Deductive Logic

- e.g. of use: Gypsy specifications and proofs
- About deductive logic…
  - (Gödel, 1931) Interesting systems (with a finite number of axioms) are necessarily either:
    - incomplete (there are statements that can’t be proven)
    - or inconsistent (∃S such that S and ¬S can be proven true)
- Interesting systems include Presberger Arithmetic (0,1,*,+) and Peano Arithmetic (0,1,+)
- Recall: all inconsistent systems are complete
First Order Predicate Logic

• Logic programming is based on FOPL
• FOPL is complete (J.A. Robinson & resolution theorem proving)
  – "All clauses logically implied by an initial formula may be derived from
    the initial formula by the proof method."

BUT
• FOPL is undecidable
  – An attempt to prove a formula may go on forever, but there will be no
    indication when to stop without sacrificing formulae that can be proven.

⇒ completeness of FOPL is of theoretical interest, but of limited
  practicality. (completeness is predicated on there being a search strategy
  that knows when to stop a particular unproductive line of deduction.)

• Higher order predicate logics (and calculi) - ones which allow
  predicates of predicates - are not complete.

Foundations of Logic Programming

• Logic programming is based on Horn Clauses
  – In the propositional calculus all formulae can be put in conjunctive
    normal form (disjuncts connected by ∧)
  – Each disjunct can be expressed as:

    \[ \text{A}_1 \lor \text{A}_2 \lor \ldots \lor \text{A}_m \lor \neg \text{B}_1 \lor \neg \text{B}_2 \lor \ldots \lor \neg \text{B}_n \]

    \[ \Rightarrow \text{A}_1 \lor \text{A}_2 \lor \ldots \lor \text{A}_m \lor \neg (\text{B}_1 \land \text{B}_2 \land \ldots \land \text{B}_n) \]

    \[ \Rightarrow \text{A}_1 \lor \text{A}_2 \lor \ldots \lor \text{A}_m \Leftarrow (\text{B}_1 \land \text{B}_2 \land \ldots \land \text{B}_n) \]

• interpretations:
  \[ m > 1 \quad \text{Conclusions are indefinite, one or more are true.} \]
  \[ m = 1 \quad \text{Horn clauses.} \]
  \[ m = 1, n > 0 \quad (A \Leftarrow B_1 \land B_2 \land \ldots \land B_n) \quad \text{-- definite clause, 1 conclusion} \]
  \[ m = 1, n = 0 \quad (A \Leftarrow \text{-- unconditional definite clause (fact)} \]
  \[ m = 0, n > 0 \quad \text{negation of (B}_1 \land B_2 \land \ldots \land B_n) \]
  \[ m = 0, n = 0 \quad \Leftarrow \text{-- the empty clause (contradiction)} \]

• In logic, all clauses can be represented as Horn Clauses...
Proof by Refutation

• An important proof method:
  - \( P \): set of axioms
  - \( Q \): clause to be proven
    - show \( P \land \neg Q \) is false by deriving a contradiction
    - i.e., assert \( \leftarrow Q \) and try to derive empty clause, which represents false.
    - In this context, \( Q \) is called a goal.

• Propositional Horn Clause Resolution (PHC Resolution)
  - In doing a refutation proof, the following general PHC resolution step can be performed:
    \[
    A_1 \Leftarrow (B_1 \land B_2 \land \ldots \land B_n) \\
    \Leftarrow A_1 \land A_2 \land \ldots \land A_m \\
    \Leftarrow (B_1 \land B_2 \land \ldots \land B_n) \land A_2 \land \ldots \land A_m \\
    \Rightarrow \text{Keep applying this until} \Leftarrow \text{is achieved.}
    \]

More PHC Resolution

• e.g. to prove \( A_2 \)
  
  (1) \( A_1 \Leftarrow \) 
  (2) \( A_2 \Leftarrow A_1, A_3 \) 
  (3) \( A_3 \Leftarrow \) 
  (4) \( \Leftarrow A_2 \) -- negated goal

• proof leading to contradiction:
  
  (5) \( \Leftarrow A_1, A_3 \) -- apply 2 & 4 
  (6) \( \Leftarrow A_3 \) -- apply 1 & 5 
  (7) \( \Leftarrow \) -- apply 3 & 6

• Note: Prolog and other logic-based languages are based on this resolution proof strategy.
First Order Predicate Logic

- Predicates can have arguments: constants, variables, other functional terms.
  
  e.g.  
  
  (1)  a(X) ⇐ m(X)  
  (2)  m(X) ⇐ e(X)  
  (3)  e(c) ⇐  
  (4)  a(X) ⇐ s(X)  
  (5)  s(b) ⇐  
  (6)  ⇐ a(X)  

- When we start dealing with variables, we need:

  **Axiom of General Specification:** A clause with logical variables is true for every set of values of the variables.
  
  - Supports generalizing PHC resolution into **Horn Clause Resolution (HCR)**
    
    - by systematically instantiating variables. ⇐ "Unification"

---

FOPL (cont)

- e.g.

  1)  p(t)  
  2)  q(X) ⇐ p(X)  
  3)  ⇐ q(t)  
  4)  q(t) ⇐ p(t) (X = t)  -- from (2), (3) and substitution  
  5)  ⇐ p(t)  -- from (3) & (4)  
  6)  ⇐  -- from (1) and (5)

  ⊢ resolution is combination of unification and elimination in one operation.
More Proofs

- Using:
  1. \( a(X) \Leftarrow m(X) \)
  2. \( m(X) \Leftarrow e(X) \)
  3. \( e(c) \Leftarrow \)
  4. \( a(X) \Leftarrow s(X) \)
  5. \( s(b) \Leftarrow \)
  6. \( \Leftarrow a(X) \)

- with goal \( \Leftarrow a(X) \) (step (6)), we can derive:

  7. \( \Leftarrow m(X) \) -- applying (1) & (6)
  8. \( \Leftarrow e(X) \) -- applying (2) & (7)
  9. \( \Leftarrow X = c \) -- applying (3) & (8) also:
  10. \( \Leftarrow s(X) \) -- applying (4) & (6)
  11. \( \Leftarrow X = b \) -- applying (5) & (10)

Alternative Proof Strategies

- **Top Down**: what we've just seen - collecting variable bindings.
  - Start with goal and reduce into subgoals until there is only the empty subgoal.

- **Bottom up**: Combining facts with rules or rules with other rules.
Bottom Up

- Using:
  1. \( a(X) \Leftarrow m(X) \)
  2. \( m(X) \Leftarrow e(X) \)
  3. \( e(c) \Leftarrow \)
  4. \( a(X) \Leftarrow s(X) \)
  5. \( s(b) \Leftarrow \)
  6. \( \Leftarrow a(X) \)

- Combine rule (2) \( m(X) \Leftarrow e(X) \) -- combining
  with fact (3) \( e(c) \Leftarrow \) -- rule with
  yielding: \( m(c) \Leftarrow \) -- a fact yields
  combined with rule (1) \( a(X) \Leftarrow m(X) \) -- a new
  yields: \( a(c) \Leftarrow \) -- fact

- or
  Combine rule (1) \( a(X) \Leftarrow m(X) \) -- combining rules
  with rule (2) \( m(X) \Leftarrow e(X) \) -- to make a new
  yields: \( a(X) \Leftarrow e(X) \) -- rule

- -- allows us to make discoveries from known facts and rules.

Closed World Assumption

- Inability to demonstrate that something is true means that it is false.
  - assumes user made no typos and specified all things that need to be specified to properly identify true queries as true.
  - leads to joining "unknown" and "not provably true" into one class.
  - failing to prove something true leads to conclusion that it is false.

- CWA says that all things that are true have been specified as such or can be derived.
Closed World Assumption (2)

- Possible alternatives:
  1. leave system alone; accept CWA
  2. allow negation in clauses but not in conclusion of Horn Clauses
  3. allow statement of negative conclusions: search positive; search negative; report unknown;
  4. work in constrained environment where everything is known
  5. work in statistical environment where answers are expressed in terms of likelihoods.

About Prolog

- Prolog lends itself nicely to concurrency

```prolog
form: p0 :- p1, p2, p3, p4
      \-----\-----\-----\-----
      \ can be executed
concurrently(with communications about bindings) -- "AND parallelism"
```

or:

```prolog
HG :- ...................... { "OR
...     \ parallelism"
HG :- ...................... {```

Copyright 1999 Paul F. Reynolds, Jr.
About Prolog (2)

- Prolog and principles:
  - Orthogonal - separates *logic* and *control* (assert, retract and cut violate this)
  - regular - regular rules
  - security - meaning of a program is determined by what a user writes
  - simplicity - simple rules

- violates:
  - localized cost - execution cost is determined by rule order
  - defense in depth - misspellings alter meaning of program