Programming with Logic
Supplementary Lecture

Stephen M. Watt

Prolog

- A programming language for logic programming.
- Based on first-order logic (predicate calculus).
- Popular in some areas of artificial intelligence.
- Developed in France 1970-1972 by Alain Colmerauer et al.

Horn Clauses

- Disjunction of literals with at most one literal not negated E.g.
  - $\sim p \vee \sim q \vee \sim r$
  - a
  - $\sim p \vee \sim q \vee a$
- Named for Alfred Horn, American mathematician. Pointed out utility in constructive logic.

  "On sentences which are true of direct unions of algebras", Journal of Symbolic Logic, 16, 14-21

Horn Clauses

- Note equivalence of

  $(p \land q \land r) \rightarrow s \qquad \sim p \lor \sim q \lor \sim r \lor s$

- Write as

  $s \leftarrow (p \land q \land r)$

- In Prolog

  s :- p, q, r .

Types of Horn Clause

- "Definite clause", "Rule" : $\sim p \lor \sim q \lor \sim r \lor s$
- "Fact" : u
- "Goal clause" : $\sim p \lor \sim q \lor \sim r$

Prolog Terms

- Atoms:
  n, guppy, ‘big dipper’, ‘George’
- Numbers:
  42, 1.2345
- Variables: *start with capital or underscore*
  X, Father, _mumble
- Compound terms: *atom(term, term, ...)*
  edge(a, b), ‘Banana’(peel, Taste)
- Lists:
  [term1, term2, term3]
- Strings:
  “Now is the time for all good men to come to.”

Prolog Facts

- Statement of facts:

```
the_dog_is_big .

hasSon(joe, joe_jr) .
```

- Querying facts:

```
?- the_dog_is_big .
   true.

?- hasSon(joe,joe).
   false.
```

+ Querying Facts with Variables

- Statement of facts:

```
the_dog_is_big .

hasSon(joe, joe_jr) .
```

- Querying facts:

```
?- hasSon(joe, Who).
```

Who = joe_jr

Rules

- General form:

  term :- term, term, …, term .

- A “fact” is equivalent to a rule of the form

  term :- true .

Using Rules

- Suppose Sarah and Bill have two children, Brenda and Barb, and that Sarah also has a daughter Dana with father Dave. Suppose also that Brenda is the proud mother of Jane.

  Using the literal *parent(P, C)* to state *P* is a parent of *C*, we would write:

  parent(sarah, brenda).
  parent(bill, brenda).
  parent(sarah, barb).
  parent(bill, barb).
  parent(sarah, dana).
  parent(dave, dana).
  parent(brenda, jane).

Using Rules

- We can now write some useful rules:

```
grandparent(G,C) :- parent(G,X), parent(X,C).

sibling(A,B) :- parent(P, A), parent(P, B).
```

- We can then ask:

```
?- grandparent(sarah, jane).
   true .

?- parent(sarah, X).
   X = brenda .

?- sibling(sarah, jane).
   false .
```

Practical Considerations

- Matching is done by unification.
- Some special predicates have side effects, e.g.   write(X), nl

Using Side Effects

- grandparent(G,C) :- parent(G,X), parent(X,C), write(X), nl .

  ?- grandparent(sarah, bill).

    false.

  ?- grandparent(sarah, jane).

    brenda

    true.

A Few Builtins

- Numerical order predicates: A < B, A >= B, etc
- Unification predicate: A = B
- Head/Tail split of list: [H | T]
- Concatenating lists: append([List1, List2, ...], ResultList)

A More Serious Programming Example (still naïve)

```
partition([], _, [], []).

partition([X|Xs], Pivot, Smalls, Bigs) :-
    X < Pivot,  Smalls = [X|Rest], partition(Xs, Pivot, Rest, Bigs) .

partition([X|Xs], Pivot, Smalls, Bigs) :-
    X >= Pivot, Bigs   = [X|Rest], partition(Xs, Pivot, Smalls, Rest).

quicksort([], Sorted) :- Sorted = [].

quicksort([X|Xs], Sorted) :-
        partition(Xs, X, Smaller, Bigger),
        quicksort(Smaller, SortedSmaller),
        quicksort(Bigger,  SortedBigger),
        append( [SortedSmaller, [X], SortedBigger], Sorted ).
```

Running Quicksort

```
slice:~/Documents/Current/Prolog [1.47] $ ./prolog
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.2.3)
Copyright (c) 1990-2012 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [sorting].
% sorting compiled 0.00 sec, 6 clauses
true.

?- quicksort([9,1,8,2,7,3,6,4,5], Sorted).
Sorted = [1, 2, 3, 4, 5, 6, 7, 8, 9] .

?-
```

Industrial Strength Prolog

- Cut, !, always succeeds, but cannot be backtracked.
- Negation as failure for "non-monotonic" reasoning: `\+/1`
- Module system.
- Memo-ization to save previous computations.
- Extensions with types, modes (input vs output), constraints, higher order programming, object orientation, etc.

+ Going Further with Prolog

- A nice Prolog tutorial:

  http://www.doc.gold.ac.uk/~mas02gw/prolog_tutorial/prologpages/

- comp.lang.prolog FAQ

  http://www.logic.at/prolog/faq/faq.html

More Sophisticated Systems

- Isabelle:
  - Interactive theorem prover
  - Written in Standard ML
  - Provides a “meta logic” which is used to encode specific logics such as first order logic, higher order logic, Zermelo-Fraenkel set theory.
  - Main proof method is higher-order version of resolution.
  - Features some automatic reasoning tools.
  - Used by HP in hardware design to catch bugs not found in testing.

More Sophisticated Systems

- Coq :
  - Interactive theorem prover
  - Implemented in Ocaml
  - Developed in France (INRIA, X, etc)
  - Curry-Howard isomorphism (proofs as programs)
  - Calculus of inductive constructions
    (work by Thierry Coquand, Gerard Huet)
  - Dependent typed functional programming language
  - Four color map theorem 2004
  - Feit-Thompson theorem 2012
    (important in classification of finite simple groups)