Foundations of Logic Programming

 Paul F. Reynolds, Jr.

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 ($\exists$S such that S and $\neg$S can be proven true)
- Interesting systems include Presberger Arithmetic (0,1,*,+) and Peano Arithmetic (0,1,+)
- Recall: all inconsistent systems are complete

 Paul F. Reynolds, Jr.

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.

 Paul F. Reynolds, Jr.

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 $\wedge$)
  - Each disjunct can be expressed as:

$$A_1 \vee A_2 \vee \ldots \vee A_m \vee \neg B_1 \vee \neg B_2 \vee \ldots \vee \neg B_n$$
$$\rightarrow A_1 \vee A_2 \vee \ldots \vee A_m \vee \neg (B_1 \wedge B_2 \wedge \ldots \wedge B_n)$$
$$\rightarrow A_1 \vee A_2 \vee \ldots \vee A_m \Leftarrow (B_1 \wedge B_2 \wedge \ldots \wedge B_n)$$

- interpretations:

m > 1	Conclusions are indefinite, one or more are true.
m = 1	Horn clauses.
m = 1, n > 0	$(A \Leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_n)$ -- definite clause, 1 conclusion
m = 1, n = 0	$(A \Leftarrow)$ unconditional definite clause (fact)
m = 0, n > 0	negation of $(B_1 \wedge B_2 \wedge \ldots \wedge B_n)$
m = 0, n = 0	$\Leftarrow$ the empty clause (contradiction)

- In logic, all clauses can be represented as Horn Clauses...

 Paul F. Reynolds, Jr.

Proof by Refutation

- An important proof method:
  - P: set of axioms
  - Q: clause to be proven
  - show $P \wedge \neg Q$ 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:

$$\begin{array}{l} A_1 \Leftarrow \quad (B_1 \wedge B_2 \wedge \ldots \wedge B_n) \\ \quad\Leftarrow \quad\quad A_1 \wedge A_2 \wedge \ldots \wedge A_m \\ \hline \quad\Leftarrow \quad (B_1 \wedge B_2 \wedge \ldots \wedge B_n) \wedge A_2 \wedge \ldots \wedge A_m \end{array}$$

  $\rightarrow$ Keep applying this until $\Leftarrow$ is achieved.

 Paul F. Reynolds, Jr.

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.

 Paul F. Reynolds, Jr.

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"

 Paul F. Reynolds, Jr.

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.

 Paul F. Reynolds, Jr.

More Proofs

- Using:

```
(1)  a(X) ⇐  m(X)
(2)  m(X) ⇐  e(X)
(3)  e(c) ⇐
(4)  a(X) ⇐  s(X)
(5)  s(b) ⇐
(6)       ⇐  a(X)
```

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

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

 Paul F. Reynolds, Jr.

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.

 Paul F. Reynolds, Jr.

Bottom Up

- Using:

```
(1)  a(X) ⇐  m(X)
(2)  m(X) ⇐  e(X)
(3)  e(c) ⇐
(4)  a(X) ⇐  s(X)
(5)  s(b) ⇐
(6)       ⇐  a(X)
```

- Combine rule (2) with fact (3) yielding: combined with rule (1) yields:

```
m(X) ⇐  e(X)  -- combining
e(c) ⇐        -- rule with
m(c) ⇐        -- a fact  yields
a(X) ⇐  m(X)  -- a new
a(c)          -- fact
```

- or Combine rule (1) with rule (2) yields:

```
a(X) ⇐  m(X)        -- combining rules
m(X) ⇐  e(X)        -- to make a new
a(X) ⇐  e(X)        -- rule
```

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

 Paul F. Reynolds, Jr.

Closed World Assumption

- Inabilty 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.

 Paul F. Reynolds, Jr.

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

```
form:      p0 :- p1, p2, p3, p4
                 ^---^---^---^----  can be executed
```

concurrently(with communications about bindings) -- "AND parallelism"

```
or:     HG :- ..............      {
        HG :- ..............      {   "OR
        ...                        {   parallelism"
        HG :- ..............      {
```


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