for Computer Science, Addison-Wesley

Ben-Ari: Mathematical Logic for Computer Science, Prentice Hall

Sperschneider, Antoniou: Logic, a Foundation for Computer Science

de Swart et al: Logic: Mathematics, Language, Computer Science and Philosophy, Vol. II: Logic and Computer Science, Peter Lang

Part 1: First-Order Logic

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

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

1.1 Syntax

- non-logical symbols (domain-specific)

  $\Rightarrow$ terms, atomic formulas
- logical symbols (domain-independent)

  $\Rightarrow$ Boolean combinations, quantifiers

Signature

Usage: fixing the alphabet of non-logical symbols

$$\Sigma = (\Omega, \Pi),$$

where

- Ω a set of function symbols f with arity $n \geq 0$, written f/n,
- Π a set of predicate symbols p with arity $m \geq 0$, written 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:

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

By $T_\Sigma(X)$ 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:

$$\begin{array}{llll} A, B & ::= & p(s_1, ..., s_m) & , p/m \in \Pi \\ \Big[& \mid & (s \approx t) & \text{(equation)} \quad \Big] \end{array}$$

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

Literals

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

Clauses

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

General First-Order Formulas

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

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

Notational Conventions

- We omit brackets according to the following rules:
  - $\neg \quad >_p \quad \vee \quad >_p \quad \wedge \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$ (binding precedences)
  - $\vee$ and $\wedge$ are associative and commutative
  - $\rightarrow$ is right-associative
- $Qx_1, \dots, x_n\, F$ abbreviates $Qx_1 \dots Qx_n\, F$.
- infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences; examples:

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

Example: Peano Arithmetic

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

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

Exampes of formulas over this signature are:

$\forall x, y(x \leq y \leftrightarrow \exists z(x + z \approx y))$

$\exists x \forall y(x + y \approx y)$

$\forall x, y(x * s(y) \approx x * y + x)$

$\forall x, y(s(x) \approx s(y) \rightarrow x \approx y)$

$\forall x \exists y\ (x < y \land \neg \exists z(x < z \land z < y))$

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

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

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 the presence of quantification it is surprisingly complex.

By $F[s/x]$ we denote the result of substituting all free occurrences of x in F by the term s.

Formally we define $F[s/x]$ by structural induction over the syntactic structure of F by the equations depicted on the next page.

Substitution of a Term for a Free Variable

$$
\begin{aligned}
x[s/x] &= s \\
x'[s/x] &= x' \ ; \quad \text{if } x' \neq x \\
f(s_1, \ldots, s_n)[s/x] &= f(s_1[s/x], \ldots, s_n[s/x]) \\
\bot[s/x] &= \bot \\
\top[s/x] &= \top \\
p(s_1, \ldots, s_n)[s/x] &= p(s_1[s/x], \ldots, s_n[s/x]) \\
(u \approx v)[s/x] &= (u[s/x] \approx v[s/x]) \\
\neg F[s/x] &= \neg(F[s/x]) \\
(F \rho G)[s/x] &= (F[s/x] \rho G[s/x]) \ ; \quad \text{for each binary connective } \rho \\
(QyF)[s/x] &= Qz((F[z/y])[s/x]) \ ; \ \text{with } z \text{ a ``fresh'' variable}
\end{aligned}
$$

Why Substitution is Complicated

We need to make sure that the (free) variables in s are not captured upon placing s into the scope of a quantifier, hence the renaming of the bound variable y into a “fresh”, that is, previously unused, variable z.

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

General Substitutions

In general, substitutions are mappings

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

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

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

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

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 $\sigma(x)$.

Modifying a Substitution

The modification of a substitution σ at x is defined as follows:

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

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

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

E: Convince yourself that for the special case $\sigma = [t/x]$ the new definition coincides with our previous definition (modulo the choice of fresh names for the bound variables).

Structural Induction

Theorem 1.1 Let $G = (N, T, P, S)$ be a context-free grammar[a] and let q be a property of T^* (the words over the alphabet T of terminal symbols of G).

q holds for all words $w \in L(G)$, whenever one can prove these 2 properties:

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

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

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

[a]Infinite grammars are also admitted.

Structural Recursion

THEOREM 1.2 Let $G = (N, T, P, S)$ be a unambiguous context-free grammar. A function f is well-defined on $L(G)$ (that is, unambiguously defined) whenever these 2 properties are satisfied:

1. (base cases)
   f is well-defined on the words $w' \in \Sigma^*$ for each rule $X ::= w'$ in P.

2. (step cases)
   If $X ::= w_0 X_0 w_1 \ldots w_n X_n w_{n+1}$ is a rule in P then
   $f(w_0 w'_0 w_1 \ldots w_n w'_n w_{n+1})$ is well-defined, assuming that each of the $f(w'_i)$ is well-defined.

Q: Why should G be unambigous?

Substitution Revisited

Q: Does Theorem 1.2 justify that our homomorphic extension

$$apply : F_\Sigma(X) \times (X \to T_\Sigma(X)) \quad\to\quad F_\Sigma(X),$$

with $apply(F, \sigma)$ denoted by $F\sigma$, of a substitution is well-defined?

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

The second problem is that Theorem 1.2 applies to unary functions only. The standard solution to this problem is to curryfy, that is, to consider the binary function as a unary function producing a unary (residual) function as a result:

$$apply : F_\Sigma(X) \quad\to\quad ((X \to T_\Sigma(X)) \to F_\Sigma(X))$$

where we have denoted $(apply(F))(\sigma)$ as $F\sigma$.

E: Convince yourself that this does the trick.

1.2. 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.

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.

Structures

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

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

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

Normally, by abuse of notation, we will have $\mathcal{A}$ denote both the algebra and its universe.

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 $\beta : X \to \mathcal{A}$.

Variable assignments are the semantic counterparts of substitutions.

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

By structural induction we define

$$\mathcal{A}(\beta) : T_\Sigma(X) \to \mathcal{A}$$

as follows:

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

In the scope of a quantifier we need to evaluate terms with respect to modified assigments. To that end, let $\beta[x \to a] : X \to \mathcal{A}$, 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

The set of truth values is given as $\{0,1\}$. $\mathcal{A}(\beta) : \Sigma\text{-formulas} \to \{0,1\}$ is defined inductively over the structure of F as follows:

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

Ex: “Standard” Interpretation $\mathbb{N}$ for Peano Arithmetic

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

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

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

1.3 Models, Validity, and Satisfiability

Validity and Satisfiability

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

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

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

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

F is valid (or is a tautology):

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

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

Substitution Lemma

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

Theorem 1.3 For any Σ-term t

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

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

Theorem 1.4 For any Σ-formula F, $\mathcal{A}(\beta)(F\sigma) = \mathcal{A}(\beta \circ \sigma)(F)$.

Corollary 1.5 $\mathcal{A}, \beta \models F\sigma \quad \Leftrightarrow \quad \mathcal{A}, \beta \circ \sigma \models F$

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

Entailment and Equivalence

F entails (implies) G (or G is entailed by F), written $F \models G$

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma$-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

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma$-Alg und $\beta \in X \to U_{\mathcal{A}}$ we have $\mathcal{A}, \beta \models F \;\Leftrightarrow\; \mathcal{A}, \beta \models G$.

Proposition 1.1 F entails G iff $(F \to G)$ is valid

Proposition 1.2 F and G are equivalent iff $(F \leftrightarrow G)$ is valid.

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

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma$-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 1.3

$$F \text{ valid} \;\Leftrightarrow\; \neg F \text{ unsatisfiable}$$

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

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

Theory of a Structure

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

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

Problem of axiomatizability:

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

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

Analoguously for sets of structures.

Two Interesting Theories

Let $\Sigma_{Pres} = (\{0/0, s/1, +/2\},\ \emptyset)$ and $\mathbb{Z}_+ = (\mathbb{Z}, 0, s, +)$ its standard interpretation on the integers.[a] $Th(\mathbb{Z}_+)$ is called Presburger arithmetic.[b] Presburger arithmetic is decidable in 3EXPTIME[c] (and there is a constant $c \geq 0$ such that $Th(\mathbb{Z}_+) \notin \text{NTIME}(2^{2^{cn}})$) and in 2EXPSPACE; usage of automata-theoretic methods.

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

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

[a] There is no essential difference when one, instead of $\mathbb{Z}$, considers the natural numbers $\mathbb{N}$ as standard interpretation.

[b] M. Presburger (1929)

[c] D. Oppen: A $2^{2^{2^n}}$ upper bound on the complexity of Presburger arithmetic. Journal of Computer and System Sciences, 16(3):323–332, July 1978

1.4 Algorithmic Problems

Validity(F):

$\models F$?

Satisfiability(F):

F satisfiable?

Entailment(F,G):

does F entail G?

Model(A,F):

$A \models F$?

Solve(A,F):

find an assignment 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 entails F

Gödel's Famous Theorems

1. For most signatures Σ, validity is undecidable for Σ-formulas.
   (We will prove this below.)

2. For each signature Σ, the set of valid Σ-formulas is recursively enumerable.
   (We will prove this by giving complete deduction systems.)

3. For $\Sigma = \Sigma_{PA}$ and $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *)$, the theory $Th(\mathbb{N}_*)$ is not recursively enumerable.

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

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

Some Decidable Fragments

- Monadic class: no function symbols, all predicates unary; validity NEXPTIME-complete
- Variable-free formulas without equality: satisfiability NP-complete
  Q: why?
- Variable-free Horn clauses (clauses with at most 1 positive atom): entailment is decidable in linear time (cf. below)
- Finite model checking is decidable in time polynomial in the size of the structure and the formula.

1.5 Normal Forms, Skolemization, Herbrand Models

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 quantifier-free, $Q_i \in \{\forall, \exists\}$; 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 $\Rightarrow_P$:

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

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):

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

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

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

Theorem 1.6 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 (wrt. Σ-Alg) $\Leftrightarrow$ H satisfiable (wrt. Σ'-Alg)

where $\Sigma' = (\Omega \cup SKF, \Pi)$, if $\Sigma = (\Omega, \Pi)$.

Clausal Normal Form (Conjunctive Normal Form)

$$
\begin{aligned}
(F \leftrightarrow G) &\quad\Rightarrow_K\quad (F \rightarrow G) \wedge (G \rightarrow F)\\
(F \rightarrow G) &\quad\Rightarrow_K\quad (\neg F \vee G)\\
\neg(F \vee G) &\quad\Rightarrow_K\quad (\neg F \wedge \neg G)\\
\neg(F \wedge G) &\quad\Rightarrow_K\quad (\neg F \vee \neg G)\\
\neg\neg F &\quad\Rightarrow_K\quad F\\
(F \wedge G) \vee H &\quad\Rightarrow_K\quad (F \vee H) \wedge (G \vee H)\\
(F \wedge \top) &\quad\Rightarrow_K\quad F\\
(F \wedge \bot) &\quad\Rightarrow_K\quad \bot\\
(F \vee \top) &\quad\Rightarrow_K\quad \top\\
(F \vee \bot) &\quad\Rightarrow_K\quad F
\end{aligned}
$$

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

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

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

Theorem 1.7 Let F be closed. $F' \models F$. The converse is not true in general.

Theorem 1.8 Let F be closed. F satisfiable iff F' satisfiable iff N satisfiable

Optimization

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

- size of the CNF exponential when done naively;
- want to preserve the original formula structure;
- want small arity of Skolem functions (cf. Info IV and tutorials)!

Herbrand Interpretations for FOL without Equality

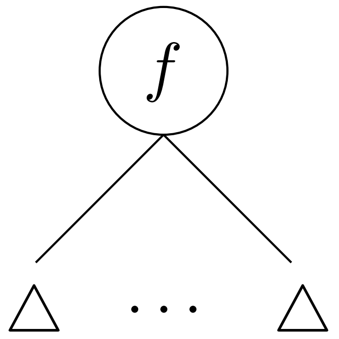
From now an we shall consider PL without equality. Ω shall contains at least one constant symbol.

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

(i) $U_{\mathcal{A}} = T_\Sigma$ (= the set of ground terms over Σ)

(ii) $f_{\mathcal{A}} : (s_1, \dots, s_n) \mapsto f(s_1, \dots, s_n),\ \ f/n \in \Omega$

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



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

Herbrand Interpretations as Sets of Ground Atoms

Proposition 1.9 Every set of ground atoms I uniquely determines a Herbrand interpretation $\mathcal{A}$ via

$$(s_1, \ldots, s_n) \in p_{\mathcal{A}} \;:\Leftrightarrow\; p(s_1, \ldots, s_n) \in I$$

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 1.10 (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

$$G_\Sigma(N) = \{C\sigma \text{ ground clause} \mid C \in N,\ \sigma : X \to T_\Sigma\}$$

the set of ground instances of N.

[Proof to 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:

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

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

$\ldots$

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

$\ldots$

1.6 Inference Systems, Proofs

Inference systems Γ (proof calculi) are sets of tuples

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

called inferences or inference rules, and written

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

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

A proof in Γ of a formula F from a a set of formulas N (called assumptions) is a sequence $F_1, \dots, F_k$ 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 $(F_{i_1},\ \dots,\ F_{i_{n_i}},\ F_i)$ in Γ, such that $0 \leq i_j < i$, for $1 \leq j \leq n_i$.

Soundness, 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 $:\Leftrightarrow$

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

Proofs as Trees

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

$$\dfrac{P(g(a,b)) \qquad \dfrac{\dfrac{\dfrac{P(f(a)) \lor Q(b) \qquad \dfrac{\dfrac{P(f(a)) \lor Q(b) \qquad \neg P(f(a)) \lor \neg P(f(a)) \lor Q(b)}{\neg P(f(a)) \lor Q(b) \lor Q(b)}}{\neg P(f(a)) \lor Q(b)}}{Q(b) \lor Q(b)}}{Q(b)} \qquad \neg P(f(a)) \lor \neg Q(b)}{\neg P(g(a,b))}}{\bot}$$

Proposition 1.11 (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).

1.7 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:

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

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

(positive) factorisation:

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

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

Resolution with Implicit Factorization RIF

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

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

Another Example

Soundness of Resolution

Theorem 1.12 Propositional resolution is sound.

Proof. Let $I \in \Sigma$-Alg. To be shown:

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

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

ad (i): Assume premises are valid in I. Two cases need to be considered:
(a) A is valid, or (b) $\neg A$ is valid.

a) $I \models A \;\Rightarrow\; I \models D \;\Rightarrow\; I \models C \vee D$

b) $I \models \neg A \;\Rightarrow\; I \models C \;\Rightarrow\; I \models C \vee D$

ad (ii): even simpler. $\square$

NB: 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 $I \models \neg A$.

1.8 Well-Founded Orderings

Literature: Baader F., Nipkow, T.: Term rewriting and all that. Cambridge U. Press, 1998, Chapter 2.

For showing completeness of resolution we will make use of the concept of well-founded orderings. A partial ordering $\succ$ on a set M is called well-founded (Noetherian) iff there exists no infinite descending chain

$$a_0 \succ a_1 \succ \ldots$$

in M.

NB: A partial ordering is transitive and irreflexive and not necessarily total (however our orderings usually are total).

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

Notation

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

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

Examples

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

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

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

on $M_1 \times M_2$ be defined as

$$(a_1, a_2) \succ (b_1, b_2) \;:\Leftrightarrow\; a_1 \succ_1 b_1, \text{ or else } a_1 = b_1 \;\&\; a_2 \succ_2 b_2$$

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

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

$$\begin{aligned} w \succ w' \quad := \quad & \alpha)\ |w| > |w'| \text{ or} \\ & \beta)\ |w| = |w'| \text{ and } w >_{\Sigma,lex} w', \end{aligned}$$

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

Basic Properties of Well-Founded Orderings

LEMMA 1.13 $(M, \succ)$ is well-founded $\Leftrightarrow$ every $\emptyset \subset M' \subseteq M$ has a minimal element.

LEMMA 1.14
$(M_i, \succ_i)$ well-founded , $i = 1, 2 \;\Leftrightarrow\; (M_1 \times M_2, (\succ_1, \succ_2)_{lex})$ well-founded.

Proof. (i) "$\Rightarrow$": Suppose $(M_1 \times M_2,\ \succ)$, with $\succ\, = (\succ_1, \succ_2)_{lex}$, is not well-founded. Then there is an infinite sequence

$$(a_0, b_0) \succ (a_1, b_1) \succ (a_2, b_2) \succ \ldots \quad .$$

Consider $A = \{a_i \mid i \geq 0\} \subseteq M_1$. A has a minimal element a_n, since $(M_1, \succ_1)$ is well-founded. But then $B = \{b_i \mid i \geq n\} \subseteq M_2$ can not have a minimal element; contradition to the well-foundedness of $(M_2, \succ_2)$.

(ii) "$\Leftarrow$": obvious. $\Box$

Noetherian Induction

Let $(M, \succ)$ be a well-founded ordering.

THEOREM 1.15 (NOETHERIAN INDUCTION) A property $Q(m)$ holds for all $m \in M$, whenever for all $m \in M$ this implication is satisfied:

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

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

[a] induction hypothesis
[b] induction step

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.

m is called 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 $S(a) = 3$, $S(b) = 2$, $S(c) = 0$.

Multi-Set Orderings

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

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

Theorem 1.16

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

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

c) $\succ$ total $\Rightarrow \succ_{mul}$ 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 such as ASCII.)

2. Extension to literals:

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

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

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

Example

Suppose $B_2 \succ A_2 \succ B_1 \succ A_1 \succ B_0 \succ A_0$. Dann:

$$A_0 \vee B_0$$

$$\prec$$

$$B_0 \vee A_1$$

$$\prec$$

$$\neg B_0 \vee A_1$$

$$\prec$$

$$\neg B_0 \vee A_2 \vee B_1$$

$$\prec$$

$$\neg B_0 \vee \neg A_2 \vee B_1$$

$$\prec$$

$$\neg B_2 \vee B_2$$

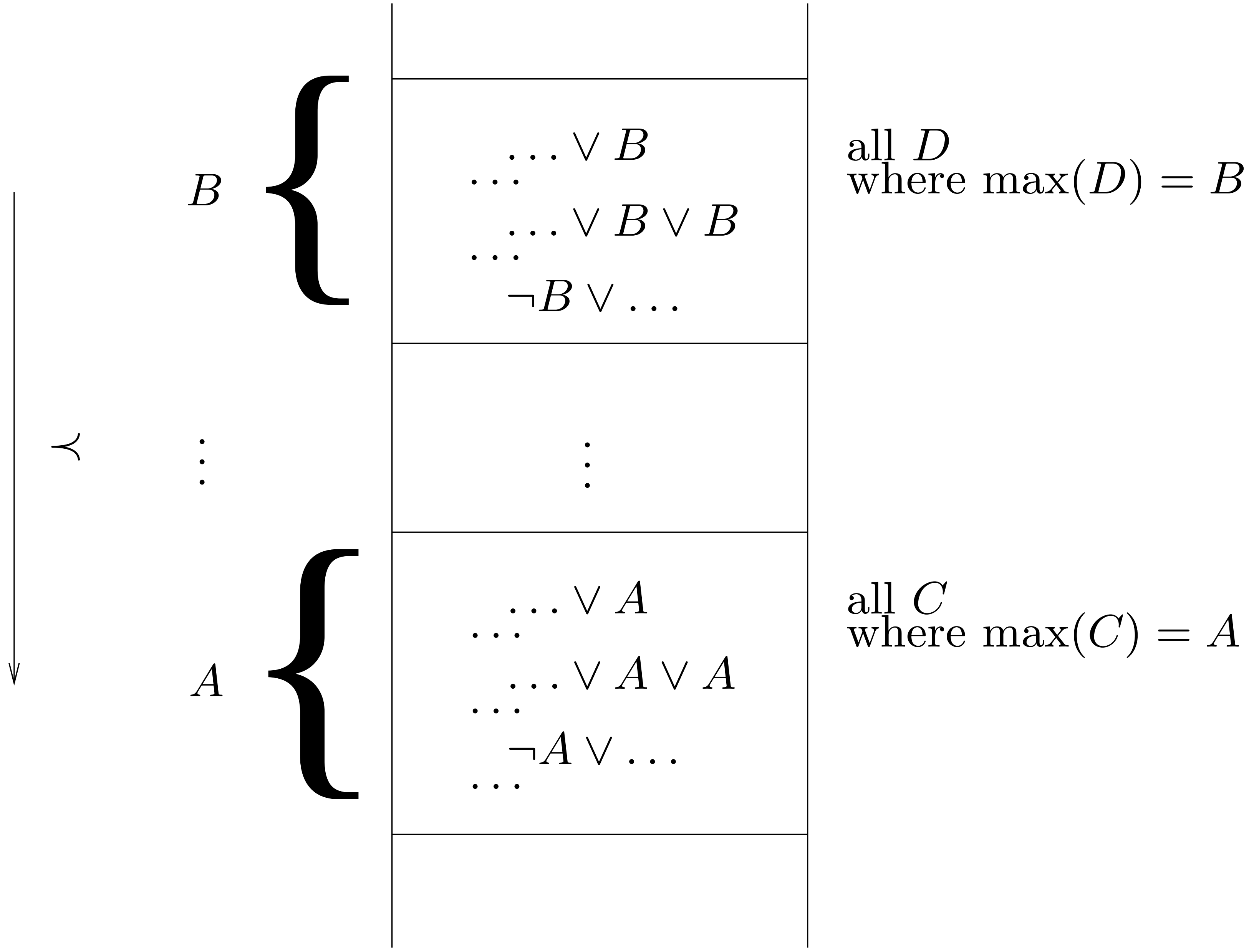
Properties of the Clause Ordering

Proposition 1.17

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*

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

N is called saturated (wrt. resolution), if $Res(N) \subseteq N$.

Proposition 1.18 (i) $Res^*(N)$ is saturated.
(ii) Res is refutationally complete, iff for each set N of ground clauses:

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

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 $B_2 \succ A_2 \succ B_1 \succ A_1 \succ B_0 \succ A_0$ (max. atoms in clauses in red)

	clauses C	I_C	Δ_C	Remarks
1	$\neg A_0$	$\emptyset$	$\emptyset$	true in I_C
2	$A_0 \lor B_0$	$\emptyset$	$\{B_0\}$	B_0 maximal
3	$B_0 \lor A_1$	$\{B_0\}$	$\emptyset$	true in I_C
4	$\neg B_0 \lor A_1$	$\{B_0\}$	$\{A_1\}$	A_1 maximal
5	$\neg B_0 \lor A_2 \lor B_1 \lor A_0$	$\{B_0, A_1\}$	$\{A_2\}$	A_2 maximal
6	$\neg B_0 \lor \neg A_2 \lor B_1$	$\{B_0, A_1, A_2\}$	$\emptyset$	B_1 not maximal; min. counterexample
7	$\neg B_0 \lor B_2$	$\{B_0, A_1, A_2\}$	$\{B_2\}$	

$I = \{B_0, A_1, A_2, B_2\}$ 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 truthvalue of clauses smaller than C shouldb 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 truthvalue of A has no effect on smaller clauses).

Resolution Reduces Counterexamples

$$\frac{\neg B_0 \lor A_2 \lor B_1 \lor A_0 \qquad \neg B_0 \lor \neg A_2 \lor B_1}{\neg B_0 \lor \neg B_0 \lor B_1 \lor B_1 \lor A_0}$$

Construction of I for the extended clause set:

clauses C	I_C	Δ_C	
$\neg A_0$	$\emptyset$	$\emptyset$	
$A_0 \lor B_0$	$\emptyset$	$\{B_0\}$	
$B_0 \lor A_1$	$\{B_0\}$	$\emptyset$	
$\neg B_0 \lor A_1$	$\{B_0\}$	$\{A_1\}$	
$\neg B_0 \lor \neg B_0 \lor B_1 \lor B_1 \lor A_0$	$\{B_0, A_1\}$	$\emptyset$	B_1 occurs twice
			minimal counterexample
$\neg B_0 \lor A_2 \lor B_1 \lor A_0$	$\{B_0, A_1\}$	$\{A_2\}$	
$\neg B_0 \lor \neg A_2 \lor B_1$	$\{B_0, A_1, A_2\}$	$\emptyset$	counterexample
$\neg B_0 \lor B_2$	$\{B_0, A_1, A_2\}$	$\{B_2\}$	

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

Factorization Reduces Counterexamples

$$\frac{\neg B_0 \lor \neg B_0 \lor B_1 \lor B_1 \lor A_0}{\neg B_0 \lor \neg B_0 \lor B_1 \lor A_0}$$

Construction of I for the extended clause set:

clauses C	I_C	Δ_C	
$\neg A_0$	$\emptyset$	$\emptyset$	
$A_0 \lor B_0$	$\emptyset$	$\{B_0\}$	
$B_0 \lor A_1$	$\{B_0\}$	$\emptyset$	
$\neg B_0 \lor A_1$	$\{B_0\}$	$\{A_1\}$	
$\neg B_0 \lor \neg B_0 \lor B_1 \lor A_0$	$\{B_0, A_1\}$	$\{B_1\}$	
$\neg B_0 \lor \neg B_0 \lor B_1 \lor B_1 \lor A_0$	$\{B_0, A_1, B_1\}$	$\emptyset$	
$\neg B_0 \lor A_2 \lor B_1$	$\{B_0, A_1, B_1\}$	$\emptyset$	true in I_C
$\neg B_0 \lor \neg A_2 \lor B_1$	$\{B_0, A_1, B_1\}$	$\emptyset$	true in I_C
$\neg B_1 \lor B_2$	$\{B_0, A_1, B_1\}$	$\{B_2\}$	

The resulting $I = \{B_0, A_1, B_1, B_2\}$ is a model of the clause set.

Construction of Candidate Models Formally

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 $\Delta_C = \{A\}$.

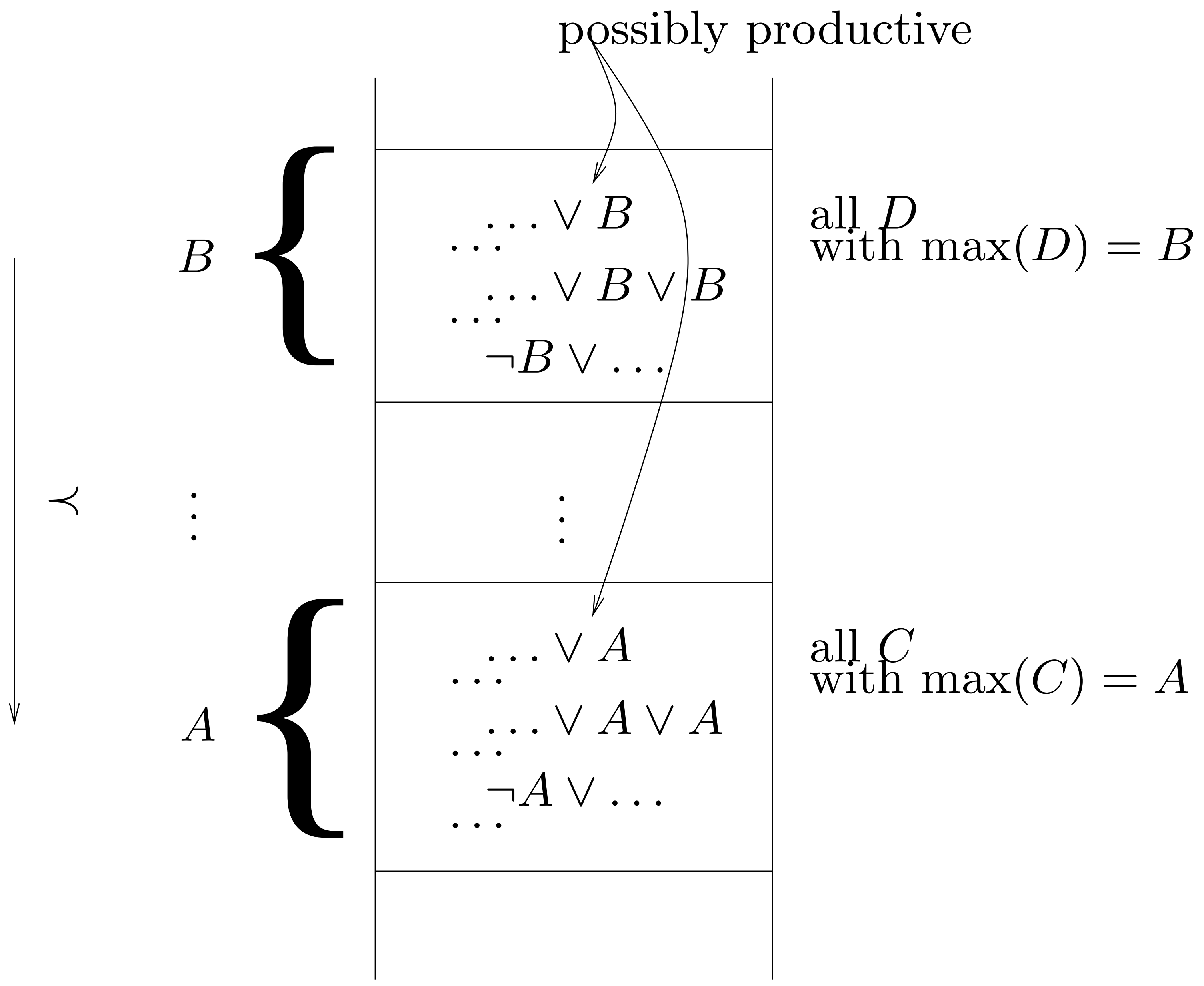
The candidate model for N (wrt. $\succ$) is given as

$$I_N^{\succ} := \bigcup_C \Delta_C.$$

We also simply write I_N, or I, for $I_N^{\succ}$ if $\succ$ is either irrelevant or known from the context.

Structure of $N, \succ$

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



Some Properties of the Construction

PROPOSITION 1.19

(i) $C = \neg A \vee C' \quad \Rightarrow \quad$ 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 1.20 (Bachmair, Ganzinger 1990) Let $\succ$ be a clause ordering, let N be saturated wrt. *Res*, and suppose that $\bot \notin N$. Then $I_N^{\succ} \models N$.

Proof. 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 $C' \vee A \in N$. Contradiction. $\Box$

Corollary 1.21 Let N be saturated wrt. *Res*. Then $N \models \bot \;\Leftrightarrow\; \bot \in N$.

Compactness of Propositional Logic

Theorem 1.22 (Compactness) Let N be a set of propositional formulas. Then N unsatisfiable if, and only if, there exists $M \subseteq N$, with $|M| < \infty$, and M unsatisfiable.

Proof.
"$\Leftarrow$": trivial.

"$\Rightarrow$": Let N be unsatisfiable.

$\Rightarrow$ $Res^*(N)$ unsatisfiable
$\Rightarrow$ (completeness of resolution) $\bot \in Res^*(N)$
$\Rightarrow$ $\exists n \geq 0 : \bot \in Res^n(N)$
$\Rightarrow$ $\bot$ has a finite resolution proof P;
choose M as the set of assumptions in P. $\square$

General Resolution through Instantiation

(We use RIF, resolution with implicit factorisation.) Observe that (i) upon instantiation two literals in a clause can become equal; and (ii) generally more than one instance of a clause participate in a proof.

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

Lifting Principle

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

Idea (Robinson 65):
- Resolution for general clauses
- Equality of ground atoms is generalized to unifiability of general atoms
- Only compute most general (minimal) unfiers

Significance: The advantage of the method in (Robinson 65) compared with (Gilmore 60) 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:

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

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

General resolution RIF with implicit factorization:

$$\frac{C \vee A_1 \vee \ldots \vee A_n \qquad D \vee \neg B}{(C \vee D)\sigma} \quad \text{if } \sigma = \mathsf{mgu}(A_1, \ldots, A_n, B) \quad \text{[RIF]}$$

We additionally assume that the variables in one of the two premises of the resolutions rule are (bijectively) renamed such that they become different to any variable in the other premise. 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 $:\Leftrightarrow$

$$\forall 1 \leq i \leq n : s_i\sigma = t_i\sigma.$$

If a unifier exists, E is called unifiable. If a unifier of E is more general than any other unifier of E, then we speak of a most general unifier (mgu) of E. Hereby a substitution σ is called more general than a substitution τ

$$\sigma \leq \tau \;:\Leftrightarrow\; \text{there exists a substitution } \varrho \text{ s.t. } \varrho \circ \sigma = \tau$$

where $(\varrho \circ \sigma)(x) := (x\sigma)\varrho$ is the composition of σ and ϱ als mappings.[a]

Proposition 1.23 (Exercise)

(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]Note that $\varrho \circ \sigma$ has a finite domain as required for a substitution.

Unification after Martelli/Montanari

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

MM: Main Properties

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

PROPOSITION 1.24 σ is idempotent iff $dom(\sigma) \cap codom(\sigma) = \emptyset$.

If $E = x_1 \doteq u_1, \ldots, x_k \doteq u_k$, with x_i pw. 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 1.25 If E is a solved form then σ_E is am mgu of E.

THEOREM 1.26 1. If $E \Rightarrow_{MM} E'$ then σ unifier of E iff σ unfier of E'

2. If $E \Rightarrow^*_{MM} \perp$ then E is not unifiable.

3. If $E \Rightarrow^*_{MM} E'$, with E' a 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 1.25. $\square$

Main Unification Theorem

THEOREM 1.27 E unifiable $\Leftrightarrow$ there exists a most general unifier σ of E, such that σ is idempotent and $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$.

Notation: $\sigma = \mathsf{mgu}(E)$

Problem: exponential growth of terms possible

Proof of the Unification Theorem

- Systems E irreducible wrt. $\Rightarrow_{MM}$ are either $\bot$ or a solved form.
- $\Rightarrow_{MM}$ 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 $t \notin X$.
- Therefore, reducing any E by MM with 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 1.26, Proposition 1.25).
- 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 1.28 Let C and D be variable-disjoint clauses. If

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

$$\frac{C\sigma \qquad D\varrho}{C'} \qquad \text{[propositional resolution]}$$

then there exists a substitution τ such that

$$\frac{C \qquad D}{C''} \qquad \text{[general resolution]}$$

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

Same for factorization.

Saturation of Sets of General Clauses

Corollary 1.29 Let N be a set of general clauses saturated unter 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. Wolog we may assume that clauses in N are pairwise variable-disjoint. (Otherwise make them disjoint, and this renaming process does neither change $Res(N)$ nor $G_\Sigma(N)$.)

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

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

Ad (ii): Similar. $\square$

Herbrand's Theorem

THEOREM 1.30 (HERBRAND) Let N be a set of Σ-clauses.

$$N \text{ satisfiable} \;\Leftrightarrow\; N \text{ has a Herbrand model over } \Sigma$$

Proof. "$\Leftarrow$"trivial

"$\Rightarrow$"

$$\begin{aligned}
N \not\models \bot \;&\Rightarrow\; \bot \notin \mathit{Res}^*(N) && \text{(resolution is sound)}\\
&\Rightarrow\; \bot \notin G_\Sigma(\mathit{Res}^*(N))\\
&\Rightarrow\; I_{G_\Sigma(\mathit{Res}^*(N))} \models G_\Sigma(\mathit{Res}^*(N)) && \text{(Theorem 1.20; Corollary 1.29)}\\
&\Rightarrow\; I_{G_\Sigma(\mathit{Res}^*(N))} \models \mathit{Res}^*(N) && (I \text{ is a Herbrand model})\\
&\Rightarrow\; I_{G_\Sigma(\mathit{Res}^*(N))} \models N && (N \subseteq \mathit{Res}^*(N))
\end{aligned}$$

$\square$

The Theorem of Löwenheim-Skolem

THEOREM 1.31 (LÖWENHEIM-SKOLEM) Let Σ be a countable signature and let S be a set of closed Σ-formulas. Then S is satisfiable iff S has a model over a countable universe.

Proof. S kann be at most countably infinite if both X and Σ are countable. Now generate, maintaining satisfiability, a set N of clauses from S. This extends Σ by at most countably many new Skolem functions to Σ'. As Σ' is countable, so is $T_{\Sigma'}$, the universe of Herbrand-interpretations over Σ'. Now apply Thereom 1.30. $\square$

Refutational Completeness of General Resolution

THEOREM 1.32 Let N be a set of general clauses where $\mathit{Res}(N) \subseteq N$. Then

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

Proof. Let $\mathit{Res}(N) \subseteq N$. By Corollary 1.29: $\mathit{Res}(G_\Sigma(N)) \subseteq G_\Sigma(N)$

$$\begin{aligned} N \models \bot \;&\Leftrightarrow\; G_\Sigma(N) \models \bot && \text{(Theorem 1.30)} \\ &\Leftrightarrow\; \bot \in G_\Sigma(N) && \text{(propositional resolution sound and complete)} \\ &\Leftrightarrow\; \bot \in N \end{aligned}$$

□

Compactness of Predicate Logic

THEOREM 1.33 (COMPACTNESS THEOREM FOR FIRST-ORDER LOGIC)
Let Φ be a set of first-order Formulas. Φ unsatisfiable $\Leftrightarrow$ there exists $\Psi \subseteq \Phi$, $|\Psi| < \infty$, Ψ unsatisfiable.

Proof.
"$\Leftarrow$": trivial.

"$\Rightarrow$": Let Φ be unsatisfiable and let N be the set of clauses obtained by Skolemization and CNF transformation of the formulas in Φ.

$\Rightarrow$ $Res^*(N)$ unsatisfiable
$\Rightarrow$ (Thm 1.32) $\bot \in Res^*(N)$
$\Rightarrow$ $\exists n \geq 0 : \bot \in Res^n(N)$
$\Rightarrow$ $\bot$ has finite resolution proof B of depth $\leq n$.
Choose Ψ als the subset of formulas in Φ such that the corresponding clauses contain the assumptions (leaves) of B. $\Box$

Complexity of Unification

Literature:

1. Paterson, Wegman: Linear Unification, JCSS 17, 348-375 (1978)
2. Dwork, Kanellakis, Mitchell: On the sequential nature of unification, Journal Logic Prog. 1, 35-50 (1984)
3. Baader, Nipkow: Term rewriting and all that. Cambridge U. Press 1998, Capter 4.8

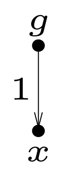
THEOREM 1.34 (PATERSON, WEGMAN 1978) Unifiability is decidable is in linear time. A most general unifiers can be computed sind in linearer time.

THEOREM 1.35 (DWORK, KANELLAKIS, MITCHELL 1984) Unifiability is log-space complete for P, that is, every problem in P can be reduced in log space to a unifiability problem.

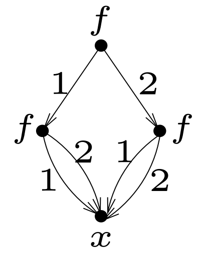
As a consequence, unifiability can, most probably, not be efficiently parallelized.

Acyclic Term Graphs

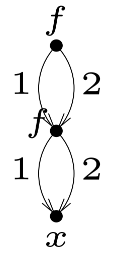
Terms and term sets as marked, ordered, acyclic graphs; each variable appears at most once



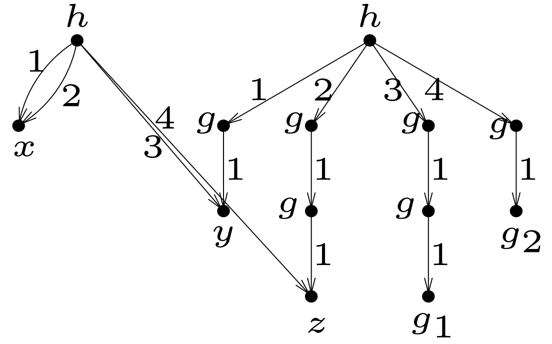
(a)



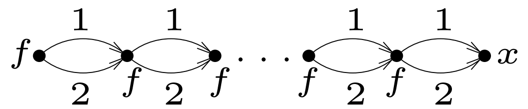
(b)



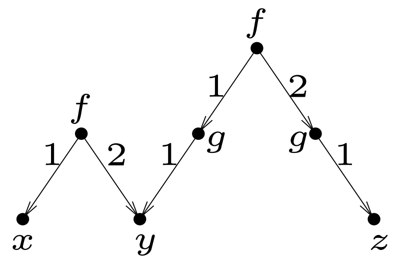
(c)



(d)



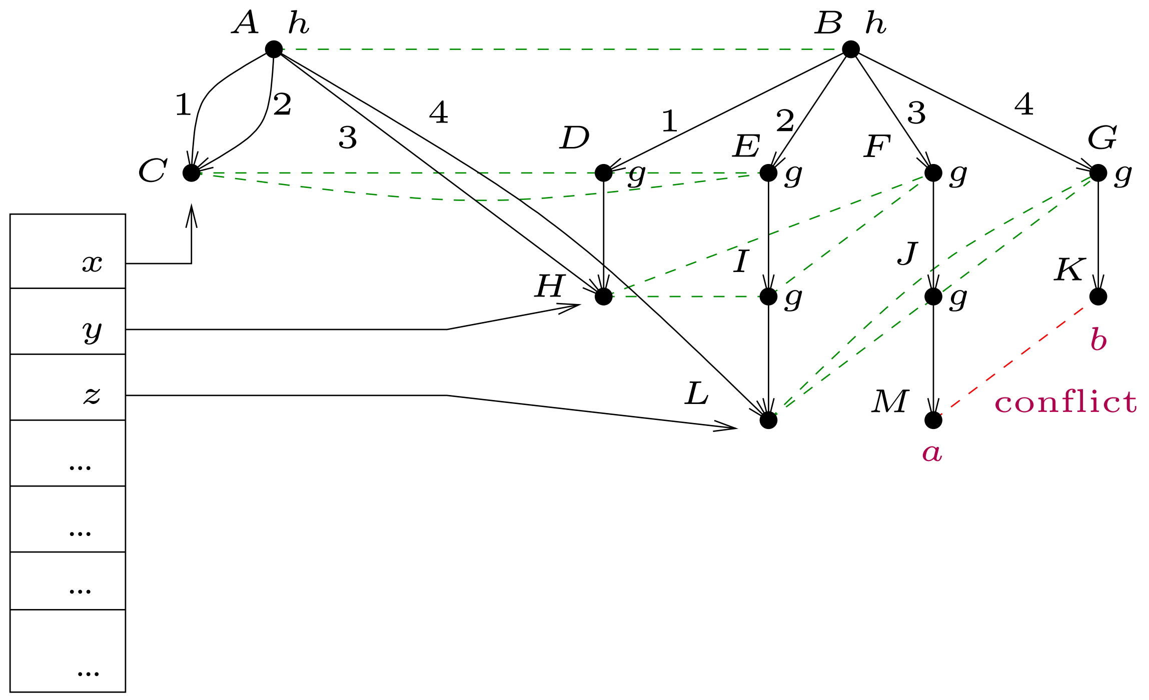
(e)



(f)

Propagation of Equality Constraints

Since variables occur at most once they don't appear as markings $m(u)$
$\Rightarrow$ binding table.



Rules (modulo symmetry of $\doteq$) for propagation of $\doteq$ in G:

$$\begin{array}{rcl} u \doteq v & \Rightarrow & u.i \doteq v.i,\ 1 \le i \le \text{arity}(u) \\ u \doteq v,\ v \doteq w & \Rightarrow & u \doteq w \\ m(u) \ne m(v) & \Rightarrow & \perp \text{ (not unifiable)} \end{array}$$

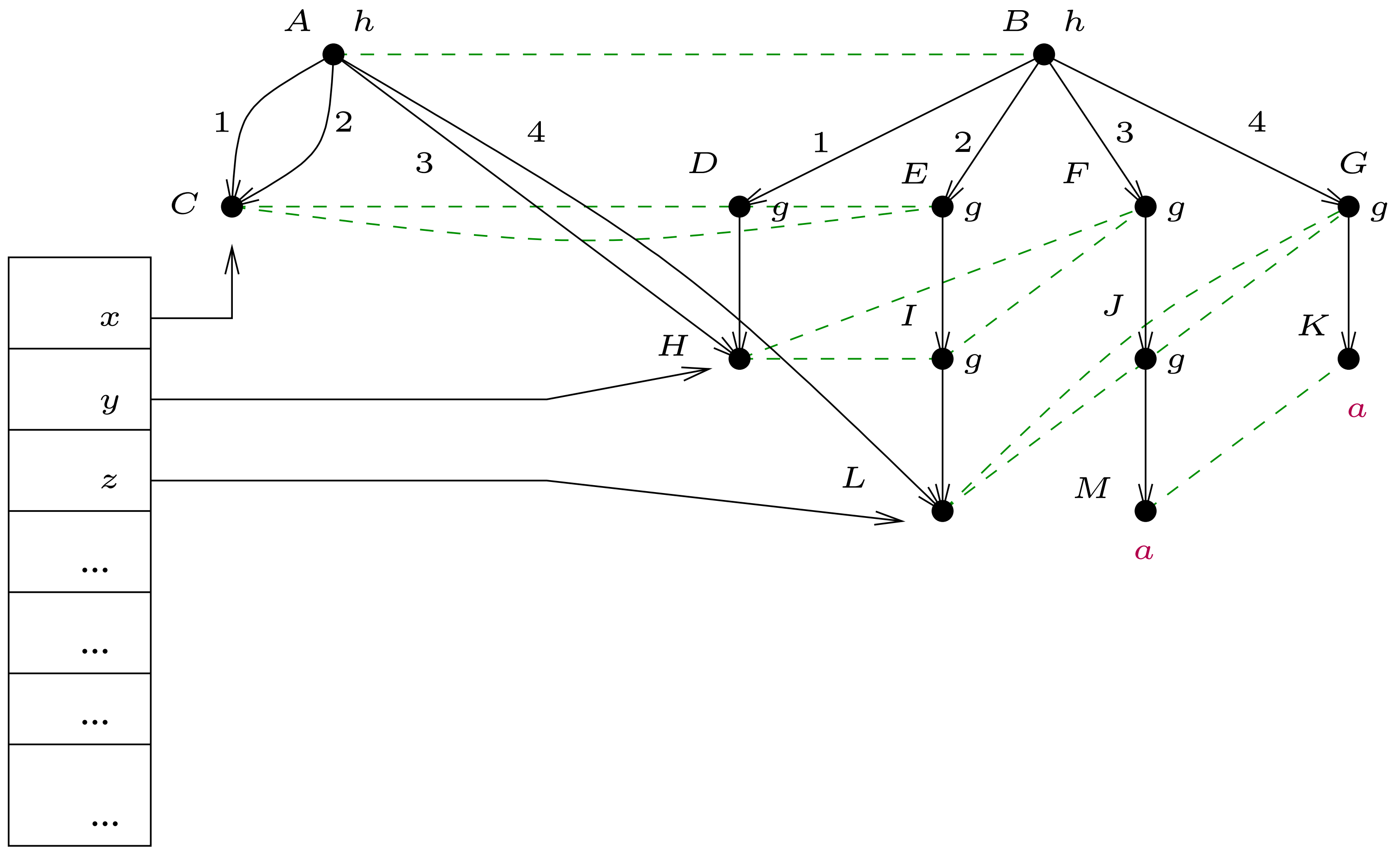
If $G/\doteq$ contains a cycle (through oriented term-subterm edges) $\Rightarrow$ not unifiable.

(Otherwise a term would have to be unified with a proper subterm of itself.)

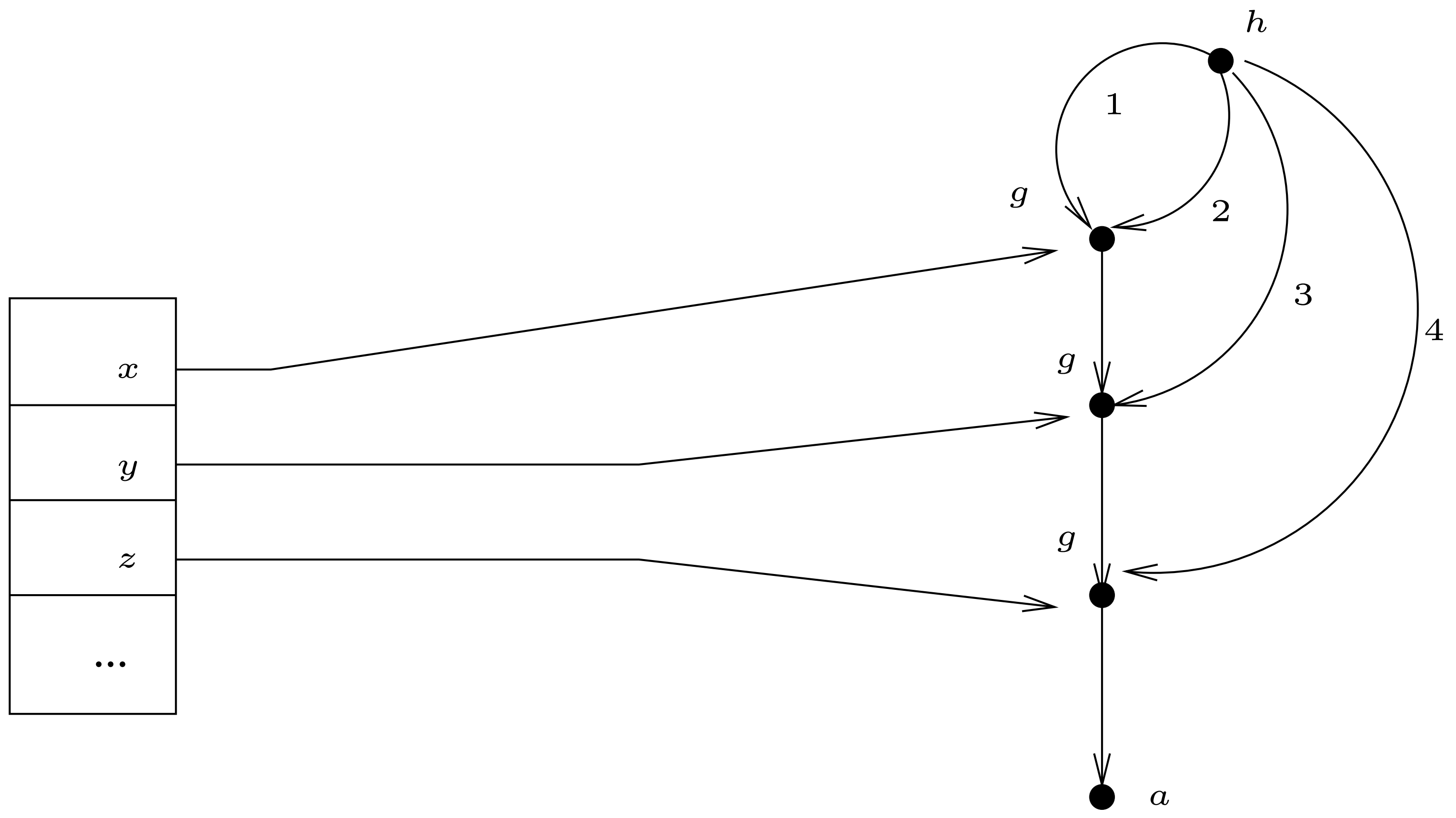
Another Example

problem $h(x, x, y, z) \doteq h(g(y), g(g(z)), g(g(a)), g(a))$

after propagation:



After Forming the Quotient



the quotient graph is cycle-free

$\Rightarrow$ $[g(g(g(a)))/x, g(g(a))/y, g(a)/z]$ is a mgu.

Analysis

For a unification problem with term graph of size n we obtain without much effort these complexity bounds:

- additional space in $O(log^2 n)$
- runtime in $O(n^3)$

In fact, at most n^2 edges can be generated by propagation, and each of those requires time $O(n)$ for a reachability test. For the quotient we have to compute the strongly connected components and then do the cycle test. This is both possible in time linear in the size of the graph, that is, in $O(n^2)$.

Matching

Let s, t be terms or atoms.

s matches t :

$s \leq t \;:\Leftrightarrow\;$ there exists a substitution σ s.t. $s\sigma = t$

(σ is called a matching substitution.)

$s \leq t \quad \Rightarrow \; \sigma = \mathsf{mgu}(s, t)$, if $var(t) \cap var(s) = \emptyset$.

Theorem 1.36 (Dwork, Kanellakis, Mitchell 1984) Matching can be efficiently parallelized.

1.9 Ordered Resolution with Selection

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

1. In the completeness proof (Model Existence Theorem 1.20) one only needs to resolve and factor maximal atoms $\Rightarrow$ order restrictions
2. Choice of negative literals don't-care $\Rightarrow$ selection

A selection function is a mapping

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

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 $Res_S^{\succ}$

Let $\succ$ be an atom ordering and S a selection function. A literal L is called [strictly] maximal wrt. a clause C $:\Leftrightarrow$ there exists a ground substitution σ such that for all L' in C: $L\sigma \succeq L'\sigma$ $[L\sigma \succ L'\sigma]$.

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

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

(i) $A\sigma$ strictly maximal wrt. $C\sigma$;

(ii) nothing is selected in C by S;

(iii) either $\neg B$ is selected,

or else nothing is selected in $\neg B \vee D$ and $\neg B\sigma$ is maximal wrt. $D\sigma$.

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

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

Special Case: Propositional Logic

For ground clauses the resolution inference simplifies to

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

if

(i) $A \succ C$;

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

(iii) $\neg A$ is selected in $D \lor \neg A$,

or else nothing is selected in $D \lor \neg A$ and $\neg A \succeq \max(D)$.

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

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)	$B \vee B$	1&3
6)	B	5
7)	$\neg A$	6&4
8)	A	6&2
9)	$\perp$	8&7

we assume $A \succ B$ and S as indicated by $\boxed{X}$; the maximal atom in a clause is depicted in red.

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

Avoiding Rotation Redundancy

From

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

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 1.37 Let C and D be variable-disjoint clauses. If

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

$$\frac{C\sigma \qquad D\rho}{C'} \qquad \text{[propositional inference in } Res_S^{\succ}\text{]}$$

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

$$\frac{C \qquad D}{C''} \qquad \text{[Inference in } Res_S^{\succ}\text{]}$$

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

Analogously for factoring.

Saturation of General Clause Sets

COROLLARY 1.38 Let N be a set of general clauses saturated under $Res^{\succ}_{S}$, i.e. $Res^{\succ}_{S}(N) \subseteq N$. Then there exists a selection function S' such that $S|_N = S'|_N$ and $G_\Sigma(N)$ is also saturated, i.e.,

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

Proof. We first define the selection function S' such that $S'(C) = S(C)$ for all clauses $C \in G_\Sigma(N) \cap N$, and for $C \in G_\Sigma(N) \setminus N$ we choose a fixed but arbitrary clause $D \in N$ mit $C \in G_\Sigma(D)$ and define $S'(C)$ to be those occurrences of literals which are the ground instances of the occurrences selected by S in D.

The rest of the proof proceeds as in the proof of Corollary 1.29 using the above lifting lemma. $\square$

Soundness and Refutational Completeness

Theorem 1.39 Let $\succ$ be an atom ordering and S a selection function such that $Res_S^{\succ}(N) \subseteq N$. Then

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

Proof. "$\Leftarrow$"trivial

"$\Rightarrow$"

(i) propositional level: construction of a candidate model 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.

(ii) general clauses: (i) + Corollary 1.38.

□

Craig-Interpolation

A theoretical application of ordered resolution is Craig-Interpolation:

THEOREM 1.40 (CRAIG 57) Let F and G be two propositional formulas such that $F \models G$. Then there exists a formula H (called the interpolant for $F \models G$), such that H contains only prop. variables occurring both in F and in G, and such that $F \models H$ and $H \models G$.

Proof. Translate F and $\neg G$ into CNF. let N and M, resp., denote the resulting clause set. Choose an atom ordering $\succ$ for which the prop. variables that occur in F but not in G are maximal. Saturate N into N^* wrt. $Res_S^{\succ}$ with an empty selection function S . Then saturate $N^* \cup M$ wrt. $Res_S^{\succ}$ saturiert to derive $\bot$. As N^* is already saturated, due to the ordering restrictions only inferences need to be considered where premises, if they are from N^*, only contain symbols that also occur in G. The conjunction of these premises is an interpolant H. $\square$

The theorem also holds for first-order formulas. For universal formulas the above proof can be easily extended. In the general case, a proof based on resolution technology is more complicated because of Skolemization.

Global Redundancy: Rules for Simplifications and Deletion

Redundancy

- many proof attempts cannot be completed to proofs: dead ends in proof search
- one proof attempt may subsume another one

Rules for simplification of TP states N (that we would like to employ)

- 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), and $C\sigma \neq D$ (subsumption is strict).

- 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 possibilites for simplification

Transition Rules for RP

Tautology elimination

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$$

if C is a tautology

Forward subsumption

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$$

if some $D \in \boldsymbol{P} \cup \boldsymbol{O}$ subsumes C

Backward subsumption

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \cup \{D\} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$$

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{D\} \quad \rhd \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$$

if C strictly subsumes D

Forward reduction

$$\boldsymbol{N} \cup \{C \vee L\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$$

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

Transition Rules for RP (II)

Backward reduction

$$\boldsymbol{N} \mid \boldsymbol{P} \cup \{C \vee L\} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$$

$$\boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{C \vee L\} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$$

if there exists $D \vee L' \in \boldsymbol{N}$ such that $\overline{L} = L'\sigma$ and $D\sigma \subseteq C$

Clause processing

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$$

Inference computation

$$\emptyset \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{C\}, \text{ mit } \boldsymbol{N} = \mathit{Res}_S^{\succ}(\boldsymbol{O} \cup \{C\})$$

Soundness and Completeness

Theorem 1.41

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

Proof in

L. Bachmair, H. Ganzinger: Resolution Theorem Proving
(on my Web page under Publications/Journals; appeared in the Handbook on Automated Theorem Proving, 2001)

Basis for the completeness proof is a formal notion of redundancy as defined subsequently.

A Formal Notion of Redundancy

Let N be a set of ground clauses and C a ground clause (not necessarliy in N).

$$C \text{ is called redundant in } N \;:\Leftrightarrow\; \text{there exists } C_1, \dots, C_n \in N,\; n \geq 0 : \\ C_i \prec C \text{ and } C_1, \dots, C_n \models C$$

Redundancy for general clauses:

$$C \text{ is called redundant in } N \;:\Leftrightarrow\; C\sigma \text{ redundant in } G_\Sigma(N), \\ \text{for all ground instances } C\sigma \text{ of } C$$

Intuition: Redundant clauses are no minimal counterexamples for any interpretation.

NB: The same ordering $\succ$ is used both for ordering restrictions and for redundancy.

Examples of Redundancy

PROPOSITION 1.42

- C tautology (i.e., $\models C$) $\quad \Rightarrow C$ redundant in any set N.
- $C\sigma \subset D \quad \Rightarrow D$ redundant in $N \cup \{C\}$
  (strict[a] Subsumption: $N \cup \{C, D\} \rhd N \cup \{C\}$)
- $C\sigma \subseteq D \quad \Rightarrow D \vee \overline{L}\sigma$ redundant in $N \cup \{C \vee L,\ D\}$

An application of the latter is reduction (subsumption resolution) in RP

[a]cf. RP for cases when clauses can be deleted even if subsumption is not strict.

Saturation up to Redundancy

N is called saturated up to redundancy (wrt. $Res_S^{\succ}$)

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

Theorem 1.43 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 model $I_N^{\succ}$ 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 1.39. □

Monotonicity Properties of Redundancy

THEOREM 1.44 (i) $N \subseteq M \;\Rightarrow\; \mathit{Red}(N) \subseteq \mathit{Red}(M)$
(ii) $M \subseteq \mathit{Red}(N) \;\Rightarrow\; \mathit{Red}(N) \subseteq \mathit{Red}(N \setminus M)$

Proof: Exercise.

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

The theorems 1.43 and 1.44 are the basis for the completeness proof of our prover RP.

Hyperresolution (Robinson 65)

We define an improved version of hyperresolution with ordering restrictions and selection. As for *Res* the calculus is parameterized by an atom ordering $\succ$ and a selection function S.

$$\frac{C_1 \lor A_1 \quad \ldots \quad C_n \lor A_n \qquad \neg B_1 \lor \ldots \lor \neg B_n \lor D}{(C_1 \lor \ldots \lor C_n \lor D)\sigma}$$

with $\sigma = \mathsf{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$, if

(i) $A_i\sigma$ strictly maximal wrt. $C_i\sigma$, $1 \le i \le n$;

(ii) nothing is selected in C_i;

(iii) the indicated occurrences of the $\neg B_i$ are exactly the ones selected by S, or else nothing is selected in the right premise and $n = 1$ and $\neg B_1\sigma$ is maximal wrt. $D\sigma$.

HR needs to be complemented by a factoring inference as for $Res_S^{\succ}$.

Hyperresolution (ctnd)

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.

There are many more variants of resolution.

We refer to [Bachmair, Ganzinger: Resolution Theorem Proving] for further reading.

1.10 Example: Neuman-Stubblebine Key Exchange Protocol

- Formalisation of a concrete application
- State-of-the-art in automated theorem proving
- Proof by consistency:

  consistency $\Rightarrow$ no unsafe states exist
- Termination requires elimination of redundancy

The Problem

Automatic Analysis of Security Protocols using SPASS: An Automated Theorem Prover for First-Order Logic with Equality

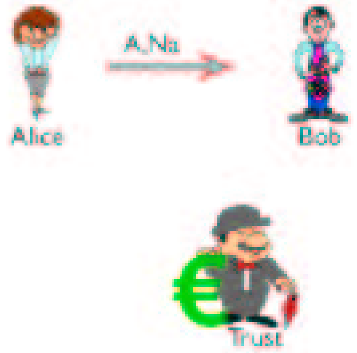
by Christoph Weidenbach

The growing importance of the internet causes a growing need for security protocols that protect transactions and communication. It turns out that the design of such protocols is highly error-prone. Therefore, there is a need for tools that automatically detect flaws like, e.g., attacks by an intruder. Here we show that our automated theorem prover SPASS can successfully be used to analyze the Newman-Stubblebine [1] key exchange protocol. To this end the protocol is formalized in logic and then the security properties are automatically analyzed by SPASS. A detailed description of the analysis can be found in [2]. The animation successively shows two runs of the Newman-Stubblebine [1] key exchange protocol. The first run works the way the protocol is designed to do, i.e., it establishes a secure key between Alice and Bob.

The Problem (ctnd)

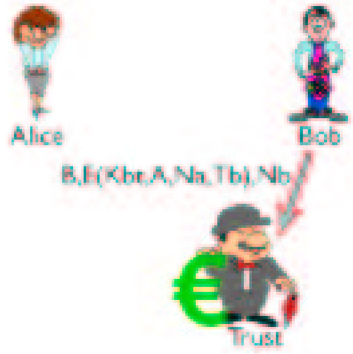
The second run shows a potential problem of the protocol. An intruder may intercept the final message sent from Alice to Bob, replace it with a different message and may eventually own a key that Bob believes to be a secure key with Alice. The initial situation for the protocol is that the two participants Alice and Bob want to establish a secure key for communication among them. They do so with the help of a trusted server Trust where both already have a secure key for communication with Trust. The below picture shows a sequence of four message exchanges that eventually establishes the key.

1



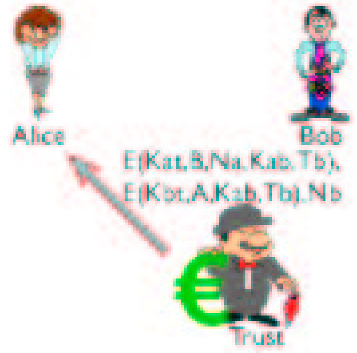
Alice sends Bob the message 1: A, Na.
It consists of her identification A together with a nonce, a random number Na. The purpose of Na is to make this run unique in order to prevent replay attacks of some intruder recording the messages.

2



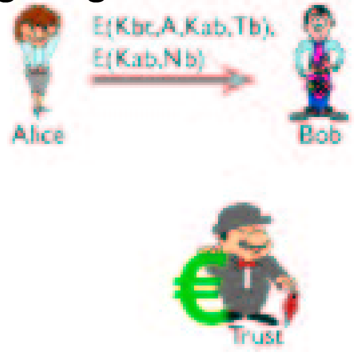
Bob sends the message 2: B, E(Kbt,A,Na,Tb), Nb to Trust.
After having received Alice's message, Bob knows that Alice wants to establish a key with him. So he sends message 2 to Trust. It consists of his identification, an encrypted middle part and again a nonce. The encrypted middle part E(Kbt,A,Na,Tb) stands for the message A,Na,Tb encrypted with the key Kbt, the secure key that Bob and the server Trust share. The time span Tb fixes the expiration time for the eventually generated key.

3



Trust sends the message 3: E(Kat,B,Na,Kab,Tb), E(Kbt,A,Kab,Tb), Nb to Alice. Trust reads the previous message, generates the new secure key Kab for communication between Alice and Bob and sends message 3 to Alice. The first part E(Kat,B,Na,Kab,Tb) can be decrypted by Alice, whereas the second part is meant to be forwarded by Alice to Bob, see message 4.

Alice sends the message 4: E(Kbt,A,Kab,Tb), E(Kab,Nb) to Bob. Alice reads message 3, decrypts the first part with the secure key Kat she shares with Trust and extracts the content of the message. In particular, the new key Kab for communication with Bob. Then she forwards the second part of Trust's message together with E(Kab,Nb) to Bob.



4

5

Bob receives message 4, decrypts the first part and extracts the key Kab, uses this key to decrypt the second part and by inspecting Nb he authentificates Alice. Eventually, Alice and Bob share the key Kab. Everything is fine.

Neuman-Stubblebine

1. $$\begin{tikzcd} A \arrow[r, "{A, N_a}"] & B \end{tikzcd}$$

2. $$\begin{tikzcd} B \arrow[r, "{B, N_b, E_{K_{bt}}(A, N_a, T_b)}"] & T \end{tikzcd}$$

3. $$\begin{tikzcd} T \arrow[r, "{E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b}"] & A \end{tikzcd}$$

4. $$\begin{tikzcd} A \arrow[r, "{E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)}"] & B \end{tikzcd}$$

What can happen?

How can an intruder now break this protocol? The key K_{ab} is only transmitted inside encrypted parts of messages and we assume that an intruder cannot break any keys nor does he know any of the initial keys K_{at} or K_{bt}. Here is the solution:

1

Alice
E(Kb A Kab,Tb),
E(Kab Nb)
Bob
Intruder
Trust

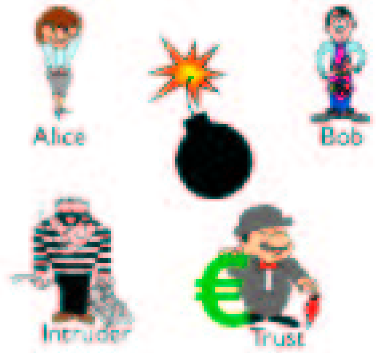
Intruder intercepts the message

2

Alice
Bob
E(Kbt,A,Na,Tb),
E(Na,Nb)
Intruder
Trust

Intruder sends the message E(Kbt,A,Na,Tb),E(Na,Nb) to Bob. The problem is that the part E(Kbt,A,Kab,Tb) of message 4 and the part E(Kbt,A,Na,Tb) of message 2 are nearly identical. The only difference is that at the position of Kab in message 4, there is the nonce Na in message 2. The intruder can of course record all messages and intercept/send messages. So he catches message 4 and instead he sends message 5 to Bob.

3



Bob decrypts message 5 and now believes that Na is a secure key he shares with Alice.

4

Bob starts communication with Alice,

5

but talks to Intruder.

Breaking Neuman-Stubblebine

$$\begin{tikzcd}
1. & A \arrow[r, "{A, N_a}"] & B \\
2. & B \arrow[r, "{B, N_b, E_{K_{bt}}(A, N_a, T_b)}"] & T \\
3. & T \arrow[r, "{E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b}"] & A \\
4. & A \arrow[r, "{E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)}"] & B \\
3. & I \arrow[r, "{E_{K_{bt}}(A, N_a, T_b), E_{N_a}(N_b)}"] & B
\end{tikzcd}$$

The Formalisation

The key idea of the formalization is to describe the set of sent messages. This is done by introducing a monadic predicate M in first-order logic. Furthermore, every participant holds its set of known keys, represented by the predicates Ak for Alice's keys, Bk for Bob's keys, Tk for Trust's keys and Ik for the keys the intruder knows. The rest of the used symbols is introduced and explained with the first appearance in a formula. Then the four messages can be translated into the following formulae:

Step 1) A, Na

$$Ak(key(at, t)) \tag{1}$$

$$M(sent(a, b, pair(a, na))) \tag{2}$$

The two formulae express that initially Alice holds the key at for communication with t (for Trust) and that she sends the first message. In order to formalize messages we employ a three place function sent where the first argument is the sender, the second the receiver and the third the content of the message. So the constant a represents Alice, b Bob, t Trust and i Intruder. The functions *pair* (*triple*, *quadr*) simply form sequences of messages of the indicated length.

Step 2) $B, E(Kbt, A, Na, Tb), Nb$

$$Bk(key(bt, t)) \tag{3}$$

$$\begin{aligned}&\forall xa, xna\ [M(sent(xa, b, pair(xa, xna)))\\ &\qquad \rightarrow M(sent(b, t, triple(b, nb(xna), encr(triple(xa, xna, tb(xna)), bt))))\end{aligned} \tag{4}$$

Bob holds the key *bt* for secure communication with Trust and whenever he receives a message of the form of message 1 (formula (2)), he sends a key request to Trust according to message 2. Note that encryption is formalized by the two place function *encr* where the first argument is the date and the second argument the key. Every lowercase symbol starting with an x denotes a variable. The functions *nb* and *tb* generate, respectively, a new nonce and time span out of *xa*'s (Alice's) request represented by her nonce *xna*.

Step 3) $E(Kat, B, Na, Kab, Tb),\ E(Kbt, A, Kab, Tb),\ Nb$

$$Tk(key(at,a))) \wedge Tk(key(bt,b)) \tag{5}$$

$$\begin{aligned}
&\forall xb, xnb, xa, xna, xbet, xbt, xat, xk \\
&\qquad [\ (M(sent(xb,t,triple(xb,xnb,encr(triple(xa,xna,xbet),xbt)))) \\
&\qquad\ \wedge Tk(key(xbt,xb)) \\
&\qquad\ \wedge Tk(key(xat,xa))) \\
&\qquad\ \rightarrow M(sent(t,xa,triple(encr(quadr(xb,xna,kt(xna),xbet),xat), \\
&\qquad\qquad encr(triple(xa,kt(xna),xbet),xbt),xnb)))\ ]
\end{aligned} \tag{6}$$

Trust holds the keys for Alice and Bob and answers appropriately to a message in the format of message 2. Note that decryption is formalized by unification with an appropriate term structure where it is checked that the necessary keys are known to Trust. The server generates the key by applying his key generation function kt to the nonce xna.

Step 4) $E(Kbt, A, Kab, Tb),\ \ E(Kab, Nb)$

$$\forall xnb, xbet, xk, xm, xb, xna$$
$$[\ M(sent(t, a, triple(encr(quadr(xb, xna, xk, xbet), at), xm, xnb)) \quad (7)$$
$$\rightarrow (M(sent(a, xb, pair(xm, encr(xnb, xk)))) \wedge Ak(key(xk, xb)))\ ]$$

$$\forall xbet, xk, xnb, xa, xna$$
$$[\ M(sent(xa, b, pair(encr(triple(xa, xk, tb(xna)), bt), encr(nb(xna), xk)))$$
$$\rightarrow Bk(key(xk, xa))] \quad (8)$$

Finally, Alice answers according to the protocol to message 3 and stores the generated key for communication, formula (7). Formula (8) describes Bob's behaviour when he receives Alice's message. Bob decodes this message and stores the new key as well.

A's Formalization Part I

$$\begin{tikzcd} \boxed{A} \arrow[r, "{A, N_a}"] & \boxed{B} \end{tikzcd}$$

$\longrightarrow P(a)$

$\longrightarrow Ak(key(at, t))$

$\longrightarrow M(sent(a, b, pair(a, na)))$

$\longrightarrow Sa(pair(b, na))$

A's Formalization Part II

$$\begin{tikzcd}
T \arrow[r, "{E_{K_{at}}(B, N_a, K_{ab}, T_b), E_{K_{bt}}(A, K_{ab}, T_b), N_b}"] & A \\
A \arrow[r, "{E_{K_{bt}}(A, K_{ab}, T_b), E_{K_{ab}}(N_b)}"] & B
\end{tikzcd}$$

$$
\begin{gathered}
M(sent(t, a, triple(encr(quadr(xb, xna, xk, xbet), at), xm, xnb))), \\
Sa(pair(xb, xna)) \\
\longrightarrow \\
M(sent(a, xb, pair(xm, encr(xnb, xk)))), \\
Ak(key(xk, xb))
\end{gathered}
$$

NB: Variables (all implicitly universally quantified) start with letter “x”.

The Intruder

The Intruder is modeled as an exhaustive hacker. He records all messages, decomposes the messages as far as possible and generates all possible new compositions. Furthermore, any object he has at hand is considered as a key and tried to used for encryption as well as for decryption. All these messages are posted. The set of messages the intruder has available is represented by the predicate Im.

The participants are Alice, Bob, Trust and Intruder:

$$P(a) \wedge P(b) \wedge P(t) \wedge P(i) \tag{9}$$

The intruder records all messages:

$$\forall xa, xb, xm\ [M(sent(xa, xb, xm)) \rightarrow Im(xm)] \tag{10}$$

He decomposes and decrypts all messages he owns the key for:

$$\forall u, v \ [Im(pair(u, v)) \rightarrow Im(u) \wedge Im(v)] \tag{11}$$

$$\forall u, v, w \ [Im(triple(u, v, w)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w)] \tag{12}$$

$$\forall u, v, w, z \ [Im(quadr(u, v, w, z)) \rightarrow Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z)] \tag{13}$$

$$\forall u, v, w \ [Im(encr(u, v)) \wedge Ik(key(v, w)) \rightarrow Im(u)] \tag{14}$$

He composes all possible messages:

$$\forall u, v \ [Im(u) \wedge Im(v) \rightarrow Im(pair(u, v))] \tag{15}$$

$$\forall u, v, w \ [Im(u) \wedge Im(v) \wedge Im(w) \rightarrow Im(triple(u, v, w))] \tag{16}$$

$$\forall u, v, w, x \ [Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(x) \rightarrow Im(quadr(u, v, w, x))] \tag{17}$$

He considers every item to be a key and uses it for encryption:

$$\forall v, w \ [Im(v) \wedge P(w) \rightarrow Ik(key(v, w))] \tag{18}$$

$$\forall u, v, w \ [Im(u) \wedge Ik(key(v, w)) \wedge P(w) \rightarrow Im(encr(u, v))] \tag{19}$$

He sends everything:

$$\forall x, y, u \ [P(x) \wedge P(y) \wedge Im(u) \rightarrow M(sent(x, y, u))] \tag{20}$$

Finally we must formalize the insecurity requirement. Intruder must not have any key for communication with Bob that Bob believes to be a secure key for Alice:

$$\exists x \; [Ik(key(x,b)) \wedge Bk(key(x,a))]$$

The Intruder Part I

$$\vdots$$

$$M(sent(xa, xb, xm)) \rightarrow Im(xm)$$

$$\vdots$$

$$Im(pair(x1, x2)) \rightarrow Im(x1)$$

$$Im(pair(x1, x2)) \rightarrow Im(x2)$$

$$\vdots$$

$$Im(x1), Im(x2) \rightarrow Im(pair(x1, x2))$$

$$\vdots$$

The Intruder Part II

$$\vdots$$

$$P(x1), P(x2), Im(x3) \rightarrow Im(sent(x1, x2, x3))$$

$$\vdots$$

$$P(x1), Im(x2) \rightarrow Ik(key(x1, x2))$$

$$Im(x1), Ik(key(x2, x3)) \rightarrow Im(encr(x1, x2))$$

SPASS solves the problem

Now the protocol formulae (1)-(8) together with the intruder formulae (9)-(20) and the insecurity formula above can be given to SPASS. Then SPASS automatically proves that this formula holds and that the problematic key is the nonce Na. The protocol can be repaired by putting type checks on the keys, such that keys can no longer be confused with nonces. This can be added to the SPASS first-order logic formalization. Then SPASS disproves the insecurity formula above. This capability is currently unique to SPASS. Although some other provers might be able to prove that the insecurity formula holds in the formalization without type checks, we are currently not aware of any prover that can disprove the insecurity formula in the formalization with type checking. Further details can be found in [2], below. The experiment is available in full detail from the SPASS home page in the download area.

References:

[1] Neuman, B. C. and Stubblebine, S. G., 1993, A note on the use of timestamps as nonces, ACM SIGOPS, Operating Systems Review, 27(2), 10-14.

[2] Weidenbach, C., 1999, Towards an automatic analysis of security protocols in

first-order logic, in 16th International Conference on Automated Deduction, CADE-16, Vol. 1632 of LNAI, Springer, pp. 378-382.

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; approximative solving of ordering constraints on the non-ground level
- completeness proof by contructing candidate models from reductive 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

1.11 Semantic Tableaux

analytic: inferences according to the logical content of the symbols
goal oriented: inferences operate directly on the goal to be proved
global: some inferences affect the entire proof state (set of formulas)

Literature: Fitting book, chapt. 3, 6, 7.
R.M. Smullyan: First-Order Logic, Dover Publ., New York, 1968, revised 1995.

Like resolution, semantic tableaux were developed in the sixties, by R.M. Smullyan,[a] on the basis of work by Gentzen in the 30ies and of Beth in the 50ies.

[a] According to Fitting, semantic tableaux were first proposed by the Polish scientist Z. Lis in a paper in Studia Logica 10, 1960 that was only recently rediscovered.

A Tableau for $\{P \downarrow (Q \vee R),\ \neg(Q \wedge \neg R)\}$

$$\begin{tikzcd}
 & 1.\quad P \downarrow (Q \vee R) & \\
 & 2.\quad \neg(Q \wedge \neg R) \arrow[dl, dash] \arrow[dr, dash] & \\
3.\quad \neg Q & & 4.\quad \neg\neg R \\
6.\quad \neg P & & 5.\quad R \\
7.\quad \neg(Q \vee R) & & \\
8.\quad \neg Q & & \\
9.\quad \neg R & &
\end{tikzcd}$$

This tableau is not “maximal”, however the first “path” is. This path is not “closed”, hence the set $\{1, 2\}$ is satisfiable. (These notions will all be defined below.)

Classification of Formulas

conjunctive			disjunctive		
α	α_1	α_2	β	β_1	β_2
$X \wedge Y$	X	Y	$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$\neg(X \vee Y)$	$\neg X$	$\neg Y$	$X \vee Y$	X	Y
$\neg(X \rightarrow Y)$	X	$\neg Y$	$X \rightarrow Y$	$\neg X$	Y
$\neg(X \leftarrow Y)$	$\neg X$	Y	$X \leftarrow Y$	X	$\neg Y$
$\neg(X \uparrow Y)$	X	Y	$X \uparrow Y$	$\neg X$	$\neg Y$
$X \downarrow Y$	$\neg X$	$\neg Y$	$\neg(X \downarrow Y)$	X	Y

"$\uparrow$" is "nand", "$\downarrow$" is "nor".

Subsequently we will only use the binary connectives listed above. The others we can eliminate.

Propositional Expansion Rules

The rules are applied to the formulas in a tableau and expand the tableau at a leaf. We append the conclusions of a rule (horizontally or vertically) at a leaf, whenever the premise of the expansion rule matches a formula appearing anywhere on the path from the root to that leaf.

Negation Elimination

$$\frac{\neg\neg F}{F} \qquad \frac{\neg\top}{\bot} \qquad \frac{\neg\bot}{\top}$$

α-Expansion (append α_1 and α_2 one on top of the other)

$$\begin{array}{c} \alpha \\ \hline \alpha_1 \\ \alpha_2 \end{array}$$

β-Expansion (append β_1 and β_2 horizontally; branch into β_1 and β_2)

$$\frac{\beta}{\beta_1 \mid \beta_2}$$

Tableaux: Notions

A semantic tableau is a marked (by formulas), finite, unordered tree and inductively defined as follows: Let $\{F_1, \ldots, F_n\}$ be a set of formulas.

(i) The tree consisting of a single path[a]

$$\begin{array}{c} F_1 \\ F_2 \\ \vdots \\ F_n \end{array}$$

is a tableau for $\{F_1, \ldots, F_n\}$.

(ii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if T' results from T by applying an expansion rule then T' is also a tableau for $\{F_1, \ldots, F_n\}$.

A path (from the root to a leaf) in a tableau is called closed, if it either contains $\perp$, or else it contains both some formula F and its negation $\neg F$. Otherwise the path is called open.

[a]We often do not draw tree edges leaving nodes having a single successor only.

Tableaux: Notions (II)

A tableau is called closed, if all paths are closed.

A tableau proof for F is a closed tableau for $\{\neg F\}$.

A path P in a tableau is called maximal, if for each non-atomic formula F on P there exists a node in P at which the expansion rule for F has been applied. In that case, if F is a formula on P, P also contains:

(i) F_1 and F_2, if F is a α-formula,
(ii) F_1 or F_2, if F is a β-formula, and
(iii) F', if F is a negation formula, and F' the conclusion of the corresponding elimination rule.

A tableau is called maximal, if each path is closed or maximal.

A tableau is called strict, if for each formula the corresponding expansion rule has been applied at most once on each path containing that formula.

A tableau is called clausal, if each of its formulas is a clause.

A Sample Proof

One starts out from the neegation of the formula to be proved.

1.	$\neg[(P \to (Q \to R)) \to ((P \vee S) \to ((Q \to R) \vee S))]$	
2.	$(P \to (Q \to R))$	$[1_1]$
3.	$\neg((P \vee S) \to ((Q \to R) \vee S))$	$[1_2]$
4.	$P \vee S$	$[3_1]$
5.	$\neg((Q \to R) \vee S))$	$[3_2]$
6.	$\neg(Q \to R)$	$[5_1]$
7.	$\neg S$	$[5_2]$

$$\begin{tikzcd}
 & & \neg S \arrow[dl, dash] \arrow[dr, dash] & \\
 & 8.\ \neg P\ [2_1] \arrow[dl, dash] \arrow[d, dash] & & 9.\ Q \to R\ [2_2] \\
10.\ P\ [4_1] & 11.\ S\ [4_2] & &
\end{tikzcd}$$

There are three paths, each of them closed.

Properties of Propositional Tableaux

We assume that T is a tableau for $\{F_1, \ldots, F_n\}$.

Theorem 1.45 $\{F_1, \ldots, F_n\}$ satisfiable $\Leftrightarrow$ some path (i.e., the set of its formulas) in T is satisfiable.

(Proof by induction over the structure of T.)

Corollary 1.46 T closed $\Rightarrow \{F_1, \ldots, F_n\}$ unsatisfiable

Theorem 1.47 Let T be a strict tableau. Then T is finite.

Proof. New formulas resulting from expansion are either $\bot$, $\top$ or subformulas of the expanded formula. By strictness, on each path a formula can be expanded at most once. Therefore, each path is finite, and a finitely branching tree with finite paths is finite (Königs Lemma). $\Box$

Conclusion: Strict and maximal tableaux can be effectively constructed.

Refutational Completeness

THEOREM 1.48 Let P be a maximal, open path in a tableau. Then set of formulas on P is satisfiable.

Proof. [for the case of a clausal tableau] Let N be the set of formulas on P. As P is open, $\bot$ is not in N. Let $C \vee A$ and $D \vee \neg A$ be two resolvable clauses in N. One of the two sublauses C or D, C say, is not empty, as otherwise P would be closed. Since P is maximal, in P the β-rule was applied on $C \vee A$. Therefore, P (and N) contains a proper subclause of $C \vee A$, and hence $C \vee A$ is redundant in N. By the same reasoning, if N contains a clause that can be factored, that clause must be redundant in N. In other words, N is saturated up to redundancy wrt. Res(olution). Now apply Theorem 1.20 to prove satisfiability of N. $\Box$

THEOREM 1.49 $\{F_1, \ldots, F_n\}$ satisfiable $\Leftrightarrow$ there exists no closed strict tableau for $\{F_1, \ldots, F_n\}$.

Proof. One direction is clear by Theorem 1.45. For the reverse direction, let T be a strict, maximal tableau for $\{F_1, \ldots, F_n\}$ and let P be an open path in T. By the previous Theorem the set of formulas on P, and hence by Theorem 1.45 the set $\{F_1, \ldots, F_n\}$, is satisfiable. $\Box$

Consequences

The validity of a propositional formula F can be established by constructing a strict, maximal tableau for $\{\neg F\}$:

- T closed $\Leftrightarrow$ F valid.
- It suffices to test complementarity of paths wrt. atomic formulas (cf. reasoning in the proof of Theorem 1.48).
- Which of the potentially many strict, maximal tableaux one computes does not matter. In other words, tableau expansion rules can be applied don't-care non-deterministically ("proof confluence").
- The expansion strategy, however, can have a dramatic impact on tableau size.
- Since it is sufficient to saturate paths wrt. ordered resolution (up to redundancy), tableau expansion rules can be even more restricted, in particular by certain ordering constraints.

Semantic Tableaux for First-Order Logic

Additional classification of quantified formulas:

universal		existential	
γ	$\gamma(t)$	δ	$\delta(t)$
$\forall x F$	$F[t/x]$	$\exists x F$	$F[t/x]$
$\neg\exists x F$	$\neg F[t/x]$	$\neg\forall x F$	$\neg F[t/x]$

Moreover we assume that the set of variables X is partitioned into 2 disjoint infinite subsets X_g and X_f, so that bound [free] variables variables can be chosen from X_g $[X_f]$. (This avoids the variable capturing problem.)

Additional Expansion Rules

γ-expansion

$$\frac{\gamma}{\gamma(x)} \qquad \text{where } x \text{ is a variable in } X_f$$

δ-expansion

$$\frac{\delta}{\delta(f(x_1, \ldots, x_n))}$$

where f is a new Skolem function, and the x_i are the free variables in δ

Skolemisation becomes part of the calculus and needs not necessarily be applied in a preprocessing step. Of course, one could do Skolemization beforehand, and then the δ-rule would not be needed.

Note that the rules are parametric, instantiated by the choices for x and f, respectively. Strictness here means that only one instance of the rule is applied on each path to any formula on the path.

In this form the rules go back to Hähnle and Schmitt: The liberalized δ-rule in free variable semantic tableaux, J. Automated Reasoning 13,2, 1994, 211–221.

Definition: Free-Variable Tableau

Let $\{F_1, \ldots, F_n\}$ be a set of closed formulas.

(i) The tree consisting of the single path $\begin{array}{c} F_1 \\ \vdots \\ F_n \end{array}$ is a tableau for $\{F_1, \ldots, F_n\}$.

(ii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if T' results by applying an expansion rule to T, then T' is also a tableau for $\{F_1, \ldots, F_n\}$.

(iii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if σ is a substitution, then $T\sigma$ is also a tableau for $\{F_1, \ldots, F_n\}$.

The substitution rule (iii) may, potentially, modify all the formulas of a tableau. This feature is what is makes the tableau method a global proof method. (Resolution, by comparison, is a local method.)

Of one took (iii) literally, by repeated application of γ-rule one can enumerate all substitution instances of the universally quantified formulas. That would be a major drawback compared with resolution. Fortunately, we can improve on this.

Example

1.	$\neg[\exists w \forall x\ p(x, w, f(x, w)) \rightarrow \exists w \forall x \exists y\ p(x, w, y)]$	
2.	$\exists w \forall x\ p(x, w, f(x, w))$	1_1 $[\alpha]$
3.	$\neg \exists w \forall x \exists y\ p(x, w, y)$	1_2 $[\alpha]$
4.	$\forall x\ p(x, a, f(x, a))$	$2(a)$ $[\delta]$
5.	$\neg \forall x \exists y\ p(x, v_1, y)$	$3(v_1)$ $[\gamma]$
6.	$\neg \exists y\ p(b(v_1), v_1, y)$	$5(b(v_1))$ $[\delta]$
7.	$p(v_2, a, f(v_2, a))$	$4(v_2)$ $[\gamma]$
8.	$\neg p(b(v_1), v_1, v_3)$	$6(v_3)$ $[\gamma]$

7. and 8. are complementary (modulo unification):

$$v_2 \doteq b(v_1),\ \ a \doteq v_1,\ \ f(v_2, a) \doteq v_3$$

is solvable with an mgu $\sigma = [a/v_1, b(a)/v_2, f(b(a), a)/v_3]$, and, hence, $T\sigma$ is a closed (linear) tableau for the formula in 1.

AMGU-Tableaux

Idea: Restrict the substitution rule to unfiers of complementary formulas.

We speak of an AMGU-Tableau, whenever the substitution rule is only applied for substitutions σ for which there is a path in T containing two literals $\neg A$ and B such that $\sigma = \mathsf{mgu}(A, B)$.

Correctness

Given an signature Σ, by Σ^{sko} we denote the result of adding infinitely many new Skolem function symbols which we may use in the δ-rule.

Let $\mathcal{A}$ be a Σ^{sko}-interpretation, T a Tableau, and β a variable assignment over A. T is called $(\mathcal{A}, \beta)$-valid, if there is a path P_β in T such that $A, \beta \models F$, for each formula F on P_β. T is called satisfiable if there exists a structure $\mathcal{A}$ such that for each assignment β the tableau T is $(\mathcal{A}, \beta)$-valid. (This implies that we may choose P_β depending on β.)

Let F_i be closed Σ-formulas.

Theorem 1.50 Let T be a tableau for $\{F_1, \ldots, F_n\}$.
$\{F_1, \ldots, F_n\}$ is satisfiable $\Leftrightarrow$ T is satisfiable.

(Proof of " $\Rightarrow$ " by induction over the depth of T. For δ one needs to reuse the ideas for proving that Skolemization preserves [un-]satisfiability.)

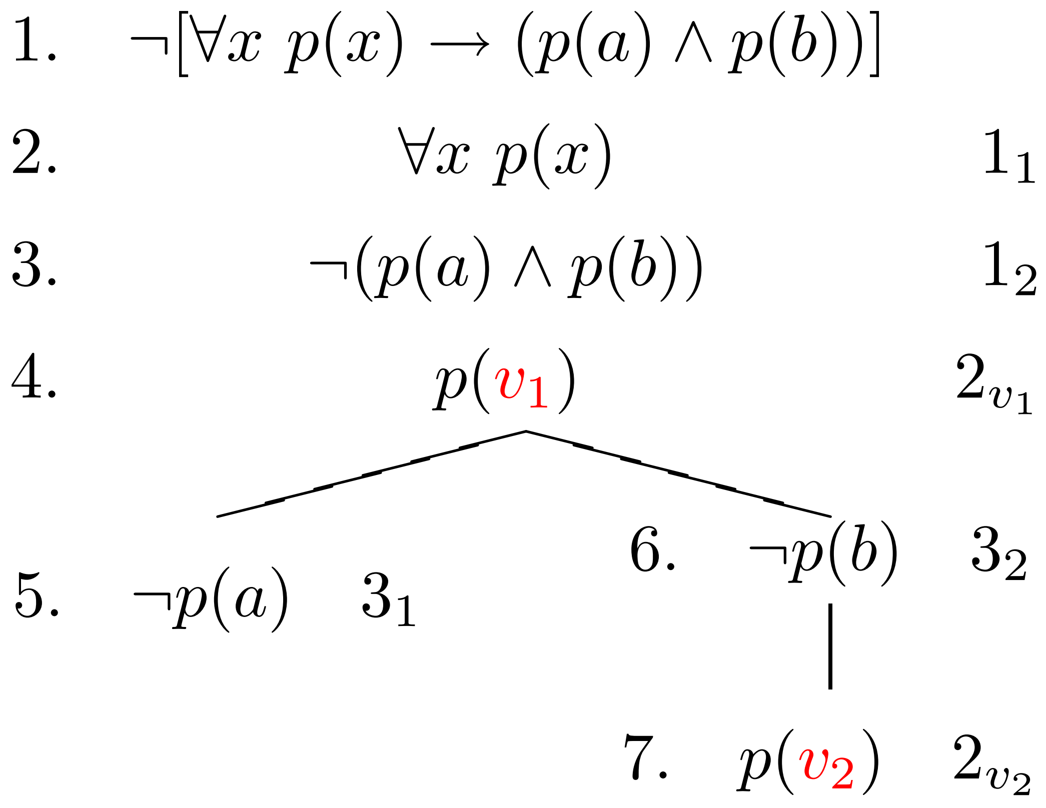
Incompleteness of Strictness

$$\begin{array}{lcr}
1. & \neg[\forall x\ p(x) \rightarrow (p(a) \wedge p(b))] & \\
2. & \forall x\ p(x) & 1_1 \\
3. & \neg(p(a) \wedge p(b)) & 1_2 \\
4. & p({\color{red}v_1}) & 2(v_1)
\end{array}$$

$$\begin{array}{lll|lll}
5. & \neg p(a) & 3_1 & 6. & \neg p(b) & 3_2
\end{array}$$

If we placed a strictness requirement also on applications of γ, the tableau would only be expandable by the substitution rule. However, there is no substitution (for v_1) that can close both paths simultaneously.

Multiple Applications of γ Solves the Problem


1. $\neg[\forall x\, p(x) \rightarrow (p(a) \wedge p(b))]$
2. $\forall x\, p(x)$ 1_1
3. $\neg(p(a) \wedge p(b))$ 1_2
4. $p(v_1)$ 2_{v_1}
5. $\neg p(a)$ 3_1
6. $\neg p(b)$ 3_2
7. $p(v_2)$ 2_{v_2}

The point is that different applications of γ to $\forall x\, p(x)$ may employ different free variables for x.

Now, by two applications of the AMGU-rule, we obtain the substitution $[a/v_1, b/v_2]$ closing the tableau.

How Often Do we Have to Apply γ?

Theorem 1.51 There is no recursive function $f : F_\Sigma \times F_\Sigma \rightarrow \mathbb{N}$ such that, if the closed formula F is unsatisfiable, then there exists a closed tableau for F where to all formulas $\forall x G$ appearing in T the γ-rule is applied at most $f(F, \forall x G)$ times on each path containing $\forall x G$.

Otherwise unsatisfiablility or, respectively, validity for first-order logic would be decidable. In fact, one would be able to enumerate in finite time all tableaux bounded in depth as indicated by f. In other words, free-variable tableaux are not recursively bounded in their depth.

Again $\forall$ is treated like an infinite conjunction. By repeatedly applying γ, together with the substitution rule, one is capable of enumerating all instances $F[t/x]$ vertically, that is, conjunctively, in each path containing $\forall x F$.

Refutational Completeness

Therefore strictness for γ should from now on mean that each instance of γ (depending on the choice of the free variable) is applied at most once to each γ-formula on any path.

THEOREM 1.52 $\{F_1, \ldots, F_n\}$ satisfiable $\Leftrightarrow$ there exists no closed, strict AMGU-Tableau for $\{F_1, \ldots, F_n\}$.

For the proof one defines a fair tableau expansion process converging against an infinite tableau where on each path each γ-formula is expanded into all its variants (modulo the choice of the free variable).

One may then again show that each path in that tableau is saturated (up to redundancy) by resolution. This requires to apply the lifting lemma for resolution in order to show completeness of the AMGU-restriction.

Semantic Tableaux vs. Resolution

- Both methods are machine methods on which todays provers are based upon.
- tableaux: global, goal-oriented, “backward”
- resolution: local, “forward”
- Resolution can be combined with more powerful redundancy elimination methods.
- Like resolution, the tableau method, in order to be useful in practice, must be accompanied by refinements: lemma generation, ordering restrictions, efficient term and proof data structures
- Because of its global nature redundancy elimination is more difficult for the tableau method.
- Resolution can be refined to work well with equality and algebraic structures; tableaux cannot (it seems).

Other Inference Systems

Hilbert Calculus: direct proof method, synthetic (axioms + modus ponens), not suitable for neither humans nor machines

Natural Deduction (Prawitz): models the concept of proofs from assumptions as humans do it (cf. Huth/Ryan-book)

Sequent Calculus (Gentzen): assumptions internalized into the data structure of sequents; a kind of mixture between natural deduction and semantic tableaux; perfect symmetry between the handling of assumptions and their consequences; can be used both backwards and forwards,

Davis/Putnam/Loveland/Logeman procedure: well engineered method for propositional satisfiability testing

Binary Decision Diagrams: data structure for the efficient representation of Boolean functions (cf. below); works often well for equivalence and tautology testing