COMP4418: Knowledge Representation and Reasoning

Logic and Prolog

Maurice Pagnucco
School of Computer Science and Engineering
University of New South Wales
NSW 2052, AUSTRALIA
morri@cse.unsw.edu.au
Logic and Prolog

■ Prolog stands for programming in logic
■ How does the implementation of Prolog relate to logic?
■ Prolog is based on resolution theorem proving in first-order logic
■ In this lecture we will look at the relationship between automated reasoning in first-order logic and Prolog

■ References:
  ▶ Ivan Bratko, Prolog Programming for Artificial Intelligence, Addison-Wesley, 2001. (Chapter 2)
Overview

- Problems
- Undecidability of first-order logic
- Horn Clauses
- SLD Resolution
- Prolog
- Back Chaining
- Forward Chaining
- Negation as Failure
- Conclusion
Resolution — Problem 1

■ We have seen that the resolution rule is sound:
  If $\Gamma \vdash \phi$, then $\Gamma \models \phi$

■ However, the resolution rule is not complete in general:
  $\{\neg P\} \models \neg P \lor \neg Q$ but cannot show this using resolution ($\{\neg P\} \vdash \neg P \lor \neg Q$)

■ Resolution is sound and complete when used as a refutation system though:
  $\Gamma \vdash \Box$ if and only if $\Gamma \models \Box$

■ Therefore, resolution should be used as a refutation system as we have done so far
Resolution — Problem 2

- $KB = \{ P(f(x) \rightarrow P(x)) \}$
- $Q = P(a)$?
- Obviously $KB \not\models Q$
- However, let us attempt to show this using resolution

```
\sim P(f(x)) \lor P(x)
\sim P(a)
\sim P(f(a))
\sim P(f(f(a)))
\sim P(f(f(f(a))))
```
Undecidability of First-Order Logic

■ Can we determine in general when this problem will arise?

■ **Answer:** no!

■ There is no general procedure

  ```
  if (KB unsatisfiable)
  return Yes; Halt
  else return No; Halt
  ```

■ Resolution is refutation complete so if KB is unsatisfiable search tree will contain empty clause somewhere

■ Can find empty clause using breadth-first search (why?) but if the search tree does not contain the empty clause the search may go on forever

■ Even in the propositional case (which is decidable), complexity of resolution is $O(2^n)$
Horn Clauses

Idea: use less expressive language

■ Review
  ▶ Literals — atomic sentence or its negation
  ▶ Clause — disjunction of literals

■ Horn Clause – at most one positive literal (e.g., \( \neg P \lor Q \), \( P \lor \neg Q \lor R \lor S \))
  ▶ Essentially represents a formula of the form \( A_1 \land \ldots \land A_n \rightarrow C \)
  ▶ That is, if \( A_1 \) and \( \ldots \) and \( A_n \), then \( C \)

■ Definite (Positive) Clause – exactly one positive literal

■ Negative Clause – no positive literals
SLD Resolution — \( \vdash_{SLD} \)

- Selected literals
- Linear form
- Definite clauses resolution

- SLD derivation of a clause \( C \) from a set of clauses \( KB \) is a sequence of clauses such that
  1. First clause of sequence comes from \( KB \)
  2. Each intermediate clause \( C_i \) is derived by resolving the previous clause \( C_{i-1} \) and a clause from \( KB \)
  3. The last clause in the sequence is \( C \)

- For set of Horn clauses \( KB \): \( KB \vdash \Box \) if and only if \( KB \vdash_{SLD} \Box \)
Prolog

- Horn clauses in first-order logic (facts and rules)
- SLD resolution
- Depth-first search strategy with backtracking
- User control
  - Ordering of predicates in Prolog database (facts and rules)
  - Ordering of subgoals in body of a rule
  - Cut (!) operator
  - Negation as failure

That is, Prolog is a restricted form of first-order logic (Horn clauses) and puts more control of the theorem proving process into the hands of the programmer allowing them to use problem-specific knowledge to reduce search
Backward Chaining

(Brachman & Levesque) Show whether Horn knowledge base satisfiable

- Goal driven

- Start with hypothesis and work backwards using rules in knowledge base to easily confirmed findings

- Check satisfiability of set of Horn clauses:

  \[
  \text{prove}(Q_1 \land \ldots \land Q_n) \{
  \begin{align*}
  & \text{if } n = 0 \text{ return yes } \quad % \text{ empty clause} \\
  & \text{for each } R \in KB \text{ do} \\
  & \quad \text{if } R = Q_1 \leftarrow G_1 \land \ldots \land G_m \text{ and prove}(G_1 \land \ldots \land G_m \land \\
  & \qquad Q_2 \land \ldots \land Q_n) \\
  & \quad \quad \text{then return yes} \\
  & \quad \text{return no}
  \end{align*}
  \}
  \]

- Depth-first, left-right, backward chaining

- Strategy applied by Prolog
Forward Chaining

(Brachman & Levesque) Determine whether Horn knowledge base entails query: $KB \models Q$

- Data driven

- 1. if $Q$ marked solved then return yes
- 2. if $G \leftarrow G_1 \land \ldots \land G_m \in KB$ and $G_1, \ldots, G_m$ marked solved and $G$ not marked solved then mark $G$ solved; goto 1 else return no
Negation as Failure

- Prolog does not implement classical negation
- Prolog not is known as negation as failure
- `not(G) :- G, !, fail. % If G succeeds return no not(G). % else return yes`
- `KB ⊢ not(G) — cannot prove G`
- `KB ⊢ ¬G — can prove ¬G`
- They are not the same
- Negation as failure is finite failure
Conclusion

- First-order logic is an expressive formal language and allows for powerful reasoning
- Theorem proving is undecidable in general
- Other options:
  - Search heuristics (ordering of predicates, subgoals; depth-first search)
  - Sacrifice expressivity (e.g., Horn clauses although still undecidable in first-order case)
  - User control (cut operator)
- Prolog is based on SLD resolution in first-order Horn logic and allows programmer to use knowledge about domain to control search
- Blend of theory and pragmatics