CogSysI Lecture 8: Automated Theorem Proving

Intelligent Agents
WS 2004/2005

Part II: Inference and Learning

Automated Theorem Proving
Remember ...

... in the last lecture we started to introduce resolution.

- Resolution calculus is a basic approach for performing logical proofs on a machine.
- Logical formula must be rewritten into clause form, using equivalence rules.
- To perform a resolution step on a pair of clauses, literals must be unified.
Clause Form

- Conjunctive Normalform (CNF): Conjunction of disjunctions of literals
\[ \wedge_{i=1}^{n} (\vee_{j=1}^{m} L_{ij}) \]

- Clause Form: Set of disjunctions of literals (can be generated from CNF)

Rewriting of formulas to clause form:
8 steps, illustrated with example

\[ \forall x [B(x) \rightarrow (\exists y [O(x, y) \wedge \neg P(y)]) \wedge \neg \exists y [O(x, y) \wedge O(y, x)] \wedge \forall y [\neg B(y) \rightarrow \neg E(x, y))] \]
Clause Form cont.

(1) Remove Implications
\[ \forall x [ \neg B(x) \lor (\exists y [O(x, y) \land \neg P(y)]) \land \neg \exists y [O(x, y) \land O(y, x)] \land \forall y [\neg (\neg B(y)) \lor \neg E(x, y)] ) ] \]

(2) Reduce scopes of negation
\[ \forall x [ \neg B(x) \lor (\exists y [O(x, y) \land \neg P(y)]) \land \forall y [\neg O(x, y) \lor \neg O(y, x)] \land \forall y [B(y) \lor \neg E(x, y)] ) ] \]

(3) Skolemization (remove existential quantifiers)
Replace existentially quantified variables by constant/function symbols.
\[ \exists x \ p(x) \text{ becomes } p(C) \]
(“There exists a human who is a student.” is satisfiable if there exists a constant in the universe \( \mathcal{U} \) for which the sentence is true.
“Human \( C \) is a student.” is satisfiable if the constant symbol \( C \) can be interpreted such that relation \( p \) is true.)
Clause Form cont.

Skolemization cont.
If an existentially quantified variable is in the scope of a
universally quantified variable, it is replaced by a function
symbol dependent of this variable:
\[
\forall x \exists y \ p(x) \land q(x, y) \text{ becomes } \forall x \ p(x) \land q(x, f(x))
\]
(“For all \(x\) holds, \(x\) is a positive integer and there exists a \(y\) which is
greater than \(x\).” is satisfiable if for each \(x\) exists an \(y\) such that the
relation “greater than” holds. E.g., \(f(x)\) can be interpreted as
successor-function.)

Skolemization is no equivalence transformation. A formula
and its Skolemization are only equivalent with respect to
satisfiability! The skolemized formula has a model iff the
orginal formula has a model.
\[
\forall x [ \neg B(x) \lor ((O(x, f(x)) \land \neg P(f(x)))) \land \forall y [ \neg O(x, y) \lor 
\neg O(y, x) ] \land \forall y [ B(y) \lor \neg E(x, y) ] ]
\]
(4) Standardize variables ("bounded renaming")
A variable bound by a quantifier is a "dummy" and can be renamed. Provide that each variable of universal quantor has a different name. (Problematic case: free variables)
\[ \forall x [\neg B(x) \lor ((O(x, f(x)) \land \neg P(f(x)))) \land \forall y [\neg O(x, y) \lor \neg O(y, x)] \land \forall z [B(z) \lor \neg E(x, z)] ] \]

(5) Prenex-form
Move universal quantifiers to front of the formula.
\[ \forall x \forall y \forall z [B(x) \lor ((O(x, f(x)) \land \neg P(f(x)))) \land (\neg O(x, y) \lor \neg O(y, x)) \land (B(z) \lor \neg E(x, z))] \]

(6) CNF
(Repeatedly apply the distributive laws)
\[ \forall x \forall y \forall z [(-B(x) \lor O(x, f(x))) \land (-B(x) \lor \neg P(f(x))) \land (-B(x) \lor \neg O(x, y) \lor \neg O(y, x)) \land (-B(x) \lor B(z) \lor \neg E(x, z))] \]
Clause Form cont.

(7) Eliminate Conjunctions

If necessary, rename variable such that each disjunction has a different set of variables.

The truth of a conjunction entails that all its parts are true.

\[ \forall x \neg B(x) \lor O(x, f(x)) , \forall w \neg B(w) \lor \neg P(f(w)) , \forall u \forall y \neg B(u) \lor \neg O(u, y) \lor \neg O(y, u) , \forall v \forall z \neg B(v) \lor B(z) \lor \neg E(v, z) \]

(8) Eliminate Universal Quantifiers

Clauses are implicitly universally quantified.

\[ M = \{ \neg B(x) \lor O(x, f(x)), \neg B(w) \lor \neg P(f(w)), \neg B(u) \lor \neg O(u, y) \lor \neg O(y, u), \neg B(v) \lor B(z) \lor \neg E(v, z) \} \]
Substitution

A substitution is a set \( \theta = \{ v_1 \leftarrow t_1, \ldots, v_n \leftarrow t_n \} \) of replacements of variables \( v_i \) by terms \( t_i \).

If \( \theta \) is a substitution and \( E \) an expression, \( E' = E \theta \) is called instance of \( E \). \( E' \) was derived from \( E \) by applying \( \theta \) to \( E \).

Example: \( E = p(x) \lor (\neg q(x, y) \land p(f(x))) \), \( \Theta = \{ x \leftarrow C \} \), \( E\Theta = p(C) \lor (\neg q(C, y) \land p(f(C))) \).

Special case: alphabetic substitution (variable renaming).

Composition of substitutions: Let be
\[
\theta = \{ u_1 \leftarrow t_1, \ldots, u_n \leftarrow t_n, v_1 \leftarrow s_1, \ldots, v_k \leftarrow s_k \} \quad \text{and}
\sigma = \{ v_1 \leftarrow r_1, \ldots, v_k \leftarrow r_k, w_1 \leftarrow q_1, \ldots, w_m \leftarrow q_m \}.
\]
The composition is defined as \( \theta\sigma \overset{\text{Def}}{=} \{ u_1 \leftarrow t_1\sigma, \ldots, u_n \leftarrow t_n\sigma, v_1 \leftarrow s_1\sigma, \ldots, v_k \leftarrow s_k\sigma, w_1 \leftarrow q_1, \ldots, w_m \leftarrow q_m \} \).

Composition of substitutions is not commutative!
Unification

Let be \( \{E_1 \ldots E_n\} \) a set of expressions. A substitution \( \theta \) is a unificator of \( E_1 \ldots E_n \), if \( E_1 \theta = E_2 \theta \ldots = E_n \theta \).

A unificator \( \theta \) is called most general unifier (mgu), if for each other unificator \( \sigma \) for \( E_1 \ldots E_n \) there exists a substitution \( \gamma \) with \( \sigma = \theta \gamma \).

Theorem: If exists a unificator, then exists an mgu.

There are lots of unification algorithms, e.g. one proposed by Robinson.
Examples

(1) \{P(x), P(A)\}  \quad \theta = \{x \leftarrow A\}

(2) \{P(f(x), y, g(y)), P(f(x), z, g(x))\}  \quad \theta = \{y \leftarrow x, z \leftarrow x\}

(3) \{P(f(x, g(A, y)), g(A, y)), P(f(x, z), z)\}  \quad \theta = \{z \leftarrow g(A, y)\}

(4) \{P(x, f(y), B), P(x, f(B), B)\}  \quad \theta = \{x \leftarrow A, y \leftarrow B\}
\quad \sigma = \{y \leftarrow B\}

In (4) holds:
\sigma \text{ is more general than } \theta: \theta = \sigma \gamma, \text{ with } \gamma = \{x \leftarrow A\}
\sigma \text{ is mgu for } \{P(x, f(y), B), P(x, f(B), B)\}
Resolution

A clause \( C = \bigvee_{i=1}^{n} L_i \) can be written as set \( C = \{L_1, \ldots L_n\} \).

Let be \( C_1, C_2 \) and \( R \) clauses. \( R \) is called resolvent of \( C_1 \) and \( C_2 \) if:

- There are alphabetical substitutions \( \sigma_1 \) und \( \sigma_2 \) such that \( C_1 \sigma_1 \) and \( C_2 \sigma_2 \) have no common variables.

- There exists a set of literals \( L_1, \ldots L_m \in C_1 \sigma_1 (m \geq 1) \)
  and \( L'_1, \ldots L'_n \in C_2 \sigma_2 (n \geq 1) \) such that
  \( L = \{-L_1, -L_2, \ldots -L_m, L'_1, L'_2, \ldots L'_n\} \) are uniﬁable with \( \theta \)
  as mgu of \( L \).

- \( R \) has the form:
  \[
  R = ((C_1 \sigma_1 \setminus \{L_1, \ldots L_m\}) \cup (C_2 \sigma_2 \setminus \{L'_1, \ldots L'_n\})) \theta. 
  \]
Resolution cont.

Derivation of a clause by application of the resolution rule can be described by a *refutation tree*:

```
  C1       C2
   \       /   \\
    R     C3
       \   / \\
        R'  \\
```

CogSysI Lecture 8: Automated Theorem Proving – p. 211
$C_1 = \{ P(f(x)), \neg Q(z), P(z) \}$

$C_2 = \{ \neg P(x), R(g(x), A) \}$

$\sigma_1 = \{ \}, \sigma_2 = \{ x \leftarrow u \}$

$L = \{ P(f(x)), P(z), \neg \neg P(x) \} = \{ P(f(x)), P(z), P(u) \}$

$\theta = \{ z \leftarrow f(x), u \leftarrow f(x) \}$

$R = \left[ \left( \{ P(f(x)), \neg Q(z), P(z) \} \setminus \{ P(f(x)), P(z) \} \right) \cup \left( \{ \neg P(u), R(g(u), A) \} \setminus \{ P(u) \} \right) \right] \theta = \{ \neg Q(f(x)), R(g(f(x)), A) \}$
Resolution Proofs

To prove that formula $G$ (assertion) logically follows from a set of formula (axioms) $F_1 \ldots F_n$: Include the negated assumption in the set of axioms and try to derive a contradiction (empty clause).

Theorem: A set of clauses is not satisfiable, if the empty clause ($\square$) can be derived with a resolution proof.

(Contradiction: $C_1 = A$, $C_2 = \neg A$, stands for $(A \land \neg A)$ and $(A \land \neg A) \vdash \square$)
Example

Axiom “All humans are mortal” and fact “Socrates is human”
(both are non-logical: their truth is presupposed)

Assertion “Sokrates is mortal.”

Formalization:

\[ F_1 : \forall x \text{ Human}(x) \rightarrow \text{Mortal}(x) \]
\[ F_2 : \text{Human}(S) \]
\[ F_3 : \neg \text{Mortal}(S) \] (negation of assertion)

Clause form:

\[ F_1' : \neg \text{Human}(x) \lor \text{Mortal}(x) \]
\[ F_2' : \text{Human}(S) \]
\[ F_3' : \neg \text{Mortal}(S) \]
Soundness and Completeness of Res.

A calculus is **sound**, if only such conclusions can be derived which also hold in the model.

A calculus is **complete**, if all conclusions can be derived which hold in the model.

The resolution calculus is sound and refutation complete. Refutation completeness means, that if a set of formula (clauses) is unsatisfiable, then resolution will find a contradiction. Resolution cannot be used to generate all logical consequences of a set of formula, but it can establish that a given formula is entailed by the set. Hence, it can be used to find all answers to a given question, using the “negated assumption” method.
Remarks

The proof ideas will be given for resolution for propositional logic (or ground clauses) only. For FOL, additionally, a lifting lemma is necessary and the proofs rely on Herbrand structures.

We cover elementary concepts of logic only.

For more details, see


Volker Sperschneider & Grigoris Antoniou, Logic – A foundation for computer science, Addison-Wesley, 1991.
Resolution Theorem

**Theorem:** A set of clauses $F$ is not satisfiable iff the empty clause $\square$ can be derived from $F$ by resolution.

- **Soundness:** (Proof by contradiction)
  Assume that $\square$ can be derived from $F$. If that is the case, two clauses $C_1 = \{L\}$ and $C_2 = \{\neg L\}$ must be contained in $F$. Because there exists no model for $L \land \neg L$, $F$ is not satisfiable.

- **Refutation completeness:** (Proof by induction over the number $n$ of atomic formulas in $F$)
  Assume that $F$ is a set of formula which is not satisfiable. Because of the compactness theorem, it is enough to consider the case that a finite non-satisfiable subset of formula exists in $F$. To show: $\square$ is derived from $F$. (see e.g., Schöning)
Resolution Strategies

- In general, there are many possibilities, to find two clauses, which are resolvable. Of the many alternatives, there are possibly only a few which help to derive the empty clause $\rightarrow$ combinatorial explosion!

- For feasible algorithms: use a resolution strategy

- E.g., exploit subsumption to keep the knowledge space, and therefore the search space, small. Remove all sentences which are subsumed (more special than) an existing sentence. If $P(x)$ is in the knowledge base, sentences as $P(A)$ or $P(A) \lor Q(B)$ can be removed.

- Well known efficient strategy: SLD-Resolution (linear resolution with selection function for definite clauses) (e.g. used in Prolog)
**SLD-Resolution**

- **linear**: Use a sequence of clauses \((C_0 \ldots C_n)\) starting with the negated assertion \(C_0\) and ending with the empty clause \(C_n\). Each \(C_i\) is generated as resolvent from \(C_{i-1}\) and a clause from the original set of axioms.

- **Selection function** (for the next literal which will be resolved) e.g. top-down-left-to-right in PROLOG; makes the strategy **incomplete**! (“user” must order clauses in a suitable way)

- **definite Horn clauses**: A Horn clause contains maximally one positive literal; a definite Horn clause contains exactly one positive literal (Prolog rule)
Prolog

<table>
<thead>
<tr>
<th>PROLOG</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact</td>
<td>isa(Fish,Animal)</td>
</tr>
<tr>
<td>isa(trout,Fish)</td>
<td>isa(Trout,Fish)</td>
</tr>
<tr>
<td>Rule</td>
<td>is(X,Y) :- isa(X,Y).</td>
</tr>
<tr>
<td>is(X,Z) :- isa(X,Y), is(Y,Z).</td>
<td>is(x,z) ∨ ¬isa(x,y) ∨ ¬is(y,z)</td>
</tr>
<tr>
<td>Query</td>
<td>is(trout,animal).</td>
</tr>
<tr>
<td></td>
<td>¬isa(Fish,x)</td>
</tr>
</tbody>
</table>

- denotes the “reversed” implication arrow.

\[ isa(x, y) \land is(y, z) \rightarrow is(x, z) \equiv \]
\[ ¬(isa(x, y) \land is(y, z)) \lor is(x, z) \equiv ¬isa(x, y) \lor ¬is(y, z) \lor is(x, z) \]

- Variables which occur in the head of a clause are implicitly universally quantified. Variables which occur only in the body are existentially quantified.

\[ ∀x∀z∃y \neg isa(x, y) \lor \neg is(y, z) \lor is(x, z) \]
Prolog Example

- Query: \( \text{is(fish,X)} \) (stands for \( \exists x \text{ is}(Fish, x) \))
- Negation of query: \( \neg \exists x \text{ is}(Fish, x) \equiv \forall x \neg \text{is}(Fish, x) \)
- SLD-Resolution:(extract)
Remarks on Prolog

- When writing Prolog programs, one should be known how the interpreter is working (i.e., understand SLD-resolution)

- Sequence of clauses has influence whether an assertion which follows logically from a set of clauses can be derived!

- **Efficiency**: Facts before rules

- **Termination**: non-recursive rule before recursive.

\[
\text{% Program}
\]

\[
\text{isa(trout,fish).}
\]

\[
\text{isa(fish,animal).}
\]

\[
\text{is(X,Z) :- is(X,Y), isa(Y,Z).}
\]

\[
\text{is(X,Y) :- isa(X,Y).}
\]

\[
\text{is(trout,Y),isa(Y,animal)}
\]

\[
\text{is(trout,Y'),isa(Y',animal),isa(Y,animal)}
\]

\[
\text{...}
\]

\[
\text{% Query}
\]

\[
\text{? is(trout,animal).}
\]

CogSysI Lecture 8: Automated Theorem Proving – p. 222
Applications of Resolution Calculus

- **PROLOG**
- as a *basic* method for theorem proving (others: e.g. tableaux)
- **Question Answering Systems**

- Yes/No-Questions: Assertion/Query *mortal*(s)
- Query *is*(trout, X) corresponds to “What is a trout?”
  The variable X is instantiated during resolution and the answer is “a fish”.
- *buys*(peter, john, X): “What does John buy from Peter?”
- *buys*(peter, X, car): “Who buys a car from Peter?”
Theorem Provers

- Theorem provers typically are more general than Prolog: not only Horn clauses but full FOL; no interleaving of logic and control (i.e. ordering of formulas has no effect on result)

- Examples: Boyer-Moore (1979) theorem prover; OTTER, Isabelle

- Theorem provers for mathematics, for verification of hardware and software, for deductive program synthesis.
Forward- and Backward Chaining

Rules (e.g. in Prolog) have the form: 
*Premises* → *Conclusion*

All rule-based systems (production systems, planners, inference systems) can be realized using either forward-chaining or backward-chaining algorithms.

Forward chaining: Add a new fact to the knowledge base and derive all consequences (data-driven)

Backward chaining: Start with a goal to be proved, find implication sentences that would allow to conclude the goal, attempt to prove the premises, etc.

Well known example for a backward reasoning expert system: MYCIN (diagnosis of bacterial infections)
The Running Gag of CogSysI

Question: How many AI people does it take to change a lightbulb?

Answer: At least 67.

4th part of the solution: The Logical Formalism Group (12)

- One to figure out how to describe lightbulb changing in predicate logic
- One to show the adequacy of predicate logic
- One to show the inadequacy of predicate logic
- One to show that lightbulb logic is nonmonotonic
- One to show that it isn’t nonmonotonic
- One to incorporate nonmonotonicity into predicate logic
- One to determine the bindings for the variables
- One to show the completeness of the solution
- One to show the consistency of the solution
- One to hack a theorem prover for lightbulb resolution
- One to indicate how it is a description of human lightbulb-changing behavior
- One to call the electrician (“Artificial Intelligence”, Rich & Knight)

CogSysI Lecture 8: Automated Theorem Proving – p. 226