CS612: Automated Reasoning, Part II
Advanced Topics

Renate Schmidt

School of Computer Science
University of Manchester

schmidt@cs.man.ac.uk

http://www.cs.man.ac.uk/~schmidt/CS612/

Overview

- Handouts: Assessed teaching week exercises
  Unassessed teaching week exercises
  Post-coursework
  Using MSPASS
  Timetable (note: approximate!)
- Specialist assistance: Konstantin Korovin
- Open book exam

Content

- Emphasis on:
  - foundation of advanced automated theorem proving
  - both theory and practice (more theoretical, but motivated by considerations of practical aspects and efficiency)
- Treatment is formal and rigorous; small selection of important topics; many examples and exercises
- **All you need to know for the exam is in the lecture notes, and it is advisable that you can solve the exercises and can do the assignments**
- Good strategy: go through the material discussed in lectures after each lecture, do as many of the unassessed exercises, do the assignments and read ahead. And, ask questions!

Textbooks

- None cover all the material, but recommended are:
  - Schöning, U. (1989), *Logic for Computer Scientists.* Birkhäuser.
  - Fitting, M. (1990), *First-Order Logic and Automated Theorem Proving*. Springer.
- Also useful:
  - Socher-Ambrosius, R., Johann, P. (1997), *Deduction Systems.* Springer.
  - Goubault-Larrecq, J. and Mackie, I. (1997), *Proof Theory and Automated Deduction*. Kluwer.
  - Leitsch, A. (1997), *The Resolution Calculus.* Springer.
- All available in the Resources Centre Library and the main library.

CS612: Automated Reasoning II

Lecture 1

Previously . . . (in Part I)

- Syntax and semantics of propositional logic and FOL
  - variables, constants, terms, quantifiers, . . .
- Syntax and semantics of clause logic
  - atoms, literals, clauses, Skolem terms
- Transformation of formulae to clausal form
- Resolution calculus
  - resolution rule
  - factoring rule
  - unification – for first-order clauses

Aims

- To discuss and study optimised transformations into clausal form
- To introduce well-founded orderings
- To study and show refutational completeness of resolution (ground case, lifting to f.o. case)
- To introduce ordered resolution with selection
- To discuss redundancy elimination and applications
- To discuss semantic tableau methods

Well-Founded Orderings

- To show the refutational completeness of resolution, we will make use of the concept of well-founded orderings.
- They will also be used when we discuss refinements of resolution.
- Reference:
  Baader, F. and Nipkow, T. (1998), *Term rewriting and all that.* Cambridge Univ. Press, Chapter 2.

Basic properties of relations

Let R be a binary relation over a set X ($R \subseteq X \times X$).

- R is transitive iff $\forall x, y, z \in X$, if $(x, y) \in R$ and $(y, z) \in R$ then $(x, z) \in R$.
- R is irreflexive iff $\forall x \in X$, $(x, x) \notin R$
- R is total, or linear, iff $\forall x, y \in X$ if $x \neq y$ then $R(x, y)$ or $R(y, x)$
- R^* denotes the reflexive-transitive closure of R. I.e.

$$R^* = \bigcup_{n=0}^{\infty} R^n = R^0 \cup R \cup R^2 \cup R^3 \cup \ldots$$

where $R^0 = \{(x, x) \mid x \in X\}$ and
$R^{n+1} = \{(x, y) \mid \exists z.\ (x, z) \in R \wedge (z, y) \in R^n\}$ for $n \geq 0$.

More Basic Definitions

- '$=$' is syntactic equality which means it is used to express that both sides name the same object, i.e. $=$ is taken to be the set $\{(x, x) \mid x \in X\}$ for some set X.
- '$\approx$' will be used for the logical equality symbol in FOL.

Orderings

- A (strict) ordering on a set X is a transitive and irreflexive binary relation on X, here denoted by $\succ$.
- The pair $(X, \succ)$ is then called a (strictly) ordered set.
- An element x of X is minimal wrt. $\succ$, if there is no y in X such that $x \succ y$.
- A minimal element x in X is called the smallest (or strictly minimal) element, if $y \succ x$, for all $y \in X$ different from x.
- Maximal and largest (or strictly maximal) elements are defined analogously.
- **Notation**: $\prec$ for the inverse relation $\succ^{-1}$
  $\succeq$ for the reflexive closure $(\succ \cup =)$ of $\succ$,
  i.e. $x \succeq y$ iff either $x \succ y$ or $x = y$

Well-Foundedness

- A (strict) ordering $\succ$ over X is called well-founded (or Noetherian or terminating), if there is no infinite decreasing chain $x_0 \succ x_1 \succ x_2 \succ \ldots$ of elements $x_i \in X$.

Lemma 1
$(X, \succ)$ is well-founded iff every non-empty subset Y of X has a minimal element.

Well-Founded Orderings: Examples

- **Natural numbers:** $(\mathbb{N}, >)$
- **Lexicographic orderings:** Let $(X_1, \succ_1)$, $(X_2, \succ_2)$ be well-founded orderings. Define their lexicographic combination

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

  as an ordering on $X_1 \times X_2$ such that

$$(x_1, x_2) \succ (y_1, y_2) \quad \text{iff} \quad \begin{array}{l} \text{(i) } x_1 \succ_1 y_1, \text{ or else} \\ \text{(ii) } x_1 = y_1 \text{ and } x_2 \succ_2 y_2 \end{array}$$

  (Analogously for more than two orderings.)

  This again yields a well-founded ordering (proof below).
- **Notation:** $\succ_{\text{lex}}$ for the lexicographic combination of $(X, \succ)$ twice (in general n times).

Well-Founded Orderings: Examples (cont'd)

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

$$w \succ w' \quad \text{iff} \quad \begin{array}{l} \text{(i) } |w| > |w'| \text{ or} \\ \text{(ii) } |w| = |w'| \text{ and } w >_{\Sigma,\text{lex}} w', \end{array}$$

  is a well-founded ordering on Σ^*.
- **Counterexamples:**
  - $(\mathbb{Z}, >)$
  - $(\mathbb{N}, <)$
  - the lexicographic ordering $>_{\Sigma,\text{lex}}$ on Σ^*

Lexicographic Combinations of Well-Founded Orderings

Lemma 2

$(X_i, \succ_i)$ is well-founded for $i \in \{1, 2\}$ iff $(X_1 \times X_2, \succ)$ with $\succ \; = (\succ_1, \succ_2)_{\text{lex}}$ is well-founded.

Proof:

(i) "$\Rightarrow$": Suppose $(X_1 \times X_2, \succ)$ is not well-founded. Then there is an infinite sequence $(a_0, b_0) \succ (a_1, b_1) \succ (a_2, b_2) \succ \ldots$.

Let $A = \{a_i \mid i \geq 0\} \subseteq X_1$. Since $(X_1, \succ_1)$ is well-founded, A has a minimal element a_n. But then $B = \{b_i \mid i \geq n\} \subseteq X_2$ cannot have a minimal element, contradicting the well-foundedness of $(X_2, \succ_2)$.

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

Noetherian Induction

Property 3 (Noetherian Induction)

Let $(X, \succ)$ be a well-founded ordering, let Q be a property of elements of X.

If for all $x \in X$ the implication

  if $Q(y)$ holds, for all $y \in X$ such that $x \succ y$, [a]

    then $Q(x)$ holds. [b]

is satisfied, then

  the property $Q(x)$ holds for all $x \in X$.

[a] induction hypothesis

[b] induction step

Noetherian Induction (cont'd)

Proof: By contradiction.
Thus, suppose for all $x \in X$ the implication above is satisfied, but $Q(x)$ does not hold for all $x \in X$.
Let $A = \{x \in X \mid Q(x) \text{ is false}\}$. Suppose $A \neq \emptyset$.
Since $(X, \succ)$ is well-founded, A has a minimal element x_1. Hence for all $y \in X$ with $x_1 \succ y$ the property $Q(y)$ holds.
On the other hand, the implication which is presupposed for this theorem holds in particular also for x_1, hence $Q(x_1)$ must be true so that x_1 cannot belong to A. *Contradiction*.

Multi-Sets

- Multi-sets are "sets which allow repetition". Formally:
- Let X be a set. A multi-set S over X is a mapping $S : X \longrightarrow \mathbb{N}$.
- Intuitively, $S(x)$ specifies the number of occurrences of the element x (of the base set X) within S.
- We say that x is an element of S, if $S(x) > 0$.
- We use set notation ($\in$, $\subset$, $\subseteq$, $\cup$, $\cap$, etc.) with analogous meaning also for multi-sets, e.g.,

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

Multi-Sets (cont'd)

- **Example:** $S = \{a, a, a, b, b\}$ is a multi-set over $\{a, b, c\}$, where $S(a) = 3$, $S(b) = 2$, $S(c) = 0$.
- A multi-set is called finite, if

$$|\{x \in X |\ S(x) > 0\}| < \infty,$$

  for each x in X.
- **From now on we consider finite multi-sets only.**

Exercise

Suppose $S_1 = \{c, a, b\}$ and $S_2 = \{a, b, b, a\}$ are multi-sets over $\{a, b, c, d\}$.
Determine $S_1 \cup S_2$ and $S_1 \cap S_2$.

Exercise

Suppose $S_1 = \{c, a, b\}$ and $S_2 = \{a, b, b, a\}$ are multi-sets over $\{a, b, c, d\}$.
Determine $S_1 \cup S_2$ and $S_1 \cap S_2$.

Answer:

$$S_1 \cup S_2 = \{a, a, a, b, b, b, c\}$$
$$S_1 \cap S_2 = \{a, b\}$$

Multi-Set Orderings

- Let $(X, \succ)$ be an ordering. The multi-set extension $\succ_{\mathrm{mul}}$ of $\succ$ to (finite) multi-sets over X is defined by

$$\begin{aligned} S_1 \succ_{\mathrm{mul}} S_2 \quad \text{iff} \quad & S_1 \neq S_2 \text{ and} \\ & \forall x \in X, \text{ if } S_2(x) > S_1(x) \text{ then} \\ & \quad \exists y \in X : S_1(y) > S_2(y) \text{ and } y \succ x \end{aligned}$$

- **Example:** Over $(\mathbb{N}, >)$:

$$\{5, 5, 4, 3, 2\} >_{\mathrm{mul}} \{5, 4, 4, 3, 3, 2\} >_{\mathrm{mul}} \{5, 4, 3\}$$

  Exercise: How is the empty set related to each of these?

Multi-Set Orderings (cont'd)

Property 4

1. $\succ_{\mathrm{mul}}$ is an ordering.
2. if $\succ$ well-founded then $\succ_{\mathrm{mul}}$ well-founded.
3. if $\succ$ total then $\succ_{\mathrm{mul}}$ total

Proof: see Baader and Nipkow, p. 22–24.

Summary

- (strict) orderings
- well-founded orderings
- lexicographic ordering $\succ_{\mathrm{lex}}$, $(\succ_1, \succ_2)_{\mathrm{lex}}$

  = lexicographic combination of one or more orderings
- Noetherian (well-founded) induction
- multi-sets
- multi-set ordering $\succ_{\mathrm{mul}}$

  = multi-set extension of ~~another~~ ordering $\succ$ on the elements

CS612: Automated Reasoning II

Lecture 2

Previously ...

- (strict) ordering
- well-founded orderings
- lexicographic orderings
- multi-set orderings
- multi-sets

Recall from Part I: Literals, Clauses

- Literals

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

- Clauses

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

- **Note:**
  - We assume $\vee$ is associative and commutative, repetitions matter.
  - I.e. from now on we regard clauses as multi-sets of literals and interpret, and denote, them as disjunctions.
  - Thus, $C = P \vee P \vee \neg Q$ is identical to $C' = P \vee \neg Q \vee P$. But neither C nor C' are the same as $D = P \vee \neg Q$.

Recap: Resolution Calculus

- **Propositional/ground resolution calculus** *Res*

$$\frac{C \vee A \qquad \neg A \vee D}{C \vee D} \qquad \text{(resolution)}$$

$$\frac{C \vee A \vee A}{C \vee A} \qquad \text{((positive !) factoring)}$$

- Terminology: resolvent for $C \vee D$; (positive) factor for $C \vee A$
  resolved atom and factored atom, resp., for 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.
- Since we assume $\vee$ is associative and commutative, note that A and $\neg A$ can occur anywhere in their respective clauses.

Notation in Part II

- Notation:

A, B	atoms
L	literals
C, D	clauses
$\perp$	the empty clause / falsum
N	sets of clauses
F, G	formulae

- $N \models C$ means 'any model which satisfies N also satisfies C'
  What does $N \models \perp$ mean?
- $N \vdash_{Res} C$ means 'C is derivable from N using the rules of *Res*'
  i.e. there is a proof of C from N in the calculus *Res*

Recap: Conversion into Conjunctive Normal Form

- The conjunctive normal form CNF(F) of a formula F can be computed by applying these rewrite rules:

$$
\begin{array}{rcl}
(F \leftrightarrow G) & \Rightarrow_{\mathrm{CNF}} & (F \rightarrow G) \wedge (G \rightarrow F) \\
(F \rightarrow G) & \Rightarrow_{\mathrm{CNF}} & (\neg F \vee G) \\
\neg(F \vee G) & \Rightarrow_{\mathrm{CNF}} & (\neg F \wedge \neg G) \\
\neg(F \wedge G) & \Rightarrow_{\mathrm{CNF}} & (\neg F \vee \neg G) \\
\neg\neg F & \Rightarrow_{\mathrm{CNF}} & F \\
(F \wedge G) \vee H & \Rightarrow_{\mathrm{CNF}} & (F \vee H) \wedge (G \vee H)
\end{array}
$$

$$
\begin{array}{rclcrcl}
(F \wedge \top) & \Rightarrow_{\mathrm{CNF}} & F & \quad & (F \wedge \perp) & \Rightarrow_{\mathrm{CNF}} & \perp \\
(F \vee \top) & \Rightarrow_{\mathrm{CNF}} & \top & & (F \vee \perp) & \Rightarrow_{\mathrm{CNF}} & F \\
\neg\top & \Rightarrow_{\mathrm{CNF}} & \perp & & \neg\perp & \Rightarrow_{\mathrm{CNF}} & \top
\end{array}
$$

- These rules are to be applied modulo associativity and commutativity of $\wedge$ and $\vee$.

Notation in Part II

- Notation:

A, B	atoms
L	literals
C, D	clauses
$\perp$	the empty clause / falsum
N	sets of clauses
F, G	formulae

- $N \models C$ means 'any model which satisfies N also satisfies C'
  What does $N \models \perp$ mean? Answer: N is unsatisfiable
- $N \vdash_{Res} C$ means 'C is derivable from N using the rules of *Res*'
  i.e. there is a proof of C from N in the calculus *Res*

Conversion into Conjunctive Normal Form (cont'd)

- The first five rules~~, plus the rule $(\neg Q)$,~~ compute the negation normal form (NNF) of a formula.
- For every formula F:
  - $\models F \leftrightarrow \mathrm{CNF}(F)$
  - $\models F \leftrightarrow \mathrm{NNF}(F)$
- Conversion to CNF (and therefore clause form) may produce a formula whose size is exponential in the size of the original formula.

Relaxing the Requirements

- The goal
  "find a formula G in CNF such that: $\models F \leftrightarrow G$"
  is impractical.
- Recall, for every closed f.o. formula F:
  - $\models \mathrm{Cls}(F) \longrightarrow F$, but not conversely.
  - F is (un)satisfiable iff $\mathrm{Cls}(F)$ is (un)satisfiable.
- Since we can only preserve satisfiability-equivalence anyway, there is lots of room for optimisation.
- We therefore relax the requirement to
  "find a formula G in CNF such that:
  F is satisfiable iff G is satisfiable"
  and can get efficient transformations.

Structural transformation

- Assume the context is propositional logic.
- Idea of structural transformation (or renaming): Exploit (from left to right) the property that for any formula F,

  $$F[G] \text{ is satisfiable iff } F[Q] \wedge (Q \leftrightarrow G) \text{ is satisfiable,}$$

  where Q is a new propositional variable.
  View Q as an abbreviation for G.

- We can use this rule recursively for all subformulas in the original formula (this introduces a linear number of new propositional variables).
- Conversion of the resulting formula to CNF increases the size only by an additional constant factor (each formula $Q \leftrightarrow G$ gives rise to at most one application of the distributivity law).

Example

- $(P \vee R) \longrightarrow P$ is satsifiable iff

  $$\begin{aligned} & (Q_0 \longrightarrow P) \\ \wedge\ & (Q_0 \leftrightarrow (P \vee R)) \end{aligned}$$

  is satisfiable iff

  $$\begin{aligned} & Q_1 \\ \wedge\ & (Q_1 \leftrightarrow (Q_0 \longrightarrow P)) \\ \wedge\ & (Q_0 \leftrightarrow (P \vee R)) \end{aligned}$$

  is satisfiable

Exercise

- Compute the structural transformation of $\neg P \vee (R \wedge P)$ in which a new symbol is introduced for every non-literal subformula.

Optimising Structural Transformation

- A further improvement is possible by taking the polarity of the subformula F into account.
- Assume that F contains neither $\longrightarrow$ nor $\leftrightarrow$.
- A subformula G of F has positive polarity in F, if it occurs below an even number of negation symbols.
  It has negative polarity in F, if it occurs below an odd number of negation symbols.

Optimising Structural Transformation (cont'd)

Property 5

Let $F[G]$ be a formula containing neither $\longrightarrow$ nor $\leftrightarrow$;
let Q be a propositional variable not occurring in $F[G]$.

1. If G has positive polarity in F, then
   $F[G]$ is satisfiable iff $F[Q] \wedge (Q \longrightarrow G)$ is satisfiable.
2. If G has negative polarity in F, then
   $F[G]$ is satisfiable iff $F[Q] \wedge (G \longrightarrow Q)$ is satisfiable.

Structural transformation has many uses and can be generalised to first-order logic.

Exercise

- What is the optimised structural transformation of $\neg P \vee (R \wedge P)$ in which a new symbol is introduced for every non-literal subformula?

Ground Expressions, Ground Instances

- Ground terms are terms with no occurrences of variables.
- Ground atoms are atoms with no occurrences of variables.
- Ground literals, ground clauses, ground formulae are defined similarly.
- A ground instance of an expression (atom, literal, clause, formula) is obtained by uniformly instantiating the variables in it with ground terms.

Herbrand Universe

- We assume there is at least one constant in our language Σ.
- The Herbrand universe (over Σ), denoted T_Σ, is the set of ground terms over Σ.
- **Example:** Suppose the language has one binary function symbol f and two constants a and b. The following are in T_Σ:

$$a,\ b,\ f(a,a),\ f(a,b),\ f(b,a),\ f(b,b),\ f(a,f(a,a)),\ \ldots$$

- If Σ contains non-constant function symbols then T_Σ is infinite.

Exercise

- Suppose Σ is the language with one unary function symbol f and one constant a.
  Write down the elements of the Herbrand universe T_Σ.

Herbrand Interpretations

- A Herbrand interpretation (over Σ), denoted I, is a set of ground atoms over Σ.
- Truth in I of ground formulae is defined inductively by:

$$I \models \top \qquad\qquad I \not\models \bot$$

$$\begin{array}{rcl} I \models A & \text{iff} & A \in I, \text{ for any ground atom } A \\ I \models \neg F & \text{iff} & I \not\models F \\ I \models F \wedge G & \text{iff} & I \models F \text{ and } I \models G \\ I \models F \vee G & \text{iff} & I \models F \text{ or } I \models G \end{array}$$

- Truth in I of any quantifier-free formula F with free variables $x_1, \ldots, x_n$ is defined by:

$$I \models F(x_1, \ldots, x_n) \ \text{ iff } \ I \models F(t_1, \ldots, t_n), \text{ for every } t_i \in \mathrm{T}_\Sigma$$

Herbrand Interpretations (cont'd)

- In a Herbrand interpretation values are fixed to be ground terms and functions are fixed to be the (Skolem) functions in Σ.
- Only predicate symbols p may be freely interpreted as relations $p^{\mathcal{M}} \subseteq \mathrm{T}_\Sigma^n$.
- I is an interpretation $\mathcal{M}$ where the values are given by:
  constant a: $\quad a^{\mathcal{M},s} = a$
  function f: $\quad (f(t_1, \ldots, t_n))^{\mathcal{M},s} = f(t_1^{\mathcal{M},s}, \ldots, t_n^{\mathcal{M},s})$

Property 6
Every Herbrand interpretation (set of ground atoms) I uniquely determines an interpretation $\mathcal{M}$ via

$$(s_1, \ldots, s_n) \in p^{\mathcal{M}} \ \text{ iff } \ p(s_1, \ldots, s_n) \in I$$

Exercise

- Suppose Σ is the language with one unary function symbol f, one constant a and one predicate symbol p.
  Which of the following are Herbrand interpretations over Σ?
  1. $I_1 = \{p(a)\}$
  2. $I_2 = \{p(a),\ p(f(a))\}$
  3. $I_3 = \{p(a),\ q(f(a))\}$
  4. $I_4 = \{p(a),\ \neg p(f(a))\}$
- For I_2 determine whether the following is true?
  1. $I_2 \models p(a)$
  2. $I_2 \models \neg p(a)$
  3. $I_2 \models \neg p(f(a))$
  4. $I_2 \models p(a) \wedge p(f(a))$
  5. $I_2 \models p(x)$

Exercise

- Suppose Σ is the language with one unary function symbol f, one constant a and one predicate symbol p.
  Which of the following are Herbrand interpretations over Σ?
  1. $I_1 = \{p(a)\}$ yes
  2. $I_2 = \{p(a),\ p(f(a))\}$ yes
  3. $I_3 = \{p(a),\ q(f(a))\}$ no; q not a symbol in Σ
  4. $I_4 = \{p(a),\ \neg p(f(a))\}$ no; only pos. atoms can belong to I
- For I_2 determine whether the following is true?
  1. $I_2 \models p(a)$ yes
  2. $I_2 \models \neg p(a)$ no
  3. $I_2 \models \neg p(f(a))$ no
  4. $I_2 \models p(a) \wedge p(f(a))$ yes
  5. $I_2 \models p(x)$ no

Exercise

- Suppose Σ is the language with one unary function symbol f, one constant a and one predicate symbol p.
  Which of the following are Herbrand interpretations over Σ?
  1. $I_1 = \{p(a)\}$ yes
  2. $I_2 = \{p(a),\ p(f(a))\}$ yes
  3. $I_3 = \{p(a),\ q(f(a))\}$ no; q not a symbol in Σ
  4. $I_4 = \{p(a),\ \neg p(f(a))\}$ no; only pos. atoms can belong to I
- For I_2 determine whether the following is true?
  1. $I_2 \models p(a)$
  2. $I_2 \models \neg p(a)$
  3. $I_2 \models \neg p(f(a))$
  4. $I_2 \models p(a) \wedge p(f(a))$
  5. $I_2 \models p(x)$

Examples of Herbrand Interpretations

- Suppose Σ_{Nat} has constant 1, binary function $+$, unary predicate symbol p.
- Herbrand interpretation over Σ_{Nat}:

$$I = \{\ p(1),\ p(1+1),\ p(1+1+1),\ p(1+1+1+1),\ \ldots\}$$

- Notation: $n + m$ instead of $+(n, m)$

Examples of Herbrand Interpretations (cont'd)

- Suppose Σ has a single predicate symbol r. Let I be a Herbrand interpretation over Σ.
  - $I \models r(x,x)$ iff $r(t,t) \in I$ for every ground term t in T_Σ.
  - $I \models \neg r(x,x)$ iff $r(t,t) \notin I$ for every ground term t in T_Σ.
  - $I \models \neg r(x,y) \vee r(y,x)$ iff when $r(s,t) \in I$ then $r(t,s) \in I$, for any ground terms $s,t \in \mathrm{T}_\Sigma$.
  - $I \models \neg r(x,y) \vee \neg r(y,z) \vee r(x,z)$ iff when $r(s,t) \in I$ and $r(t,u) \in I$ then $r(s,u) \in I$, for any ground terms $s,t,u \in \mathrm{T}_\Sigma$.

Existence of Herbrand Models

- A Herbrand interpretation I is called a Herbrand model of F, if $I \models F$.
- Let $G_\Sigma(N)$ denote the set of ground instances of N over the language Σ. I.e. $G_\Sigma(N) = \{C\boldsymbol{\sigma} \mid C \in N, \boldsymbol{\sigma} : X \longrightarrow \mathrm{T}_\Sigma \text{ a ground substitution}\}$.

Property 7 (Herbrand)
Let N be a set of Σ-clauses.

N satisfiable	iff	N has a Herbrand model (over Σ)
	iff	$G_\Sigma(N)$ has a Herbrand model (over Σ)

(The proof will be given below in the context of the completeness proof for resolution.)

Examples

- Consider

$$\begin{aligned} N &= \{p(x),\ q(f(y)) \vee r(y)\} \\ T_\Sigma &= \{a, f(a), f(f(a)), \ldots\} \end{aligned}$$

- $p(a)$ and $p(f(a))$ are both ground instances of the clause $p(x)$.
- **Exercise**: Give examples of ground instances of N.

Example of a set of ground instances G_Σ

- For $\Sigma_\mathbb{N}$ one obtains for

$$N = \{p(1),\ \neg p(x) \vee p(x+1)\}$$

the following ground instances:

$$\begin{aligned} & p(1) \\ & \neg p(1) \vee p(1+1) \\ & \neg p(1+1) \vee p(1+1+1) \\ & \neg p(1+1+1) \vee p(1+1+1+1) \\ & \ldots \end{aligned}$$

Exercise

- For N on the previous slide write down a Herbrand model of $G_\Sigma(N)$

- Is N satisfiable?

CS612: Automated Reasoning II

Lecture 3

Summary

- atoms, literals (positive & negative), clauses (= multi-sets)
- Res: resolution & positive factoring
- optimised conversion to clause form: structural transformation
- ground expressions, ground instances
- Herbrand universe
- Herbrand interpretation, Herbrand model
- satisfiability in I: $\models$
- Herbrand's Theorem

Previously ...

- orderings
- multi-set orderings
- ground terms, ground atoms, ground literals, ground clauses
- Herbrand universe = set of ground terms
- Herbrand interpretation I = set of ground atoms
  - For atoms: $A \in I$ iff $I \models A$
    $A \notin I$ iff $I \not\models A$

Soundness of Resolution

Property 8

Propositional resolution (resolution on ground clauses) is sound.

Proof: We have to show: $N \vdash_{Res} C \ \Rightarrow\ N \models C$,
It suffices to show that
(*) for every rule $\frac{C_1 \ \dots\ C_n}{D}$ we have $C_1, \dots, C_n \models D$.
For resolution, assume $I \models C \lor A$, $I \models \neg A \lor D$ and show $I \models C \lor D$.
(a) Case $I \models A$: Then $I \models D$ for else $I \not\models \neg A \lor D$. Hence $I \models C \lor D$. (b) Case $I \not\models A$: Then $I \models \neg A$. Since $I \models C \lor A$, $I \models C$ and consequently $I \models C \lor D$.
For factoring, assume $I \models C \lor A \lor A$ and show $I \models C \lor A$.
Exercise.

- Any rule which satisfies (*) is said to be sound.

Refutational Completeness of Resolution

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

Defining Clause Orderings

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

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

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

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

Example

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

$$\neg A_5 \succ A_5 \succ \neg A_4 \succ A_4 \succ \dots \succ \neg A_0 \succ A_0$$

- And:

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

Exercise

- Suppose $A_4 \succ A_3 \succ A_2 \succ A_1$
- How are these clauses ordered by $\succ_C$?

1. $\neg A_3 \vee A_4$
2. $A_3 \vee A_1 \vee A_1$
3. $\neg A_4 \vee A_2$
4. $A_3 \vee A_1$

Exercise

- Suppose $A_4 \succ A_3 \succ A_2 \succ A_1$
- How are these clauses ordered by $\succ_C$?

1. $\neg A_3 \vee A_4$
2. $A_3 \vee A_1 \vee A_1$
3. $\neg A_4 \vee A_2$
4. $A_3 \vee A_1$

- Ordering of literals:

$$\neg A_4 \succ_L A_4 \succ_L \neg A_3 \succ_L A_3 \succ_L \neg A_2 \succ_L A_2 \succ_L \neg A_1 \succ_L A_1$$

- Ordering of clauses: $3 \succ_C 1 \succ_C 2 \succ_C 4$
  $(4 \prec_C 2 \prec_C 1 \prec_C 3)$

Exercise

- Suppose $A_4 \succ A_3 \succ A_2 \succ A_1$
- How are these clauses ordered by $\succ_C$?

1. $\neg A_3 \vee A_4$
2. $A_3 \vee A_1 \vee A_1$
3. $\neg A_4 \vee A_2$
4. $A_3 \vee A_1$

- Ordering of literals:

$$\neg A_4 \succ_L A_4 \succ_L \neg A_3 \succ_L A_3 \succ_L \neg A_2 \succ_L A_2 \succ_L \neg A_1 \succ_L A_1$$

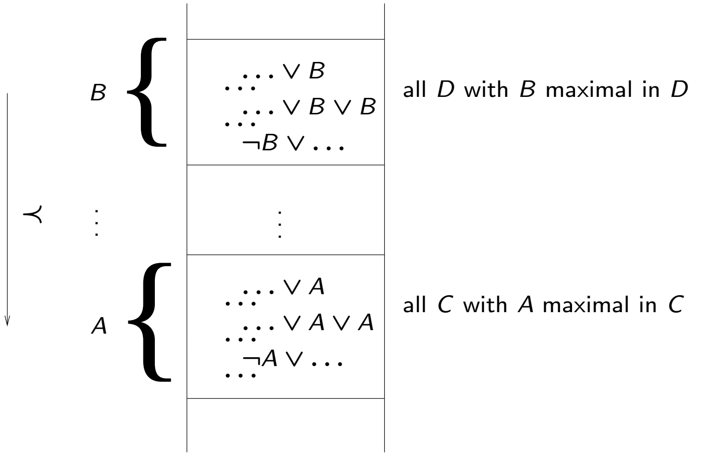
Properties of Clause Orderings

Property 9

1. The orderings ($\succ_L$ and $\succ_C$) on (ground) literals and clauses are total and well-founded.
2. Let C and D be clauses with A an occurrence of a maximal atom in C and B an occurrence of a maximal atom in D.
   (i) If $A \succ B$ then $C \succ D$.
   (ii) If $A = B$ and 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:



Saturation of Clause Sets under *Res*

- N is called saturated (wrt. resolution), if $Res(N) \subseteq N$, where
  $Res(N) = \{C \mid C$ is a conclusion of applying a rule in Res to premises in $N\}$
- Method of level saturation computes the saturation of a set N as the closure of N:
  $Res^*(N) = \bigcup_{n \geq 0} Res^n(N)$, where
  $Res^0(N) = N$
  $Res^{n+1}(N) = Res(Res^n(N)) \cup Res^n(N)$, for $n \geq 0$

Property 10

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

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

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

Construction of Herbrand Interpretations

- We now show the "$\Rightarrow$" direction.
- **Given**: set N of ground clauses, atom ordering $\succ$.
- **Wanted**: Herbrand interpretation I such that
  - "many" clauses from N are true in I, and
  - $I \models N$, if N is saturated and $\bot \notin N$.

Example

Let $A_5 \succ A_4 \succ A_3 \succ A_2 \succ A_1 \succ A_0$ (strictly maximal literals in red)

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

$I = \{A_1, A_2, A_4, A_5\}$ is not a model of the clause set because there exists a counter-example (or failure clause) (clause 6).

Main Ideas of the Construction

- Approximate (!) description: Define I inductively by:
  - Starting with a minimal clause C in N.
    (Since in the ground case the ordering is total, there is a smallest clause and we start in fact with this clause.)
  - Consider the largest atom in C and attempt to define (in a certain way) $I_C \cup \Delta_C$ (!) as the minimal extension of the partial interpretation constructed so far (I_C) so that C becomes true.
  - Iterate for $N \setminus \{C\}$, and so forth.
- I.e. clauses are considered in the order given by $\prec$.
- When considering C, one already has a partial interpretation I_C available (initially $I_C = \emptyset$).

Main Ideas of the Construction (cont'd)

- If C is true in the partial interpretation I_C, nothing is done ($\Delta_C = \emptyset$).
- If C is false, change I_C such that C becomes true.
- Changes should, however, be monotone. One never deletes anything from I_C and the truth value of any clause smaller than C should be maintained the way it was in I_C.
- Hence, one chooses $\Delta_C = \{A\}$ iff C is false in I_C, when both
  (i) A occurs positively in C, and
  (ii) this occurrence in C is strictly maximal (i.e. largest) in the ordering on literals.

Note: (i) $\Rightarrow$ adding A will make C become true.
(ii) $\Rightarrow$ changing the truth value of A has no effect on smaller clauses.

Resolution Reduces Counterexamples

$$\frac{\neg A_1 \lor A_4 \lor A_3 \lor A_0 \qquad \neg A_1 \lor \neg A_4 \lor A_3}{\neg A_1 \lor \neg A_1 \lor A_3 \lor A_3 \lor A_0}$$

Construction of I for the extended clause set:

clauses C	I_C	Δ_C	Remarks
$\neg A_0$	$\emptyset$	$\emptyset$	
$A_0 \lor A_1$	$\emptyset$	$\{A_1\}$	
$A_1 \lor A_2$	$\{A_1\}$	$\emptyset$	
$\neg A_1 \lor A_2$	$\{A_1\}$	$\{A_2\}$	
$\neg A_1 \lor \neg A_1 \lor A_3 \lor A_3 \lor A_0$	$\{A_1, A_2\}$	$\emptyset$	A_3 occurs twice
			min. counter-ex.
$\neg A_1 \lor A_4 \lor A_3 \lor A_0$	$\{A_1, A_2\}$	$\{A_4\}$	
$\neg A_1 \lor \neg A_4 \lor A_3$	$\{A_1, A_2, A_4\}$	$\emptyset$	counter-example
$\neg A_1 \lor A_5$	$\{A_1, A_2, A_4\}$	$\{A_5\}$	

The same I, but smaller counter-example, hence some progress was made.

Factoring Reduces Counterexamples

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

Construction of I for the extended clause set:

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

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

Construction of Candidate 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 guided by $\succ$:

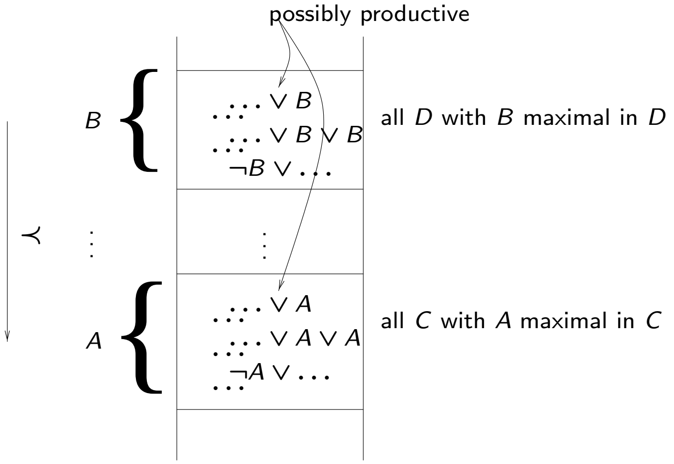
$$I_C \quad := \quad \bigcup_{C \succ D} \Delta_D$$

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

- 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 $(N, \succ)$

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



Some Properties of the Construction

Property 11

(i) $C = \neg A \vee C' \;\Rightarrow\;$ no D s.t. $D \succeq C$ produces A.

(ii) C productive $\;\Rightarrow\; I_C \cup \Delta_C \models C$ and $I_N \models C$.

(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 $B \succ A$, where B and A are maximal atoms in D and C, respectively, then

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

Some Properties of the Construction (cont'd)

(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 $B \succ A$, where B and A are maximal atoms in D and C, respectively, then

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

(v) $C = C' \vee A$ produces $A \;\Rightarrow\; I_N \not\models C'$.

Model Existence Theorem

Property 12 (Bachmair, Ganzinger 1990)

Let $\succ$ be a clause ordering, let N be saturated wrt. *Res*, and suppose that $\bot \notin N$. Then

$$I_N^{\succ} \models N.$$

Corollary 13

Let N be saturated wrt. *Res*. Then

$$N \models \bot \quad \text{iff} \quad \bot \in N.$$

Corollary 14

Res is refutationally complete.

Model Existence Theorem (cont'd)

Proof of Property 12:

Suppose $\bot \notin N$, but $I_N^{\succ} \not\models N$.

Let $C \in N$ be minimal (wrt. $\succ$) such that $I_N^{\succ} \not\models C$.

Since C is false in I_N, C is not productive. (NB: $I_N = I_N^{\succ}$)

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)

$\Rightarrow$ $I_N \not\models \neg A$ and $I_N \not\models C'$ $\Rightarrow$ $I_N \models A$ 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 $I_N \not\models A$ and $I_N \not\models C'$. Then $\frac{C' \vee A \vee A}{C' \vee A}$ yields a smaller counter-example $C' \vee A \in N$.

$\Rightarrow$ contradicts minimality of C.

Compactness of Propositional Logic

Property 15 (Compactness)

Let N be a set of propositional clauses. Then:

N is unsatisfiable iff there exists a finite subset $M \subseteq N$ which is unsatisfiable.

Proof:

"$\Leftarrow$": trivial (why?).

"$\Rightarrow$": Let N be unsatisfiable, i.e. $N \models \bot$

$\Rightarrow$ $Res^*(N)$ unsatisfiable, since $N \subseteq Res^*(N)$

$\Rightarrow$ $\bot \in Res^*(N)$ by refutational completeness of resolution

$\Rightarrow$ $\exists n \geq 0$ s.t. $\bot \in Res^n(N)$

$\Rightarrow$ $\bot$ has a finite resolution proof Π;

let M be the set of assumptions in Π.

Summary

- Soundness of *Res*
- Model construction
  - Given: Set N of ground clauses; atom ordering $\succ$
  - Output: Candidate model $I_N^{\succ}$
- Model Existence Theorem
- Refutational completeness of *Res*
- Compactness of propositional logic

- sound rule, saturated set, productive clause

CS612: Automated Reasoning II

Lecture 4

Previously . . .

- Soundness and refutational completeness of *Res* for propositional/ground clauses
- Model construction
  - guided by an ordering $\succ$
- Model Existence Theorem
- Compactness of propositional logic

Generalising Resolution to Non-Ground Clauses

- Propositional/ground resolution:
  - refutationally complete,
  - in its most naive version:
    not guaranteed to terminate for satisfiable sets of clauses,
    (improved versions do terminate, however)
  - clearly inferior to the DPLL procedure
    (even with various improvements).

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

General Resolution through Instantiation

Idea: instantiate clauses appropriately:

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

General Resolution through Instantiation: Problems

- Problems:
  - More than one instance of a clause can participate in a proof.
  - Even worse: There are infinitely many possible instances.
- Observation:
  - Instantiation must produce complementary literals (so that inferences become possible).
- Idea:
  - Do not instantiate more than necessary to get complementary literals.

General Resolution through Lazy Instantiation

Idea: do not instantiate more than necessary:

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

Lifting Principle

- **Problem:**
  Make saturation of infinite sets of clauses as they arise from taking the (ground) instances of finitely many general clauses (with variables) effective and efficient.
- **Idea (Robinson 1965):**
  - Resolution for general clauses:
  - Equality of ground atoms (matching) is generalised to unifiability of general atoms;
  - Only compute most general (minimal) unifiers.

Lifting Principle (cont'd)

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

Resolution for General Clauses

- **General binary resolution calculus *Res*:**

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

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \quad \text{if } \sigma = \mathrm{mgu}(A, B) \quad \text{(positive factoring)}$$

- **General resolution calculus *RIF* with implicit factoring:**

$$\frac{C \vee A_1 \vee \ldots \vee A_n \qquad D \vee \neg B}{(C \vee D)\sigma} \qquad \text{(RIF)}$$

$$\text{if } \sigma = \mathrm{mgu}(A_1, \ldots, A_n, B)$$

Resolution for General Clauses (cont'd)

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

Lifting Lemma

Lemma 16

Let C and D be variable-disjoint clauses. If

$$\begin{array}{cc} C & D \\ \downarrow \sigma & \downarrow \rho \\ C\sigma & D\rho \\ \hline \multicolumn{2}{c}{C'} \end{array} \qquad \text{(propositional resolution)}$$

then there exist C'' and a substitution τ such that

$$\begin{array}{c} \dfrac{C \qquad D}{C''} \\ \downarrow \tau \\ C' = C''\tau \end{array} \qquad \text{(general resolution)}$$

Lifting Lemma (cont'd)

- An analogous lifting lemma holds for factoring.

Saturation of Sets of General Clauses

Recall that $G_\Sigma(N)$ denotes the set of ground instances of N over the signature Σ.

Corollary 17

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

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

Saturation of Sets of General Clauses (cont'd)

Proof:

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

Let $C' \in Res(G_\Sigma(N))$, meaning

(i) there exist resolvable ground instances $C\sigma$ and $D\rho$ of C and D belonging to N and C' is their resolvent, or else

(ii) C' is a factor of a ground instance $C\sigma$ of $C \in N$.

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

Case (ii): Similar (exercise).

Herbrand's Theorem

Lemma 18

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

Lemma 19

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

Herbrand's Theorem (cont'd)

Property 20 (Herbrand Theorem)

A set N of Σ-clauses is satisfiable iff it has a Herbrand model over Σ.

Proof:

The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part let $N \not\models \bot$.

$$
\begin{array}{llll}
N \not\models \bot & \Rightarrow & \bot \notin Res^*(N) & \text{(resolution is sound)} \\
& \Rightarrow & \bot \notin G_\Sigma(Res^*(N)) & \\
& \Rightarrow & I_{G_\Sigma(Res^*(N))} \models G_\Sigma(Res^*(N)) & \text{(Prt. 12; Cor. 17)} \\
& \Rightarrow & I_{G_\Sigma(Res^*(N))} \models Res^*(N) & \text{(Lemma 19)} \\
& \Rightarrow & I_{G_\Sigma(Res^*(N))} \models N & (N \subseteq Res^*(N))
\end{array}
$$

The Theorem of Löwenheim-Skolem

Property 21 (Löwenheim-Skolem Theorem)
Let Σ be a countable signature and let S be a set of closed Σ-formulae. Then S is satisfiable iff S has a model over a countable universe.

Proof:
If both X and Σ are countable, then S can be at most countably infinite. Now generate, maintaining satisfiability, a set N of clauses from S. This extends Σ by at most countably many new Skolem functions to Σ'. As Σ' is countable, so is $T_{\Sigma'}$, the universe of Herbrand-interpretations over Σ'. Now apply Herbrand's Theorem (Property 20).

Refutational Completeness of General Resolution

Property 22
Let N be a set of general clauses where $Res(N) \subseteq N$. Then

$$N \models \bot \text{ iff } \bot \in N.$$

Proof:
Let $Res(N) \subseteq N$.
By Corollary 17: $Res(G_\Sigma(N)) \subseteq G_\Sigma(N)$

$$\begin{aligned} N \models \bot \;&\Leftrightarrow\; G_\Sigma(N) \models \bot && \text{(Lemmas 18 \& 19; Property 20)} \\ &\Leftrightarrow\; \bot \in G_\Sigma(N) && \text{(prop. resol. is sound and complete)} \\ &\Leftrightarrow\; \bot \in N \end{aligned}$$

Compactness of First-Order Logic

Property 23 (Compactness Theorem for First-Order Logic)
Let Φ be a set of first-order formulae.
Φ is unsatisfiable iff some finite subset $\Psi \subseteq \Phi$ is unsatisfiable.

Proof:
The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part let Φ be unsatisfiable and let N be the set of clauses obtained by Skolemisation and CNF transformation of the formulae in Φ. Clearly $Res^*(N)$ is unsatisfiable. By Property 22, $\bot \in Res^*(N)$, and therefore $\bot \in Res^n(N)$ for some $n \in \mathbb{N}$. Consequently, $\bot$ has a finite resolution proof Π of depth $\leq n$. Choose Ψ as the subset of formulae in Φ such that the corresponding clauses contain the assumptions (leaves) of Π.

Summary

- Resolution for general, first-order clauses
- Refutational completeness, consequence of:
  1. refutational completeness of ground case
  2. lifting lemmas
  3. Herbrand's Theorem
- Löwenheim–Skolem Theorem
- Compactness of FOL

CS612: Automated Reasoning II

Lecture 5

Previously ...

- Basic resolution calculus for ground clauses & non-ground clauses
- Refutational completeness
- Model existence theorem
- Model construction, guided by $\succ$

Ordered Resolution with Selection

- Motivation: Search space for *Res* is very large.
- Ideas for improvement:
  - In the completeness proof (Model Existence Theorem, Pty 12) one only needs to resolve upon and factor maximal atoms

    $\Rightarrow$ if the calculus is restricted to inferences involving maximal atoms, the proof remains correct

    $\Rightarrow$ ordering restrictions
  - In the proof, it does not really matter with which negative literal an inference is performed

    $\Rightarrow$ choose a negative literal don't-care-nondeterministically

    $\Rightarrow$ selection

Selection Functions

- A selection function is a mapping

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

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

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

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

Maximality wrt non-ground clauses

- In the completeness proof, we talk about (strictly) maximal literals of ground clauses.
- In the non-ground calculus, we have to consider those literals that correspond to (strictly) maximal literals of ground instances.
- Thus, let $\succ$ be a total and well-founded ordering on ground atoms.
- A literal L is called [strictly] maximal wrt. a clause C iff there exists a ground substitution σ such that for all L' in C: $L\sigma \succeq L'\sigma$ [$L\sigma \succ L'\sigma$].

Resolution Calculus $Res_S^{\succ}$

- Let $\succ$ be an atom ordering and S a selection function.
- **General ordered resolution calculus with selection $Res_S^{\succ}$:**

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

provided $\sigma = \mathrm{mgu}(A, B)$ 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. $D\sigma$.

Resolution Calculus $Res_S^{\succ}$ (cont'd)

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

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

(i) $A\sigma$ is maximal wrt. $C\sigma$ and

(ii) nothing is selected in C.

Special Instance: $Res_S^{\succ}$ for Propositional Logic

- For propositional and ground clauses the resolution inference simplifies to

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

provided

(i) A is strictly maximal wrt. C, i.e. $A \succ C$;

(ii) nothing is selected in C by S;

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

- Note:
  - $A \succ C$ is the same as $A \succ B$, where B is any maximal atom in C.
  - $\neg A$ is maximal wrt. D means $\neg A \succeq L$, for every literal L in D, which means $A \succeq B$ for every maximal atom B in D.

Special Instance: $Res_S^{\succ}$ for Propositional Logic (cont'd)

- Ordered factoring:

$$\frac{C \vee A \vee A}{\cancel{(}C \vee A\cancel{)\sigma}}$$

provided

(i) A is maximal wrt. C and

(ii) nothing is selected in C.

Search Spaces Become Smaller

- Example:

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

we assume $A \succ B$ and S as indicated by $\boxed{L}$. The maximal literal in a clause is depicted in red.

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

Exercise

Consider the following set N of clauses.

1. $\neg P(x) \vee P(f(x))$
2. $P(a)$

(i) Give a derivation for it under unrestricted resolution.

(ii) Define an ordering or selection function, or both, so that no inference is performed on N.

Avoiding Rotation Redundancy

- From

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

we can obtain by rotation

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

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 ordering restrictions.

Avoiding Rotation Redundancy (cont'd)

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

Lifting Lemma for $Res_S^{\succ}$

Lemma 24

Let C and D be variable-disjoint clauses. If

$$\begin{array}{ccc} C & & D \\ \downarrow \sigma & & \downarrow \rho \\ C\sigma & & D\rho \\ \hline & C' & \end{array} \qquad \text{(propositional inference in } Res_S^{\succ}\text{)}$$

and if $S(C\sigma) \simeq S(C)$, $S(D\rho) \simeq S(D)$ (that is, "corresponding" literals are selected), then there exist C'' and a substitution τ s.t.

$$\begin{array}{c} \dfrac{C \qquad D}{C''} \\ \downarrow \tau \\ C' = C''\tau \end{array} \qquad \text{(inference in } Res_S^{\succ}\text{)}$$

Lifting Lemma for $Res_S^{\succ}$ (cont'd)

- An analogous lifting lemma holds for factoring.

Saturation of General Clause Sets

Corollary 25

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

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

Proof:

We first define the selection function S' such that $S'(C) = S(C)$ for all clauses $C \in G_\Sigma(N) \cap N$. For $C \in G_\Sigma(N) \setminus N$ we choose a fixed but arbitrary clause $D \in N$ with $C \in G_\Sigma(D)$ and define $S'(C)$ to be those occurrences of literals that are ground instances of the occurrences selected by S in D. Then proceed as in the proof of Corollary 17 using the above lifting lemma.

Soundness and Refutational Completeness

Property 26

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

$$N \models \bot \text{ iff } \bot \in N$$

Proof:

The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part consider first the propositional level: Construct a candidate model $I_N^{\succ}$ as for unrestricted resolution, except that clauses C in N that have selected literals are not productive, even when they are false in I_C and when their maximal atom occurs only once and positively. The result for general clauses follows using Corollary 25.

Craig-Interpolation

- A theoretical application of ordered resolution is Craig-Interpolation:

Property 27 (Craig 1957)

Let F and G be two propositional formulae such that $F \models G$. Then there exists a formula H, such that H contains only propositional variables occurring both in F and in G, and such that $F \models H$ and $H \models G$.

- H is called an interpolant for $F \models G$

Craig-Interpolation (cont'd)

Proof:

Translate F and $\neg G$ into CNF. Let N and M, resp., denote the resulting clause sets. 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 the empty selection function S. Then saturate $N^* \cup M$ wrt. $Res_S^{\succ}$ to derive $\bot$.

As N^* is already saturated, due to the ordering restrictions only inferences need to be considered where premises, if they are from N^*, only contain symbols that also occur in G. The conjunction of these premises is an interpolant H.

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

Redundancy

- So far: local restrictions of the resolution inference rules using orderings and selection functions.
- Is it also possible to delete clauses altogether?
  Under which circumstances are clauses unnecessary?
  (Conjecture: e. g., if they are tautologies or if they are subsumed by other clauses.)
- Intuition: If a clause is guaranteed to be neither a minimal counter-example nor productive, then we do not need it.

A Formal Notion of Redundancy

- Let N be a set of ground clauses and C a ground clause (not necessarily in N). C is called redundant wrt. N, if there exist $C_1, \ldots, C_n \in N$, $n \geq 0$, such that all $C_i \prec C$ and $C_1, \ldots, C_n \models C$.
- Redundancy for general clauses: C is called redundant wrt. N, if all ground instances $C\sigma$ of C are redundant wrt. $G_\Sigma(N)$.
- Intuition: Redundant clauses are neither minimal counter-examples nor productive.
- Note: The same ordering $\succ$ is used for ordering restrictions and for redundancy (and for the completeness proof).

Examples of Redundancy

Property 28

(i) C tautology (i.e., $\models C$) $\Rightarrow$ C redundant wrt. any set N.

(ii) $C\sigma \subset D$ $\Rightarrow$ D redundant wrt. $N \cup \{C\}$

(iii) $C\sigma \subseteq D$ $\Rightarrow$ $D \vee \overline{L}\sigma$ redundant wrt. $N \cup \{C \vee L,\ D\}$, where $\overline{L}$ denotes the complement of L

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

Saturation up to Redundancy

- Let $Red(N)$ denote the set of clauses redundant wrt. N.
- N is called saturated up to redundancy (wrt. $Res_S^\succ$) iff

$$Res_S^\succ(N \setminus Red(N)) \subseteq N \cup Red(N)$$

Property 29

Let N be saturated up to redundancy. Then

$$N \models \bot \text{ iff } \bot \in N$$

Saturation up to Redundancy (cont'd)

Proof (Sketch):

(i) Ground case:

- consider construction of candidate model $I_N^\succ$ for $Res_S^\succ$
- redundant clauses are not productive
- redundant clauses in N are not minimal counter-examples for $I_N^\succ$

The premises of "essential" inferences are either minimal counter-examples or productive.

(ii) Lifting: no additional problems over the proof of Property 26.

Monotonicity Properties of Redundancy

Property 30

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

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

Proof: Exercise.

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

CS612: Automated Reasoning II

Lecture 6

Summary

- ordered resolution with selection $Res_S^{\succ}$
  - inferences limited by ordering $\succ$
  - inferences limited by selection function S
- soundness and refutational completeness
- general notion redundancy
  - justifies deletion of clauses
  - standard instances:
    tautology deletion, strict subsumption deletion
- non-ground maximality, saturation up to redundancy

Previously . . .

- Ways of limiting inferences & enhancing efficiency
  - ordering restriction $\succ$
  - selection function S
  - redundancy elimination
- Powerful and versatile

A Resolution Prover

- So far: static view on completeness of resolution:
  - Saturated sets are inconsistent iff they contain $\bot$.
- We will now consider a dynamic view:
  - How can we get saturated sets in practice?
  - The Properties 29 and 30 are the basis for the completeness proof of our prover *RP*.

Rules for Simplifications and Deletion

- We want to employ the following rules for simplification of prover states N:
  - Deletion of tautologies
    $$N \cup \{C \vee A \vee \neg A\} \;\triangleright\; N$$
  - Deletion of subsumed clauses
    $$N \cup \{C,\ D\} \;\triangleright\; N \cup \{C\}$$
    if $C\sigma \subseteq D$ (C subsumes D).
  - Reduction (also called subsumption resolution)
    $$N \cup \{D \vee L,\ C \vee D\sigma \vee \overline{L}\sigma\} \;\triangleright\; N \cup \{D \vee L,\ C \vee D\sigma\}$$

Resolution Prover *RP*

- **3 clause sets:**
  - **N**(ew) containing new conclusions
  - **P**(rocessed) containing simplified resolvents
  - **O**(ld) where clause are put once their inferences have been computed
- **Strategy:** Inferences will only be computed when there are no possibilities for simplification

Transition Rules for *RP* (I)

Tautology elimination

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \;\triangleright\; \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$$

if C is a tautology

Forward subsumption

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \;\triangleright\; \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$$

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

Backward subsumption

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \cup \{D\} \mid \boldsymbol{O} \;\triangleright\; \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$$

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{D\} \;\triangleright\; \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$$

if C strictly subsumes D

Transition Rules for *RP* (II)

Forward reduction

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

if there exists $D \vee L' \in \boldsymbol{P} \cup \boldsymbol{O}$

such that $\overline{L} = L'\sigma$ and $D\sigma \subseteq C$

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 $D\sigma \subseteq C$

Transition Rules for *RP* (III)

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

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

with $\boldsymbol{N} = Res_S^{\succ}(\boldsymbol{O} \cup \{C\})$

Soundness and Completeness

Property 31

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

Proof in

L. Bachmair, H. Ganzinger (2001), "Resolution Theorem Proving" (on H. Ganzinger's Web page under Publications/Journals; published in "Handbook on Automated Reasoning").

Fairness

- Problem:
  - If N is inconsistent, then $N \mid \emptyset \mid \emptyset \;\overset{*}{\rhd}\; N' \cup \{\bot\} \mid \_ \mid \_$.
  - Does this imply that every derivation starting from an inconsistent set N eventually produces $\bot$?
  - No: a clause could be kept in $\boldsymbol{P}$ without ever being used for an inference.

Fairness (cont'd)

- We need in addition a fairness condition:
  - If an inference is possible forever (that is, none of its premises is ever deleted), then it must be performed eventually.
- One possible way to guarantee fairness:
  Implement $\boldsymbol{P}$ as a queue
  (there are other techniques to guarantee fairness).

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

Hyperresolution

- There are many variants of resolution.
  (Refer to Bachmair and Ganzinger (2001), "Resolution Theorem Proving", for further reading.)
- One well-known example is hyperresolution (Robinson 1965):
  - Assume that several negative literals are selected in a clause D.
    If we perform an inference with D, then one of the selected literals is eliminated.
  - Suppose that the remaining selected literals of D are again selected in the conclusion.
  - Then we will eliminate the remaining selected literals one by one by further resolution steps.

Hyperresolution (cont'd)

- Hyperresolution replaces these successive steps by a single inference.
- As for $Res_S^{\succ}$, the calculus is parameterised by an atom ordering $\succ$ and a selection function S.
- But S is the 'maximal' selection function, i.e. selects all negative literals in a clause.

Hyperresolution (cont'd)

- **Hyperresolution calculus** *HRes*

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

  provided $\boldsymbol{\sigma}$ is the mgu s.t. $A_1\boldsymbol{\sigma} = B_1\boldsymbol{\sigma}, \ldots, A_n\boldsymbol{\sigma} = B_n\boldsymbol{\sigma}$, and
  (i) $A_i\boldsymbol{\sigma}$ strictly maximal in $C_i\boldsymbol{\sigma}$, $1 \le i \le n$;
  (ii) nothing is selected in C_i (i.e. C_i is positive);
  (iii) the indicated $\neg B_i$ are exactly the ones selected by S, and D is positive.

- Similarly as for resolution, hyperresolution has to be complemented by a factoring rule.

Hyperresolution (cont'd)

- As we have seen, hyperresolution can be simulated by iterated binary resolution.
- However this yields intermediate clauses which *HRes* might not derive, and many of them might not be extendable into a full *HRes* inference.

Exercise

- Let $P \succ Q \succ R$ and consider:

$$P \vee Q$$
$$R \vee Q$$
$$\neg R \vee \neg P$$

- Give a *HRes* derivation.
- What does the derivation look like under a different ordering?

Reduction orderings

- A strict ordering $\succ$ is a reduction ordering iff
  - (i) $\succ$ is well-founded
  - (ii) $\succ$ is stable under substitutions, i.e.
    $s \succ t$ implies $s\sigma \succ t\sigma$
    for all terms s, t and substitutions σ
  - (iii) $\succ$ is compatible with contexts, i.e.
    $s \succ t$ implies $u[s] \succ u[t]$
    for all terms s, t and contexts u
- Examples:
  - For (ii): $f(x) \succ g(x)$ implies $f(a) \succ g(a)$.
  - For (iii): $a \succ b$ implies $f(a) \succ f(a)$.

Lexicographic Path Orderings

- Let Σ be a finite signature and let X be a countably infinite set of variables.
- Let $\succ$ be a strict ordering (precedence) on the set of predicate and functions symbols in Σ.
- The lexicographic path ordering $\succ_{\mathrm{lpo}}$ on the set of terms (and atoms) over Σ and X is an ordering induced by $\succ$, satisfying:
  $s \succ_{\mathrm{lpo}} t$ iff
  1. $t \in \mathrm{var}(s)$ and $t \neq s$, or
  2. $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and
     - (a) $s_i \succeq_{\mathrm{lpo}} t$ for some i, or
     - (b) $f \succ g$ and $s \succ_{\mathrm{lpo}} t_j$ for all j, or
     - (c) $f = g$, $s \succ_{\mathrm{lpo}} t_j$ for all j, and
       $(s_1, \ldots, s_m)\ (\succ_{\mathrm{lpo}})_{\mathrm{lex}}\ (t_1, \ldots, t_n)$.

Lexicographic Path Orderings (cont'd)

- Definition: $s \succeq_{\mathrm{lpo}} t$ iff $s \succ_{\mathrm{lpo}} t$ or $s = t$
- Examples:
  - $f(x) \succ_{\mathrm{lpo}} x$
  - if t is a subterm of s then $s \succ_{\mathrm{lpo}} t$
  - $f(a, b, g(c), a) \succ_{\mathrm{lpo}} f(a, b, c, g(b))$
  - If t can be homomorphically embedded into s and $s \neq t$ then $s \succ_{\mathrm{lpo}} t$
    E.g.
    $$h(f(g(a), f(b, y))) \succ_{\mathrm{lpo}} f(g(a), y)$$

Properties of LPOs

Lemma 32

$s \succ_{\mathrm{lpo}} t$ implies $\mathrm{var}(s) \supseteq \mathrm{var}(t)$.

Proof:
By induction on $|s| + |t|$ and case analysis.

Properties of LPOs (cont'd)

Property 33

$\succ_{\mathrm{lpo}}$ is a reduction ordering on the set of terms (and atoms) over Σ and X.

Proof:
Show transitivity, stability under substitutions, compatibility contexts, and irreflexivity, usually by induction on the sum of the term sizes and case analysis.
Details: Baader and Nipkow, page 119–120.

Properties of LPOs (cont'd)

Property 34

If the precedence $\succ$ is total, then the lexicographic path ordering $\succ_{\mathrm{lpo}}$ is total on ground terms (and ground atoms), i.e. for all ground terms (or atoms) s, t of the following is true:
$s \succ_{\mathrm{lpo}} t$ or $t \succ_{\mathrm{lpo}} s$ or $s = t$.

Proof:
By induction on $|s| + |t|$ and case analysis.

Variations of the LPO

There are several possibilities to compare subterms in 2.(c):

- compare list of subterms lexicographically left-to-right ("lexicographic path ordering (lpo)", Kamin and Lvy)
- compare list of subterms lexicographically right-to-left (or according to some permutation π)
- compare multiset of subterms using the multiset extension ("multiset path ordering (mpo)", Dershowitz)
- to each function symbol f/n associate a status $\in \{mul\} \cup \{ lex_\pi \mid \pi : \{1, \ldots, n\} \to \{1, \ldots, n\} \}$ and compare according to that status ("recursive path ordering (rpo) with status")

CS612: Automated Reasoning II

Lecture 7

Summary

- Implementing a (sound & complete) resolution prover
  - Specific redundancy elimination & simplification rules
  - Transition rules for deduction, deletion & simplification steps
  - Fairness
- Hyperresolution *HRes*

  $=$ ordered resolution with maximal selection
- Lexicographic path orderings

Previously . . .

- Advanced techniques of resolution theorem proving
- Implementing a resolution theorem prover

Example: Neuman-Stubblebine Protocol

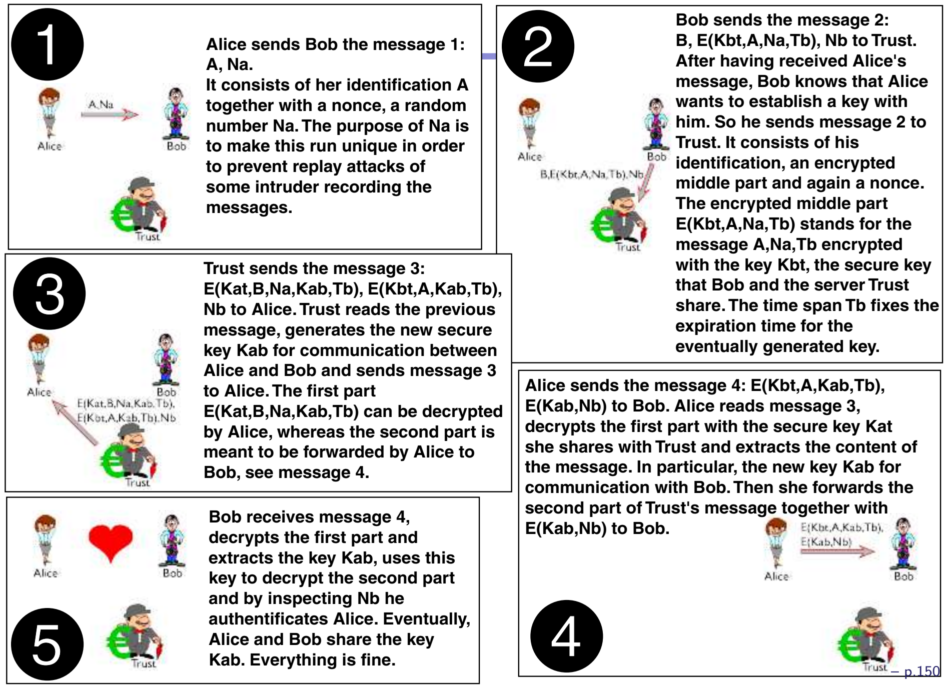
- Formalisation of a concrete application:
  Neuman-Stubblebine key exchange protocol.
- Proof by refutation:
  inconsistency $\Rightarrow$ intruder can break the protocol.
- Proof by consistency:
  consistency $\Rightarrow$ no unsafe states exist.
- Termination requires elimination of redundancy.

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 be used successfully to analyze the Neuman-Stubblebine key exchange protocol [1]. 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 Problem (cont'd)

- The animation successively shows two runs of the Neuman-Stubblebine [1] key exchange protocol. The first run works the way the protocol is designed to do, i.e. it establishes a secure key K_{ab} between Alice and Bob.
- 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 N_a 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 next picture shows a sequence of four message exchanges that eventually establishes the key.



Neuman-Stubblebine: A Regular Run

1. $A \xrightarrow{A,\ N_a} B$

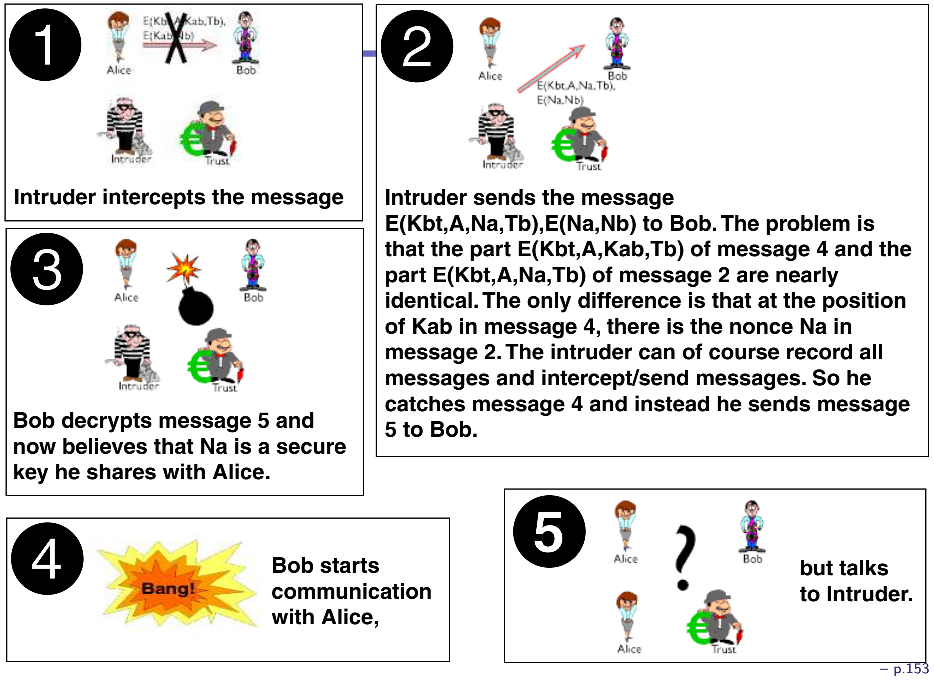
2. $B \xrightarrow{B,\ N_b,\ E_{K_{bt}}(A,\ N_a,\ T_b)} T$

3. $T \xrightarrow{E_{K_{at}}(B,\ N_a,\ K_{ab},\ T_b),\ E_{K_{bt}}(A,\ K_{ab},\ T_b),\ N_b} A$

4. $A \xrightarrow{E_{K_{bt}}(A,\ K_{ab},\ T_b),\ E_{K_{ab}}(N_b)} B$

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 it know any of the initial keys K_{at} or K_{bt}.
- Here is the solution ...



Breaking Neuman-Stubblebine

1. $A \xrightarrow{A,\ N_a} B$

2. $B \xrightarrow{B,\ N_b,\ E_{K_{bt}}(A,\ N_a,\ T_b)} T$

3. $T \xrightarrow{E_{K_{at}}(B,\ N_a,\ K_{ab},\ T_b),\ E_{K_{bt}}(A,\ K_{ab},\ T_b),\ N_b} A$

4. $A \xrightarrow{E_{K_{bt}}(A,\ K_{ab},\ T_b),\ E_{K_{ab}}(N_b)} B$

3. $I \xrightarrow{E_{K_{bt}}(A,\ N_a,\ T_b),\ E_{N_a}(N_b)} B$

The Formalisation

- The key idea of the formalisation 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 remaining symbols are introduced and explained with their first appearance in a formula.)
- Then the four messages can be translated into the following formulae . . .

The Formalisation: Step 1

- **Step 1)** $A \longrightarrow B : A, Na$

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

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

- (1) expresses that initially Alice holds the *key* *at* for communication with *t* (for Trust). (2) states 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.

The Formalisation: Step 2

- **Step 2)** $B \longrightarrow T : B, Nb, E_{Kbt}(A, Na, Tb)$

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

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

- 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 message 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*.

The Formalisation: Step 3

- **Step 3)** $T \longrightarrow A : E_{Kat}(B, Na, Kab, Tb),\ \ E_{Kbt}(A, Kab, Tb),\ \ Nb$

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

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

- Trust holds the keys for Alice and Bob and answers appropriately to a message in the format of message 2.
- Note: 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*.

The Formalisation: Step 4

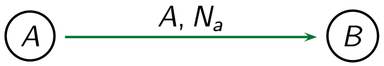
- **Step 4)** $A \longrightarrow B : E_{Kbt}(A, Kab, Tb),\ \ E_{Kab}(Nb)$

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

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

- 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.

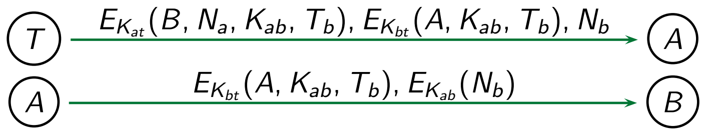
A's Formalisation Part I



$$P(a)$$
$$Ak(key(at, t))$$
$$M(sent(a, b, pair(a, na)))$$
$$Sa(pair(b, na))$$

- *Sa* is Alice's local store that will eventually be used to verify the nonce when it is sent back to her in Step (3).

A's Formalisation Part II



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

The Intruder

- The Intruder is modeled as an exhaustive hacker. It records all messages, decomposes the messages as far as possible and generates all possible new compositions.
- Furthermore, any object it has in hand is considered as a key and tried 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:

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

The Intruder

- It decomposes and decrypts all messages it owns the key for:

$$\forall u, v\ [Im(pair(u, v)) \longrightarrow Im(u) \wedge Im(v)] \quad (11)$$

$$\forall u, v, w\ [Im(triple(u, v, w)) \longrightarrow Im(u) \wedge Im(v) \wedge Im(w)] \quad (12)$$

$$\forall u, v, w, z\ [Im(quadr(u, v, w, z)) \longrightarrow Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z)] \quad (13)$$

$$\forall u, v, w\ [Im(encr(u, v)) \wedge Ik(key(v, w)) \longrightarrow Im(u)] \quad (14)$$

- It composes all possible messages:

$$\forall u, v\ [Im(u) \wedge Im(v) \longrightarrow Im(pair(u, v))] \quad (15)$$

$$\forall u, v, w\ [Im(u) \wedge Im(v) \wedge Im(w) \longrightarrow Im(triple(u, v, w))] \quad (16)$$

$$\forall u, v, w, x\ [Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(x) \longrightarrow Im(quadr(u, v, w, x))] \quad (17)$$

The Intruder

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

$$\forall v, w\ [Im(v) \wedge P(w) \longrightarrow Ik(key(v, w))] \quad (18)$$

$$\forall u, v, w\ [Im(u) \wedge Ik(key(v, w)) \wedge P(w) \longrightarrow Im(encr(u, v))] \quad (19)$$

- It sends everything:

$$\forall x, y, u\ [P(x) \wedge P(y) \wedge Im(u) \longrightarrow M(sent(x, y, u))] \quad (20)$$

- 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:

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

SPASS Solves the Problem

- Now the protocol formulae 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 formalisation.
  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 formalisation without type checks, we are currently not aware of any prover that can disprove the insecurity formula in the formalisation 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 or the CS612 directory.

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.
  - On the non-ground level, ordering constraints can (only) be solved approximatively.
- Completeness proof by constructing candidate models
  - from productive clauses $C \vee A$, $A \succ C$; inferences with these reduce counter-examples.

CS612: Automated Reasoning II

Lecture 8

Summary: Resolution Theorem Proving

- Local restrictions of inferences via $\succ$ and S

  $\Rightarrow$ fewer proof variants;

  $\Rightarrow$ justification for numerous standard refinements;

  $\Rightarrow$ simulation of certain kinds of tableaux.
- Global restrictions of the search space via elimination of redundancy

  $\Rightarrow$ computing with "smaller" clause sets;

  $\Rightarrow$ termination on many decidable fragments.
- Further specialisation of inference systems required for reasoning with orderings, equality and specific algebraic theories (lattices, abelian groups, rings, fields)

Previously . . .

- Resolution theorem proving
- Construction of candidate models

Semantic Tableaux

- Literature:
  - M. Fitting (1996), First-Order Logic and Automated Theorem Proving. Springer, Chapters 3, 6, 7.
  - R. M. Smullyan (1968), First-Order Logic. Dover Publ., New York, revised 1995.

- Like resolution, semantic tableaux were developed in the sixties, by R. M. Smullyan on the basis of work by Gentzen in the 1930s and of Beth in the 1950s.
- (According to Fitting, semantic tableaux were first proposed by the Polish scientist Z. Lis in (1960) (Studia Logica 10), that was only recently rediscovered.)

Idea for the propositional case

- A set $N \cup \{F \wedge G\}$ of formulae has a model iff $N \cup \{F \wedge G,\ F,\ G\}$ has a model.
- A set $N \cup \{F \vee G\}$ of formulae has a model iff $N \cup \{F \vee G,\ F\}$ or $N \cup \{F \vee G,\ G\}$ has a model.
- Similarly for other connectives.

- To avoid duplication, represent sets as paths of a tree.
- Continue splitting until two complementary formulae are found $\Rightarrow$ inconsistency detected.

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

1. $P \wedge \neg(Q \vee \neg R)$
2. $\neg Q \vee \neg R$

3. $\neg Q$	4. $\neg R$
5. P	10. P
6. $\neg(Q \vee \neg R)$	11. $\neg(Q \vee \neg R)$
7. $\neg Q$	
8. $\neg\neg R$	
9. R	

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

Properties of tableau calculi

- Tableau calculi are:
  - analytic: inferences according to the logical content of the symbols.
  - goal-oriented: inferences operate directly on the goal to be proved (unlike, e. g., ordered resolution).
  - global: some inferences affect the entire proof state (set of formulae), as we will see later.

Application of Expansion Rules

- Expansion rules are applied to the formulae 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.

Propositional Tableau Expansion Rules

- **Negation Elimination**

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

- **α-Expansion**
  For formulae that are essentially conjunctions:
  append subformulae α_1 and α_2 one on top of the other

$$\frac{\alpha}{\begin{matrix}\alpha_1 \\ \alpha_2\end{matrix}}$$

Propositional Tableau Expansion Rules (cont'd)

- **β-Expansion**
  For formulae that are essentially disjunctions:
  append β_1 and β_2 horizontally, i.e. branch into β_1 and β_2

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

Classification of Formulae

- Definition of α- and β-formulae:

conjunctive			disjunctive		
α	α_1	α_2	β	β_1	β_2
$F \wedge G$	F	G	$\neg(F \wedge G)$	$\neg F$	$\neg G$
$\neg(F \vee G)$	$\neg F$	$\neg G$	$F \vee G$	F	G
$\neg(F \rightarrow G)$	F	$\neg G$	$F \rightarrow G$	$\neg F$	G

- We assume that the binary connective $\leftrightarrow$ has been eliminated in advance.

Definition of Semantic Tableau

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

(i) The tree consisting of a single branch

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

is a tableau for $\{F_1, \ldots, F_n\}$.
(We do not draw edges if nodes have only one successor.)

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

Definition of Semantic Tableau (cont'd)

- A branch (i.e. a path from the root to a leaf) in a tableau is called closed, if it either contains $\perp$, or else it contains both F and $\neg F$ for some formula F.
  Otherwise the branch is called open.
- A tableau is called closed, if all branches are closed.
- A tableau proof for F is a closed tableau for $\{\neg F\}$.

Further Notions

- A branch $\mathcal{B}$ in a tableau is called maximal (or complete), if for each non-atomic formula F on $\mathcal{B}$ there exists a node in $\mathcal{B}$ at which the expansion rule for F has been applied.
- In that case, if F is a formula on $\mathcal{B}$, $\mathcal{B}$ also contains:

(i) F_1 and F_2, if F is an α-formula,

(ii) F_1 or F_2, if F is a β-formula, and

(iii) F', if F is a negated formula, where F' the conclusion of the corresponding elimination rule.

- A tableau is called maximal (or complete), if each branch is closed or maximal.

Further Notions (cont'd)

- A tableau is called strict, if for each formula the corresponding expansion rule has been applied at most once on each branch containing that formula.
- A tableau is called clausal, if each of its formulae is a clause.

A Sample Proof

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

1. $\neg\Big[\big(P \rightarrow (Q \rightarrow R)\big) \rightarrow \big((P \vee S) \rightarrow ((Q \rightarrow R) \vee S)\big)\Big]$
2. $P \rightarrow (Q \rightarrow R)$ $[\alpha, 1_1]$
3. $\neg\big((P \vee S) \rightarrow ((Q \rightarrow R) \vee S)\big)$ $[\alpha, 1_2]$
4. $P \vee S$ $[\alpha, 3_1]$
5. $\neg((Q \rightarrow R) \vee S))$ $[\alpha, 3_2]$
6. $\neg(Q \rightarrow R)$ $[\alpha, 5_1]$
7. $\neg S$ $[\alpha, 5_2]$

8. $\neg P$ $[\beta, 2_1]$ 9. $Q \rightarrow R$ $[\beta, 2_2]$

10. P $[\beta, 4_1]$ 11. S $[\beta, 4_2]$

There are three branches, each of them closed.

Properties of Propositional Tableaux

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

Property 35

$\{F_1, \ldots, F_n\}$ satisfiable iff some branch (i.e. the set of its formulae) in T is satisfiable.

(Proof by induction over the structure of T.)

Corollary 36

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

Properties of Propositional Tableaux (cont'd)

Property 37

Let T be a strict propositional tableau. Then T is finite.

Proof:

New formulae resulting from expansion are either $\perp$, $\top$ or subformulae of the expanded formula.

By strictness, on each branch a formula can be expanded at most once.

Therefore, each branch is finite, and a finitely branching tree with finite branches is finite (König's Lemma).

- Conclusion: Strict and maximal tableaux can be effectively constructed

  $\Rightarrow$ provides decision procedure for propositional logic

Refutational Completeness

Property 38

Let $\mathcal{B}$ be a maximal, open branch in a tableau. Then the set of formulae on $\mathcal{B}$ is satisfiable.

Proof (we consider only the case of a clausal tableau):

Let N be the set of formulae on $\mathcal{B}$. As $\mathcal{B}$ is open, $\perp$ is not in N.

Let $C \vee A$ and $D \vee \neg A$ be two resolvable clauses in N. One of the two subclauses C or D, C say, is not empty, as otherwise $\mathcal{B}$ would be closed.

Since $\mathcal{B}$ is maximal, the β-rule was applied on $C \vee A$. Therefore, $\mathcal{B}$ (and N) contains a proper subclause of $C \vee A$, and hence $C \vee A$ is redundant w.r.t. N.

By the same reasoning, if N contains a clause that can be factored, that clause must be redundant w.r.t. N.

In other words, N is saturated up to redundancy wrt. *Res*(olution). Now apply Property 12 to prove satisfiability of N.

Refutational Completeness (cont'd)

Property 39
$\{F_1, \ldots, F_n\}$ satisfiable iff there exists no closed strict tableau for $\{F_1, \ldots, F_n\}$.

Proof:
One direction is clear by Property 35. For the reverse direction, let T be a strict, maximal tableau for $\{F_1, \ldots, F_n\}$ and let $\mathcal{B}$ be an open branch in T. By the previous theorem, the set of formulae on $\mathcal{B}$, and hence by Property 35 the set $\{F_1, \ldots, F_n\}$, is satisfiable.

Tableau Algorithm for Propositional Logic

- The validity of a propositional formula F can be established by constructing a strict, maximal tableau for $\{\neg F\}$:
  - T closed $\Leftrightarrow$ F valid.
- Algorithm for checking validity of F, or (un)satisfiability of $\neg F$:
  1. Start with $\neg F$.
  2. Repeatedly apply the expansion rules to the branches.
  3. Stop expanding a branch when either:
     (a) the branch is closed, or
     (b) the branch is maximal, open, i.e. no more non-redundant inferences are possible.
  4. Unless a maximal, open branch was found, or all branches are closed continue with step 2.

Consequences

- It suffices to test complementarity of branches wrt. atomic formulae (cf. reasoning in the proof of Property 38).
- 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 branches wrt. ordered resolution (up to redundancy), tableau expansion rules can be even more restricted, in particular by certain ordering constraints.

Summary

- Semantic tableau
  - negation elimination expansion rules
  - α-expansion rules
  - β-expansion rules
- closed branch/tableau, open branch/tableau, maximal branch/tableau, strict tableau
- soundness & refutational completeness
- tableau decidability of propositional logic
- contruction of tableau derivations

CS612: Automated Reasoning II

Lecture 9

Previously ...

- Propositional semantic tableau
  - negation elimination expansion rules
  - α-expansion rules
  - β-expansion rules
- closedness, openness, maximality, strictness

Semantic Tableaux for First-Order Logic

- Additional classification of quantified formulae:

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 Tableau Expansion Rules

- **γ-expansion**

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

- **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

Notes

- Skolemisation becomes part of the calculus and needs not necessarily be applied in a preprocessing step.
  Of course, one could do Skolemisation 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 branch to any formula on the branch.
- In this form the rules go back to Hähnle and Schmitt (1994), The Liberalized δ-Rule in Free Variable Semantic Tableaux. In J. Automated Reasoning 13(2), 211–221.

Definition: Free-Variable Tableau

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

(i) The tree consisting of a single branch

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

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 $\{F_1, \ldots, F_n\}$.

Notes on Free-Variable Tableau

- The substitution rule (iii) may, potentially, modify all the formulae of a tableau.
- This feature is what makes the tableau method a global proof method. (Resolution, by comparison, is a local method.)
- If one took (iii) literally, by repeated application of γ-rule, one could enumerate all substitution instances of the universally quantified formulae.
- That would be a major drawback compared with resolution. Fortunately, we can improve on this.

Example

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

- 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 $\sigma = \{a/v_1,\ b(a)/v_2,\ f(b(a), a)/v_3\}$.
- Hence, $T\sigma$ is a closed (linear) tableau for the formula in 1.

AMGU-Tableaux

- Idea: Restrict the substitution rule to unifiers of complementary formulae.
- We speak of an AMGU-Tableau, whenever the following A(tomic)MGU (substitution) rule is applied:

  The substitution rule is only applied for substitutions σ for which there is a branch in T containing two literals $\neg A$ and B such that $\sigma = \text{mgu}(A, B)$.

Correctness

- Given a signature Σ, by $\Sigma^{\mathbf{sko}}$ we denote the result of adding infinitely many new Skolem function symbols which we may use in the δ-rule.
- Let $\mathcal{M}$ be a $\Sigma^{\mathbf{sko}}$-interpretation, T a tableau, and s a variable assignment over $\mathcal{M}$.
- T is called $(\mathcal{M}, s)$-valid, if there is a branch $\mathcal{B}_s$ in T such that $\mathcal{M}, s \models F$, for each formula F on $\mathcal{B}_s$.
- T is called satisfiable if there exists a structure $\mathcal{M}$ such that for each assignment s the tableau T is $(\mathcal{M}, s)$-valid.
  (This implies that we may choose $\mathcal{B}_s$ depending on s.)

Correctness (cont'd)

Property 40

Let T be a tableau for $\{F_1, \ldots, F_n\}$, where the F_i are closed Σ-formulae. Then

$$\{F_1, \ldots, F_n\} \text{ is satisfiable iff } T \text{ is satisfiable.}$$

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

Meaning of Strictness?

- Forbidding re-use a γ-formulae is incomplete:

  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

- If we placed a once-only restriction on applications to γ formulae, the tableau is now only be expandable by the substitution rule.
- However, there is no substitution (for v_1) that can close both branches simultaneously.

Multiple Application of γ Solves the Problem

- Apply γ-rule again with 2:

$$
\begin{array}{lccc}
1. & & \neg[\forall x\ p(x) \longrightarrow (p(a) \wedge p(b))] & \\
2. & & \forall x\ p(x) & 1_1 \\
3. & & \neg(p(a) \wedge p(b)) & 1_2 \\
4. & & p(v_1) & 2_{v_1}
\end{array}
$$

$$
\begin{array}{lll|lll}
5. & \neg p(a) & 3_1 & 6. & \neg p(b) & 3_2 \\
 & & & 7. & p(v_2) & 2_{v_2}
\end{array}
$$

- 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 $\{a/v_1,\ b/v_2\}$ which closes the tableau.

Strictness in AMGU-Tableau

- 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 branch.

Refutational Completeness

Property 41

$\{F_1, \ldots, F_n\}$ satisfiable iff there exists no closed, strict AMGU-tableau for $\{F_1, \ldots, F_n\}$.

Proof outline:

One defines a fair tableau expansion process converging against an infinite tableau where on each branch each γ-formula is expanded into all its variants (modulo the choice of the free variable).
One may then again show that each branch 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.

How Often Do we Have to Apply γ?

Property 42

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

Otherwise unsatisfiability 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.
$\forall$ is treated like an infinite conjunction: By repeatedly applying γ, together with the substitution rule, one can enumerate all instances $F[t/x]$ vertically, that is, conjunctively, in each branch containing $\forall xF$.

Summary

- Free-variable tableau
  - rules of propositional tableau
  - γ-expansion rules
  - δ-expansion rules
  - substitution rule
- AMGU tableau
  - AMGU substitution rule
- strictness in first-order tableau
  - re-use of γ formulae
- soundness & refutational completeness

Semantic Tableaux vs. Resolution

- Both methods are machine methods upon which present day provers are based.
- Tableaux: global, goal-oriented, "backward" approach.
- Resolution: local, "forward" approach.
- Goal-orientation is a clear advantage if only a small subset of a large set of formulae is necessary for a proof. In general resolution provers saturate also those parts of the clause set that are irrelevant for proving the goal.
- Resolution is good for:
  - saturation of theories, yielding simplified theories
  - developing decision procedures

Semantic Tableaux vs. Resolution

- 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.
- Resolution can be combined with more powerful redundancy elimination methods.
- Because of its global nature redundancy elimination is more difficult for the tableau method.
- Resolution can be refined to work well with equality (beyond the scope of this course) and algebraic structures; for tableaux this seems to be harder, if not impossible.

Current research

- Resolution decision procedures for solvable fragments of f.-o. logic
- Automated reasoning for specific theories (equality, transitive relations, lattices, groups, . . .)
- Implementing fast automated theorem provers (optimisations, heuristics, experimentation, . . .)
- Relationship and comparison of different proof methods
- Combination of different provers; combination with model checkers
- Automated reasoning with distributed, heterogeneous, dynamic information
- Automated reasoning for description logics, modal logics
- Automated reasoning for ontologies and the semantic web
- Program and hardware design & verification
- . . .

Research at Manchester (incomplete summary)

- FM
  - automated theorem proving, resolution, other methods
  - decision procedures, model generation, s.o. quantifier elimination
  - verification, model checking
  - first-order logic, modal logics, description logics, program logics, temporal logics
  - Vampire, MSPASS, ...
- IMG
  - ground tableau, decision procedures
  - semantic web, ontologies, description logics
  - FACT, ICOM, ...

Notation in Part II

- p, q, P, Q predicate symbols
- x, y, z variables
- a, b, c constants
- f, g function symbols
- s, t terms
- A, B atoms
- L literals
- C, D clauses
- N sets of clauses
- F, G formulae
- σ substitutions
- Σ given signature
- X given set of variables
- T_Σ terms over signature
- s/x substitution of term s into variable x
- $\mathcal{M}$ f.o. interpretation (mapping)
- $\overline{L}$ complement of literal L

CS612: Automated Reasoning II

Appendix