Industrial-strength inference

Chapter 9.5–6, Chapters 8.1 and 10.2–3

Outline

◇ Completeness
◇ Resolution
◇ Logic programming
Completeness in FOL

Procedure \(i\) is complete if and only if

\[ KB \vdash_i \alpha \quad \text{whenever} \quad KB \models \alpha \]

Forward and backward chaining are complete for Horn KBs
but incomplete for general first-order logic

E.g., from

\[ PhD(x) \Rightarrow HighlyQualified(x) \]
\[ \neg PhD(x) \Rightarrow EarlyEarnings(x) \]
\[ HighlyQualified(x) \Rightarrow Rich(x) \]
\[ EarlyEarnings(x) \Rightarrow Rich(x) \]

should be able to infer \( Rich(Me) \), but FC/BC won’t do it

Does a complete algorithm exist?

A brief history of reasoning

450 B.C. Stoics propositional logic, inference (maybe)
322 B.C. Aristotle "syllogisms" (inference rules), quantifiers
1565 Cardano probability theory (propositional logic + uncertainty)
1847 Boole propositional logic (again)
1879 Frege first-order logic
1922 Wittgenstein proof by truth tables
1930 Gödel \( \exists \) complete algorithm for FOL
1930 Herbrand complete algorithm for FOL (reduce to propositional)
1931 Gödel \( \neg \exists \) complete algorithm for arithmetic
1960 Davis/Putnam "practical" algorithm for propositional logic
1965 Robinson "practical" algorithm for FOL—resolution
Resolution

Entailment in first-order logic is only semidecidable:
can find a proof of $\alpha$ if $KB \vdash \alpha$
cannot always prove that $KB \not\vdash \alpha$
Cf. Halting Problem: proof procedure may be about to terminate with success or failure, or may go on for ever

Resolution is a refutation procedure:

to prove $KB \vdash \alpha$, show that $KB \land \neg \alpha$ is unsatisfiable

Resolution uses $KB, \neg \alpha$ in CNF (conjunction of clauses)

Resolution inference rule combines two clauses to make a new one:

$$\frac{C_1 \lor \neg \beta, \neg \beta \lor \gamma}{C \lor \gamma} \quad \text{or equivalently} \quad \frac{\neg \alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\neg \alpha \Rightarrow \gamma}$$

Inference continues until an empty clause is derived (contradiction)

Resolution inference rule

Basic propositional version:

$$\frac{\alpha \lor \beta, \neg \beta \lor \gamma}{\alpha \lor \gamma} \quad \text{or equivalently} \quad \frac{\neg \alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\neg \alpha \Rightarrow \gamma}$$

Full first-order version:

$$\frac{p_1 \lor \ldots \ p_j \ldots \lor p_m, \quad q_1 \lor \ldots \ q_k \ldots \lor q_n}{(p_1 \lor \ldots \ p_{j-1} \lor p_{j+1} \ldots \lor p_m \lor q_1 \ldots \ q_{k-1} \lor q_{k+1} \ldots \lor q_n)\sigma}$$

where $p_j \sigma = \neg q_k \sigma$

For example,

$$\frac{\neg Rich(x) \lor Unhappy(x)}{Rich(Me) \quad Unhappy(Me)}$$

with $\sigma = \{ x/Me \}$
Conjunctive Normal Form

Literal = (possibly negated) atomic sentence, e.g., \( \neg \text{Rich(Me)} \)

Clause = disjunction of literals, e.g., \( \neg \text{Rich(Me)} \lor \text{Unhappy(Me)} \)

The KB is a conjunction of clauses

Any FOL KB can be converted to CNF as follows:
1. Replace \( P \rightarrow Q \) by \( \neg P \lor Q \)
2. Move \( \neg \) inwards, e.g., \( \neg \forall x \ P \) becomes \( \exists x \ \neg P \)
3. Standardize variables apart, e.g., \( \forall x \ P \lor \exists x \ Q \) becomes \( \forall x \ P \lor \exists y \ Q \)
4. Move quantifiers left in order, e.g., \( \forall x \ P \lor \exists x \ Q \) becomes \( \forall x \exists y \ P \lor Q \)
5. Eliminate \( \exists \) by Skolemization (next slide)
6. Drop universal quantifiers
7. Distribute \( \land \) over \( \lor \), e.g., \( (P \land Q) \lor R \) becomes \( (P \lor Q) \land (P \lor R) \)

Skolemization

\( \exists x \ \text{Rich}(x) \) becomes \( \text{Rich}(G1) \) where \( G1 \) is a new "Skolem constant"

\( \exists k \ \frac{d}{dy}(k^y) = k^y \) becomes \( \frac{d}{dy}(e^y) = e^y \)

More tricky when \( \exists \) is inside \( \forall \)

E.g., “Everyone has a heart”
\[ \forall x \ \text{Person}(x) \Rightarrow \exists y \ \text{Heart}(y) \land \text{Has}(x,y) \]

Incorrect:
\[ \forall x \ \text{Person}(x) \Rightarrow \text{Heart}(H1) \land \text{Has}(x,H1) \]

Correct:
\[ \forall x \ \text{Person}(x) \Rightarrow \text{Heart}(H(x)) \land \text{Has}(x,H(x)) \]

where \( H \) is a new symbol (“Skolem function”)

Skolem function arguments: all enclosing universally quantified variables
To prove $\alpha$:
- negate it
- convert to CNF
- add to CNF KB
- infer contradiction

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

$\neg PhD(x) \lor HighlyQualified(x)$
$PhD(x) \lor EarlyEarnings(x)$
$\neg HighlyQualified(x) \lor Rich(x)$
$\neg EarlyEarnings(x) \lor Rich(x)$
Sound bite: computation as inference on logical KBs

<table>
<thead>
<tr>
<th>Logic programming</th>
<th>Ordinary programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify problem</td>
<td>Identify problem</td>
</tr>
<tr>
<td>2. Assemble information</td>
<td>Assemble information</td>
</tr>
<tr>
<td>3. Tea break</td>
<td>Figure out solution</td>
</tr>
<tr>
<td>4. Encode information in KB</td>
<td>Program solution</td>
</tr>
<tr>
<td>5. Encode problem instance as facts</td>
<td>Encode problem instance as data</td>
</tr>
<tr>
<td>6. Ask queries</td>
<td>Apply program to data</td>
</tr>
<tr>
<td>7. Find false facts</td>
<td>Debug procedural errors</td>
</tr>
</tbody>
</table>

Should be easier to debug $\text{Capital}(\text{NewYork}, \text{US})$ than $x := x + 2$!

---

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques $\Rightarrow$ 10 million LIPS

Program = set of clauses = head :- literal$_1$, ... literal$_n$.
Efficient unification by open coding
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., $X$ is $Y*Z+3$
Closed-world assumption ("negation as failure")
  e.g., not PhD($X$) succeeds if PhD($X$) fails
Prolog examples

Depth-first search from a start state X:

dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).

No need to loop over S: successor succeeds for each

Appending two lists to produce a third:

append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).

query: append(A,B,[1,2]) ?
answers: A=[]  B=[1,2]
A=[1,2]  B=[]