CHAPTER 9.5-6, CHAPTERS 8.1 AND 10.2-3

Industrial-strength inference
Outline

Logic Programming

Resolution

Completeness
Does a complete algorithm exist?

Should be able to infer \( \text{Reach}(M,e) \), but FC/BC won't do it.

\[
\begin{align*}
\text{Reach} \models (x) \text{ Reach}(x) & \models (x) \text{ Reach}(x) \\
\text{Reach} \models (x) \text{ Reach}(x) & \models (x) \text{ Reach}(x) \\
\end{align*}
\]

E.g., from

but incomplete for general first-order logic

Forward and backward chaining are complete for Horn KBs.

\[
\begin{align*}
\neg a & \models \neg a \quad \text{Whenever KB} \\
\end{align*}
\]

Procedure is complete if and only if

Completeness in FOL
A brief history of reasoning

- Practical algorithm for FOL—resolution
- Complete algorithm for propositional logic
- Complete algorithm for arithmetic
- Complete algorithm for FOL (reduce to propositional)
- Proof by truth tables
- First-order logic
- Propositional logic (again)
- Probability theory (propositional logic + uncertainty)
- "Syllogisms" (inference rules), quantifiers
- Propositional logic, inference (maybe)

- Aristotle
- Stoics
- 450 B.C.

- Boole
- 1847

- Frege
- 1879

- Wittgenstein
- 1922

- Godel
- 1930

- Herbrand
- 1930

- Davis/Putnam
- 1960

- Robinson
- 1965
Inference continues until an empty clause is derived (contradiction)

Resolution

Resolution rule combines two clauses to make a new one:

Resolution uses \( \not\alpha \) in CNF (conjunctive normal form) to prove \( \not\alpha \) is unsatisfiable

Resolution is a refutation procedure:

success or failure, or may go on forever

Cf. Halting Problem: proof procedure may be about to terminate with

\( \not\alpha \) cannot always prove that \( \not\alpha \) is unsatisfiable

can find a proof of \( \alpha \) if \( \not\alpha \)

Entailment in first-order logic is only semi-decidable:

Resolution
\{ \forall x \in W / x \} = \varnothing \text{ with }

\frac{\text{Rich} \land (x \neg \text{happy} \land (x \neg \text{rich}))}{\text{Rich} \land \exists x (x \neg \text{happy} \land (x \neg \text{rich}))}

For example,

\varnothing \forall \varnothing \land \exists \varnothing = \varnothing \land \exists \varnothing

where

\varnothing \land \exists \varnothing \land \exists \varnothing \land \exists \varnothing \land \exists \varnothing \land \exists \varnothing

\text{Full first-order version:}

\varnothing \land \exists \varnothing

\text{Basic propositional version:}

\text{Resolution inference rule}
\[ \neg (R \land p) \lor (\exists \neg (R \land p)) \]

7. Distribute \( \lor \) over \( \land \), e.g., \( (R \land p) \lor (\exists \neg (R \land p)) \)

6. Drop universal quantifiers

5. Eliminate \( \exists \) by Skolemization (next slide)

\[ \forall x \neg \exists y \neg p \land \neg \neg p \]

4. Move quantifiers left in order, e.g., \( \forall x \exists y \neg p \land \neg \neg p \)

3. Standardize variables apart, e.g., \( \forall x \exists y \neg p \land \neg \neg p \)

2. Move \( \neg \) inwards, e.g., \( \neg \exists x p \lor \neg \neg p \)

1. Replace \( p \) by \( \exists x \neg p \)

Any FOL KB can be converted to CNF as follows:

The KB is a conjunction of clauses:

\((\neg (R \land p) \lor (\exists \neg (R \land p)) \lor (\forall x \neg \exists y \neg p \land \neg \neg p) \lor (\forall x \exists y \neg p \land \neg \neg p) \lor (\exists x \neg p \lor \neg \neg p))\)

Clause = disjunction of literals, e.g., \( \forall x \neg \exists y \neg (\text{Rich}(x) \land \neg \text{Unhappy}(y)) \)

Literal = (possibly negated) atomic sentence, e.g., \( \neg \text{Rich}(x) \land \neg \text{Unhappy}(y) \)
Skolem function arguments: all enclosing universally quantified variables.

\[
\begin{align*}
\text{Correct:} & \quad \exists x \phi(x) \Rightarrow (\exists x \phi(x))' \\
\text{Incorrect:} & \quad \exists x \phi(x) \Rightarrow (\exists x \phi(x))'
\end{align*}
\]

More tricky when \( x \in \text{inside A} \)

\[
\theta = (\exists x \frac{\theta x}{p}) \quad \text{becomes} \quad \exists x \phi(x) = (\exists x \frac{\phi x}{p})
\]

\[
\exists x \phi(x) \quad \text{becomes} \quad (\exists x \phi(x))'
\]

Ex Skolem Constant
\[
\neg \forall x \in \text{Family}\ (x) \land \neg \forall x \in \text{Friend}\ (x) \land \neg \forall x \in \text{Highly Qualified}\ (x) \\
\neg \forall x \in \text{Qualified}\ (x) \land \forall x \in \text{Highly Qualified}\ (x) \\
\neg \forall x \in \text{Qualified}\ (x) \land \forall x \in \text{Highly Qualified}\ (x)
\]

E.g., to prove \( \text{Rich}(\text{me}) \), add \( \neg \text{Rich}(\text{me}) \) to the CNF KB

- Inter contradiction
- Add to CNF KB
- Convert to CNF
- Negate it

To prove C.

Resolution Proof
Resolution Proof
Should be easier to debug \( \text{Capitol(New York, \cup S \cup I) then } x := x + 2 \) i

<table>
<thead>
<tr>
<th>Debug procedural errors</th>
<th>7. Find false facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply program to data</td>
<td>6. Ask queries</td>
</tr>
<tr>
<td>Encode problem instance as data</td>
<td>5. Encode problem instance as facts</td>
</tr>
<tr>
<td>Program solution</td>
<td>4. Encode information in KB</td>
</tr>
<tr>
<td>Figure out solution</td>
<td>3. Tea break</td>
</tr>
<tr>
<td>Assemble information</td>
<td>2. Assemble information</td>
</tr>
<tr>
<td>Identity problem</td>
<td>1. Identity problem</td>
</tr>
<tr>
<td>Ordinary programming</td>
<td>Logic Programming</td>
</tr>
</tbody>
</table>
e.g., not P(X) succeeds iff P(X) fails

Closed-world assumption (negation as failure)

Built-in predicates for arithmetic etc., e.g., X is Y + Z

Depth-first, left-to-right, backward chaining

Efficient retrieval of matching clauses by direct linking

Efficient unification by open coding

Program = set of clauses = head − \texttt{head:−}

Compilation techniques ⇒ 10 million LIPS

Widely used in Europe; Japan (basis of 5th generation project)

Basis: backward chaining with Horn clauses + bells & whistles

Prolog systems
\[ A = 1 \{ 2 \} \quad B = \{ 1 \} \quad C = \{ 2 \} \]

**answers:** \( A = \emptyset, B = \{ 1, 2 \} \)

**query:** append(A, B, [1, 2])

\[ (Z, y', x') \leftarrow \text{append}(X, y', Z) \]

Append two lists to produce a third:

No need to loop over \( S \): successor succeeds for each:

\[ dts(X) \leftarrow \text{successor}(X, S), dts(S) \]

\[ dts(X) \leftarrow \text{goal}(X) \]

**Examples**