Logic Programming

- Declarative programming style
- Example: Prolog
  - Fundamental concept: First-order Predicate Logic
  - The execution of a Prolog program is a proof
  - A proof is an execution of a Prolog program

Horn Clauses

- Conjunctive Normal Form
  - Conjunction of clauses, clauses are disjunction of literals
  - \((A \lor B) \land (C \lor D \lor E)\) ...

- Horn Clauses
  - At most one literal is positive, the others are negative

- Definite Horn Clauses
  - Exactly one positive literal
  - \(A \lor \neg B \lor \neg C \lor \neg D\) ...
  - \(A \leftarrow B \land C \land D\) ...

Facts

- Facts model a relation between elements.
- Facts are definite Horn clauses without negative literals

- Example:
  - olympics.
  - olympics(1896, athens).
  - olympics(1908, london).
  - olympics(2012, london).
  - olympics(2020, tokyo).
Rules

- Conditional expressions of the form
  - A :- B, C, D, ... Z
  - Semantics: A becomes true if B is true and C is true, ...

- Example:
  - hostCity(X) :- olympics(_, X).
  - megaHostCity(City) :- olympics(_, City), population(City, Size), Size > 5000000.

- Variables need to start with capital letter or underscore.
- '_' means "don't care"

Queries

- Queries are expressions that Prolog should try to proof by binding the free variable in such a way that the expression becomes true.
  - Queries are negative Horn clauses
    - ← A ∧ B ∧ ...

- Example:
  - olympics(1906, london).
  - olympics(X, atlanta).
  - olympics(1896, Y).
  - olympics(Z, athens).

Queries

- More facts:
  - in_country(atlanta, usa).
  - in_country(athens, greece).
  - ...
  - continent(usa, northAmerica).
  - continent(greece, europe).
  - ...
  - olympics_by_continent(Continent, Year, City) :- olympics(Year, City), in_country(City, Country), continent(Country, Continent).

Notation

- <predicate_name>/<arity>
  - olympics/0, olympics/2

- Predicate Description
  - +
    - Argument must be fully instantiated to a term that satisfies the required argument type. Think of the argument as input.
  - -
    - Argument must be unbound. Think of the argument as output.
  - ?
    - Argument must be bound to a partial term of the indicated type. Note that a variable is a partial term for any type. Think of the argument as either input or output or both input and output.

From the SWI Prolog Manual
Resolution
- gradStudent(bill).
- gradStudent(sally).
- newborn(tom).
- newborn(mary).
- father(bill, mary).
- father(bill, joe).
- mother(sally, tom).
- parent(X, Y) :- father(X, Y).
- parent(X, Y) :- mother(X, Y).
- tired(X) :- gradStudent(X), parent(X, Y), newborn(Y).

SLD-Resolution
← tired(Z)
← tired(X) :- gradStudent(X), parent(X, Y), newborn(Y).
← gradStudent(Z), parent(Z, Y), newborn(Y).
← gradStudent(bill).
← parent(bill, Y), newborn(Y).
← parent(X, Y) :- father(X, Y).
← father(bill, Y), newborn(Y).
← father(bill, goey).
← newboran(goey).
← newborn(mary).
← newboran(mary).

SLD-Resolution
← tired(Z)
← tired(X) :- gradStudent(X), parent(X, Y), newborn(Y).
← gradStudent(Z), parent(Z, Y), newborn(Y).
← gradStudent(bill).
← parent(bill, Y), newborn(Y).
← parent(X, Y) :- father(X, Y).
← father(bill, Y), newborn(Y).
← father(bill, mary).
← father(bill, mary).
← parent(sally, Y), newborn(Y).
← parent(X, Y) :- father(X, Y).
← parent(sally, Y), newborn(Y).
← parent(sally, Y), newborn(Y).
← parent(sally, Y), newborn(Y).
Structured Data and Pattern Matching

- Creation of compound terms
  - foo(Y) :- bar(2018, Y).
  - bar(X, result("olympics", X)) :- olympics (X, _).

- Pattern matching
  - foobar(result("olympics", X)) :- …
  - foobar(result("soccer world cup", X)) :-

- Example: Iterating over Data
  - iter(node(L,R)) :- iter(L), iter(R).
  - iter(X) :- write("value "), writeln(X).

Lists

- The empty list is a list.
- A list can be described as [Head|Tail] where Head is an element and Tail is a list.

  - We write:
    - []
    - [a,b,c]
      - this is the same as [a][b][c]]

- List membership:
  - member(X, [X|T]).
  - member(X, [Y|T]) :- member(X, T).

Lists

- Appending lists:
  - append([], X, X).
  - append([X|L1], Y, [X|L2]) :- append(L1, Y, L2).

Failure-driven Loops

- olympics(X, athens), write(‘athens: ‘), writeln(X), fail; true.
Conditions

- `olympics(X, Y), writeln(X), Y == atlanta -> writeln(Y).`

- `olympics(X, Y), Y == athens -> writeln(' in greece'); writeln(' somewhere else').`

The Cut operator

- `olympics(X, Y), write(Y), write(': '), writeln(X), Y == atlanta -> !, fail; true.`

Negation

- **Negation as failure**
  - `not(Goal) :- Goal, !, fail.`
  - `not(Goal).`

  Built-in predicate in SWI-Prolog but deprecated.

  - `\+`
    - `notHost(City) :- \+ hostCity(City).`

Equality

- `=`
  - Unification
  - `==, \==`
    - Term equality, inequality
  - `=:, =\=`
    - Arithmetic or boolean equality, inequality
  - `is/2`
    - Evaluation and unification

- `1==1.0.`
- `false`
- `1==:1.0.`
- `true`
Arithmetic

- Factorial
  \[ n! = \begin{cases} 
  1 & (n = 0) \\
  (n - 1)! \times n & (n > 0) 
\end{cases} \]

- \texttt{factorial(0,1)}.
- \texttt{factorial(N,M) :- \_n is N-1, factorial(_n, _fact), M is N * _fact.}

Operators

- \texttt{op/3 (Directive)}
- \texttt{op(+Precedence, +Type, :Name)}
  - Precedence is number between 0 and 1200
    - Operators like + have ~ 200, */ have 400.
  - Type is
    - xfx, xfy, yfx for infix
    - fx, fy for prefix
    - xf, yf for suffix
- \texttt{:- op(Precedence, Type, Name).}

DCG Grammars

- Parsing:
  - \texttt{s --> [olympic], [games], year, [in], city.}
  - year --> [Num], { number(Num) }.
  - city --> [City].
  - phrase(s, ['olympic', 'games', 1984, 'in', 'losAngeles']).

- Syntactic sugar over difference lists:
  - \texttt{s(S1, S2) :- olympic(S1, _t1), games(_t1, _t2), year(_t2, _t3), in(_t3, _t4), city(_t4, S2).}

- Desugared version
  - \texttt{s([olympic][A], E) :-}
    - A=[games][B],
    - year(B, C),
    - C=[in][D],
    - city(D, E)
  - year([A|C], B) :-
    - number(A),
    - B=C.
  - city([_|A], A).

- Can be inspected with listing//1.
DCG Grammars

- Parsing:
  
  - s(Year, City) → [olympic], [games], year(Year), [in], city(City).
  
  - year(Num) → [Num], { number(Num) }.
  
  - city(City) → [City].

- phrase(s(Year, City), ['olympic', 'games', 1984, 'in', 'losAngeles']).

Association Lists

- empty_assoc(-Assoc)

- put_assoc(+Key, +Assoc, +Value, ?NewAssoc)

- get_assoc(+Key, +Assoc, ?Value)

The (ugly) Truth About Unification

- ?- = (X=f(X), X).
  
  - X = (X=f(X)).

- ?- unify_with_occurs_check(X=f(X), X).
  
  - false.

The Zebra Puzzle

- There are five houses.
- The Englishman lives in the red house.
- The Spaniard owns the dog.
- Coffee is drunk in the green house.
- The Ukrainian drinks tea.
- The green house is immediately to the right of the ivory house.
- The Old Gold smoker owns snails.
- Kools are smoked in the yellow house.
- Milk is drunk in the middle house.
- The Norwegian lives in the first house.
- The man who smokes Chesterfields lives in the house next to the man with the fox.
- Kools are smoked in the house next to the house where the horse is kept.
- The Lucky Strike smoker drinks orange juice.
- The Norwegian lives next to the blue house.
- Now, who drinks water? Who owns the zebra?