Chapter 11 Logic Programming Language (LPL)

Overview

**Important characteristics**

* Use logical statements to describe facts and relationships – conceptually, it is based on and similar to predicate calculus, specifically, [horn clause](http://en.wikipedia.org/wiki/Horn_clause) in implication form.

Clause - a disjunction of literals, ex., A ˅ B ˅ C …

Where A, B, C are either literals (atom and their negation) or their conjunctive.

Horn Clause - a clause with at most one positive,

Definitive clause – a horn clause with exactly one positive.

ex., ¬A ˅ B ˅ ¬C // A definitive clause,

=== A & C 🡪 B // can be transformed to describe conditional truth

ex., A // a horn clause

ex., C

We can drive new fact from existing ones, ex., B is true based on the previous 3 clauses

---------- Predicate Calculus (PC) – a quick review -----------

11.3 Theoretical Foundation

Express these using First Order Logic (FOL==PC)

1. It’s cold in Edwardsville // propositional logic, NOT FOL

cold(Edwardsville) // 1st order logic allows variables and as such, is more versatile.

cold(STL) // It’s cold in the Standard Template Library

1. if it’s raining, it is wet. // conditional truth on universal var

x Raining(x) 🡪 Wet(x) // true for all values of x, == for all

FOL express fact, relationships (i.e., rules)

By using:

* Variables, ex., x
* Terms, ex.,

raining(STL) // better than “Raining in STL” in propositional logic

* *Two quantifiers:*

1. *Universal* quantification - (for all)

* Express general rules and facts that are true for ALL objects, ex.,

// if it’s raining, it’s wet

x raining(x) 🡪 wet(x) // if it’s raining in/at X (location), it’s wet in/at X

* It is almost always used with => (🡪) to describe rules, conditional facts.

Can we say: x raining(x) ∧ wet(x) ?

*2, Existential* quantification - (there exists)

* Express facts about *some* object without naming it, ex.,

// Somewhere is raining

x raining(x)

// True with *at least* one value of x

x Raining(x) => Wet(x) // rules

Raining(STL) // fact

// via deduction or



// resolution

x = STL 🡺 Wet(STL) // conclusion and var binding, creation of a bond

---------- end of FOL recap -------

**Characteristics of Logical Programming Languages in general and prolog in particular.**

* Declarative (remember SQL?)

A logic program contains facts (unconditional truth) and rules (conditional truth) of our mini-world.

* 1. *Facts* – What is already known and believed to be true, i.e., what’s known
  2. *Rules* – Describe how new facts can be inferred from known ones, the relationship that allows us (or whoever does the inference) to gain/derive new knowledge.

grandfather(X, Y) :- father(X, Z), father(Z, Y).

% X is grandfather of Y if X is father of Z and Z is father of Y.

* 1. *Queries (or Goals) – where you ask Yes/No questions.* ex.,

?- grandfather(john, david). % Is john grandfather of david given known facts?

?- grandfather(john, G). % list all grandchildren of john,

% *use ; to get additional answer in swi-prolog*

?- grandfather(Y, X).

% list all grandfather/grandchild pairs in the family

**Notes on variables:**

1. Names are Capitalized.
2. Only appears in rules or queries.
3. Always of the universal type

A logic program does not need to describe h*ow* a result is to be computed, or the actual procedure for verifying the goal - *no algorithm (at least not explicitly), just constraints or relationships.*

Who is responsible for the algorithm and finding the solution for us?

How does it work?

The Language Implementation System (Ex., SWI-Prolog)

* Developed in 1980s (1st release 1987) by the **Sociaal-Wetenschappelijke Informatica** ("Social Science Informatics") group at the University of Amsterdam led by Wielemaker.
* Uses deductive reasoning to answer the queries, these queries also called goals.
* It does so by searching for ALL possible ways of satisfying the goal (i.e., the query), *using depth-first search with backtracking!!! (DFS)*
* Clauses are tried in the order in which they are listed in your program (i.e., knowledge base).
* Conditions are tried from left to right.

**Prolog**

1. Developed at the University of Marseilles, France by [Alain Colmerauer](http://alain.colmerauer.free.fr/) and colleagues in the early 1970s.
2. Why Prolog
   1. A new problem-solving technique, new way of thinking, a new paradigm,
   2. Used for many applications: AI, Expert Systems, compiler, and database applications.
3. There are many prolog implementation systems:
4. [Swi-prolog](http://www.swi-prolog.org/) 9.0.x (stable release)
   * + Not as restrictive as visual
     + Many build-in predicates (ex., permutation)
     + Better tracing
     + More interactive
     + Small, fast (compilation), and robust
     + Graphics possible with XPCE toolkit
5. [Visual Prolog](http://www.visual-prolog.com/) 
   * + Better GUI
     + Much beyond pure logical programming
     + Bigger
     + Require extra declaration and program formats.
     + Requires more resources (CPU and memory)
6. Many others, some incorporate OOP capabilities: <https://en.wikipedia.org/wiki/Comparison_of_Prolog_implementations>

**Example of problem solving in Prolog - Suite design (floor plan)**

Motel 7 decided to add suite to accommodate the need of their VIP guests. They want to find out *all possible* *directions* in which the *doors* and *windows* may face in their suites. Each suite must follow these basic requirements/constraints:

1. There will be two doors: the Front Door (FD) to the lounge and a Bedroom Door (BD) and two windows: Lounge Window (LW) and Bedroom Window (BW).
2. The LW should be opposite to the FD.
3. The BD should be in one of the walls adjacent to the FD to provide a little privacy.
4. The BW should be in one of the walls adjacent to the BD.
5. The BW should face east to catch the morning light.

How do we solve this problem?

1. Manually



How many solutions are there?

1. Use C++

4 variables: FD, BD, LW, BW each with four possible values: {east, south, west, north}

How do we find all possible solutions?

1. Use prolog.

We set up the constraints and facts.

See suite.pl and suite.cpp

Continue to prologBasic.doc