Industrial-strength inference

Chapter 9.5–6, Chapters 8.1 and 10.2–3

Completeness in FOL

Procedure \( \alpha \) is complete if and only if
\[ KB \vdash \alpha \quad \text{whenever} \quad KB \models \alpha \]

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

E.g., from
\[ PhD(x) \Rightarrow HighQualified(x) \]
\[ \neg PhD(x) \Rightarrow EarlyEarnings(x) \]
\[ HighQualified(x) \Rightarrow Rich(x) \]
\[ EarlyEarnings(x) \Rightarrow Rich(x) \]

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

Does a complete algorithm exist?

Resolution

Entailment in first-order logic is only semidecidable:
can find a proof of \( \alpha \) if \( KB \models \alpha \)
cannot always prove that \( KB \models \alpha \)

Cl. 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 \( KB \models \alpha \), show that \( KB \wedge \neg \alpha \) is unsatisfiable

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

Resolution inference rule combines two clauses to make a new one:
\[
\begin{array}{c}
C_1 \\
C_2
\end{array}
\]

Inferece continues until an empty clause is derived (contradiction)

Resolution inference rule

Basic propositional version:
\[
\frac{\alpha \lor \beta, \neg \beta \lor \gamma \quad \text{or equivalently} \quad \neg \alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\alpha \lor \gamma}
\]

Full first-order version:
\[
\frac{p_1 \lor \ldots \lor p_j \lor \ldots \lor p_m, q_1 \lor \ldots \lor q_j \lor \ldots \lor q_n, \ldots \lor q_k \lor \ldots \lor q_m \lor q_{j+1} \ldots \lor q_n \lor \sigma}{\neg p_j \sigma}
\]
where \( p_j \sigma = \neg q_i \sigma \)

For example,
\[
\frac{\neg \text{Rich}(x) \lor \text{Unhappy}(x)}{	ext{Rich}(\text{Me}) \lor \text{Unhappy}(\text{Me})}
\]

with \( \sigma = \{x/\text{Me}\} \)

Outline

◊ Completeness
◊ Resolution
◊ Logic programming

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
\( \exists \) complete algorithm for arithmetic

1960
Davis/Putnam
"practical" algorithm for propositional logic

1965
Robinson
"practical" algorithm for FOL—resolution
Conjunctive Normal Form

Literal = (possibly negated) atomic sentence, e.g., ¬Rich(Me)

Clause = disjunction of literals, e.g., ¬Rich(Me) ∨ Unhappy(Me)

The KB is a conjunction of clauses

Any FOL KB can be converted to CNF as follows:
1. Replace P ⇒ Q by ¬P ∨ Q
2. Move ¬ inwards, e.g., ¬∀x P becomes ∃x ¬P
3. Standardize variables apart, e.g., ∀x P ∨ ∃y Q becomes ∀x P ∨ ∃y Q
4. Move quantifiers left in order, e.g., ∃x P ∨ ∃y Q becomes ∃x ∃y P ∨ Q
5. Eliminate ∀ by Skolemization (next slide)
6. Drop universal quantifiers
7. Distribute ∧ over ∨, e.g., (P ∧ Q) ∨ R becomes (P ∨ Q) ∧ (P ∨ R)

Resolution proof

To prove α:
- negate it
- convert to CNF
- add to CNF KB
- infer contradiction

E.g., to prove Rich(me), add ¬Rich(me) to the CNF KB
¬PhD(x) ∨ HighlyQualified(x)
¬PhD(x) ∨ EarlyEarnings(x)
¬HighlyQualified(x) ∨ Rich(x)
¬EarlyEarnings(x) ∨ Rich(x)

Logic programming

Sound bite: computation as inference on logical KBs

Logic programming
1. Identify problem
2. Assemble information
3. Tea break
4. Encode information in KB
5. Encode problem instance as facts
6. Ask queries
7. Find false facts
Should be easier to debug Capital(New York, US) than x := x + 2!

Skolemization

∀x Rich(x) becomes Rich(G1) where G1 is a new “Skolem constant”
∃k k^2 = 9 becomes ∃k k^2 = 9

More tricky when ∃ is inside ∀

E.g., “Everyone has a heart”
∀x Person(x) ⇒ ∃y Heart(y) ∧ Has(x, y)

Incorrect:
∀x Person(x) ⇒ Heart(H) ∧ Has(x, H)

Correct:
∀x Person(x) ⇒ Heart(H[x]) ∧ Has(x, H[x])

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

Skolem function arguments: all enclosing universally quantified variables

Resolution proof

¬PhD(x) ∨ H(x)
¬H(x) ∨ Rich(x)

PhD(x) ∨ ES(x)

Rich(x) ∨ ES(x)
¬ES(x) ∨ Rich(x)

(α/ϕ)

Prolog systems

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

Program = set of clauses = head : ¬ literal, ... literal, .
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*X+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 \( X \):

\[
dfs(X) :- \text{goal}(X).
dfs(X) :- \text{successor}(X,S), dfs(S).\]

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

Appending two lists to produce a third:

\[
\text{append}([], Y, Y).
\text{append}([X|L], Y, [X|Z]) :- \text{append}(L, Y, Z).
\]

query: \text{append}(A, B, \{1,2\}) ?

answers: \begin{align*}
A &= [] & B &= \{1,2\} \\
A &= \{1\} & B &= \{2\} \\
A &= \{1,2\} & B &= []
\end{align*}