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:

Soundness and Completeness Again

- First-order resolution refutation is **sound**, i.e. it preserves truth (if a set of premises are all true, any conclusion drawn from those premises must also be true)
- First-order resolution refutation is **complete**, i.e. it is capable of proving all consequences of any knowledge base (not shown here!)
- First-order resolution refutation is **not decidable**, i.e. there is no algorithm implementing resolution which when asked whether $S \vdash P$, can always answer ‘yes’ or ‘no’ (correctly)
Undecidability of First-Order Logic

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

Horn Clauses

**Idea:** use less expressive language

- **Review**
  - literal — atomic formula or negation of atomic formula
  - clause — disjunction of literals
- **Definite Clause** — exactly one positive literal
  - e.g. $C \leftarrow A_1 \land \ldots \land A_n$, i.e. $C \leftarrow A_1 \land \ldots \land A_n$ (Prolog rule)
- **Negative Clause** — no positive literals
  - e.g. $\neg Q$ (negation of a query)
- **Horn Clause** — clause with at most one positive literal

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;
  - else return No;
- Resolution refutation is complete so if KB is unsatisfiable, the search tree will contain the empty clause somewhere
- ... 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 $O(2^n)$ for problems of size $n$

SLD Resolution — $\vdash_{SLD}$

- Selected literals Linear form Definite clauses resolution
- SLD refutation of a clause $C$ from a set of clauses $KB$ is a sequence of clauses such that
  1. First clause of sequence is $C$
  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 $\Box$

- **Theorem.** For a definite $KB$ and negative clause query $Q$: $KB \cup Q \vdash_{SLD} \Box$
  if and only if $KB \cup Q \vdash_{SLD} \Box$
Prolog

- Horn clauses in first-order logic (facts and rules)
- SLD resolution
- Depth-first search strategy with backtracking
- User control
  - Ordering of clauses 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.

Negation as Failure

- Prolog does not implement classical negation
- Prolog not is known as negation as failure
- \( \text{not}(G) :- G, !, \text{fail}. \) % If G succeeds return no not(G). % else return yes
- \( KB \vdash \text{not}(G) \) — cannot prove \( G \)
- \( KB \vdash \neg G \) — can prove \( \neg G \)
- They are not the same
- Negation as failure is finite failure

Abstract Prolog Interpreter

Input: A query \( Q \) and a logic program \( KB \)
Output: ‘yes’ if \( Q \) follows from \( KB \), ‘no’ otherwise
- Initialise current goal set to \( \{ Q \} \);
- while the current goal set is not empty do
  - Choose \( G \) from the current goal set; (first in goal set)
  - Choose a copy \( G^\prime :- B_1, \ldots, B_n \) of a rule from \( KB \) for which most general unifier of \( G, G^\prime \) is \( \theta \); (try all in KB)
  - (if no such rule, undo unifications and try alternative rules)
  - Apply \( \theta \) to the current goal set;
  - Replace \( G\theta \) by \( B_1\theta, \ldots, B_n\theta \) in current goal set;
    if current goal set is empty,
      output yes;
    else output no;
- Depth-first, left-right with backtracking

Soundness and Completeness Again

- Prolog including cut and negation as failure is not sound, i.e. it does not preserve truth
- Pure Prolog (without cut and negation as failure) is not complete, i.e. it is incapable of proving all consequences of any knowledge base (this is because of the search order)
- Even pure Prolog is not decidable, i.e. the Prolog implementation of resolution when asked whether \( KB \vdash Q \), can not always answer ‘yes’ or ‘no’ (correctly)
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, breadth-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 clause logic and allows programmer to use knowledge about domain to control search
- Blend of theory and pragmatics