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 \( \text{Rich}(w) \), but FC/BC won't do it

\[
\begin{align*}
(\text{EarlyException}(x) & \iff \text{Rich}(x)) \\
(\text{HighlyQualified}(x) & \iff \text{Rich}(x)) \\
(\text{EarlyException}(x) & \iff \text{PhD}(x)) \\
(\text{HighlyQualified}(x) & \iff \text{PhD}(x))
\end{align*}
\]

E.g., from

but incomplete for general first-order logic

Forward and backward chaining are complete for Horn KBs

\[ KB \vdash A \text{ whenever } KB \models A \]

procedure is complete if and only if

Completeness in FOL
A brief history of reasoning

practical algorithm for FOL—resolution
practical algorithm for propositional logic
complete algorithm for arithmetic
complete algorithm for FOL (reduce to propositional)

Robinson 1965

Davis/Putnam 1950

Godel 1931

Herbrand 1930

Godel 1930

Wittgenstein 1922

Frege 1879

Boole 1847

Cardano 1555

322B.C. Aristote

450B.C. Stoics
Inference continues until an empty clause is derived (contradiction).

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

Resolution uses \( A \lor B \), \( A \) in CNF (conjunction of clauses).

Resolution is a refutation procedure:

- To prove \( \forall x. Y \models B \lor \neg x \) show that \( Y \models \neg \exists x. B \).

Resolution, Halting Problem: proof procedure may be about to terminate with success or failure, or may go on forever.

Entailment in first-order logic is only semidecidable.
\[ \{ w / x \} = \emptyset \]

\[ \bigcup \neg \text{happy} \]

\[ \text{Rich} \]

\[ \neg \text{Rich} \]

For example,

\[ \forall y \, \forall x \]

\[ \text{where} \]

\[ \forall (u y \land \cdots \land f y \land \cdots \land d \land \cdots \land f d) \]

\[ \forall y \land \forall (u y \land \cdots \land f y \land \cdots \land d) \]

\[ \forall (u y \land \cdots \land f y \land \cdots \land d) \land \forall (u y \land \cdots \land f y \land \cdots \land d) \]

Full First-Order Version:

or Equivalence

Basic Propositional Version:

Resolution Inference Rule
\((R \land p) \lor (\neg R \land p)\) becomes \(R \land (\neg R \lor p)\)

1. Distribute \(\lor\) over \(\land\), e.g.

2. Drop universal quantifiers (next slide)

3. Standardize variables apart, e.g.

4. Move quantifiers left in order, e.g.

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

6. Drop universal quantifiers

Any FOL KB can be converted to CNF as follows:

The KB is a conjunction of clauses

\((\neg\text{unhappy}(e)) \land (\neg\text{Rich}(e))\) = \text{disjunction of literals, e.g.} \(\neg\text{Rich}(e) \lor \neg\text{unhappy}(e)\).

\((\text{Rich}(e)) \land (\text{unhappy}(e))\) = \text{possibly negated (atomic) sentence, e.g.} \(\text{Rich}(e)\).
Skolem function arguments: all ending universally quantified variables

\[ ((x)H, x) \text{has} H \lor ((x)H) \text{heart} \Leftrightarrow (x)\text{Person} \land x \land \text{Correct:} \]

\[ (\exists x)H, x) \text{has} H \lor (\exists x)\text{heart} \Leftrightarrow (x)\text{Person} \land x \land \text{Incorrect:} \]

E.g., “Everyone has a heart”

More tricky when \( \exists \) is inside

\[ \exists x \theta = (\exists x)\text{Person} \quad \exists y \gamma = (\exists y)\text{Person} \]

\( \exists x \text{Rich}(x) \land (\exists x)\text{Rich}(x) \text{ becomes Rich constant} \)

Skolemization
\[ \neg \exists x \text{rich}\(\text{x}\) \land \neg \exists x \text{highly\#qualifying}\(\text{x}\) \land \neg \exists x \text{PhD}\(\text{x}\) \land \neg \exists x \text{early\#qualifying}\(\text{x}\) \land \neg \exists x \text{rich\(\text{me}\)} \]
Resolution Proof
$i \Rightarrow x = x + 2$

Should be easier to debug Capital(NeW YOrK, \cup S) than $x = x + 2$

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

Debug procedural errors
Apply program to data
Program solution
Figure out solution
Assemble information
Ordinary programming

Sound bite: computation as inference on logical KBs

Logic Programming
Chapter 9.5-9.8, Chapters 31 and 10.2-3

**Prolog Systems**

E.g., not $\text{pred}(X)$ succeeds if $\text{pred}(X)$ fails

Closed-world assumption (negation as failure)

Built-in predicates for arithmetic etc., e.g., $x \in \mathbb{Z} + 3$

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 :: tail :: ... tail.

Compilation techniques ⇒ 10 million LIPS

Widely used in Europe, Japan (basis of 6th Generation Project)

Basis: backward chaining with Horn clauses + bells and whistles
\[ A = [\, 1, \, 2, \, ] \quad B = [\, 2, \, ] \]
\[ A = [\, 1, \, ] \quad B = [\, ] \]
\[ answers : A = [\, 1, \, 2, \, ] \quad B = [\, ] \]

\[ \text{query: append}(A, B) \]

\[ \cdot (Z | X | Y) \quad \cdot (Z | X | Y) \quad \cdot (Z | X | Y) \]

\[ \text{append}(\text{append}(\text{append}(\text{append}([\, ], [\, ]), [\, ]), [\, ]), [\, ]), [\, ]) \]

Appending two lists to produce a third:

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

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

Depth-first search from a start state \( X \):

Prolog examples