CPSC 312 — Functional and Logic Programming

- Assignment 4 is due next Thursday!
- No textbook for logic programming; see readings tab.
- Get SWI Prolog.
- "Learn at least a half dozen programming languages. Include one language that emphasizes class abstractions (like Java or C++), one that emphasizes functional abstraction (like Lisp or ML or Haskell), one that supports syntactic abstraction (like Lisp), one that supports declarative specifications (like Prolog or C++ templates), and one that emphasizes parallelism (like Clojure or Go)."

  Peter Norvig "Teach Yourself Programming in Ten Years"
  `http://norvig.com/21-days.html`

Plan

- Logic Programming
  - Propositional logic programs
    - Semantics
    - Bottom-up and top-down proof procedures
  - Datalog
  - Logic programs with function symbols
  - Applications (e.g., natural language processing)
  - Semantic web

Today:
  - Syntax and semantics of propositional definite clauses
  - Model a simple domain using propositional definite clauses
  - Bottom-up proof procedure

What is Logic programming

- Functional programming + search + flexible pattern matching + relations
- As a simple database language (Datalog) + function symbols (= data constructors)
- Statements of a subset of first-order logic, with procedural interpretation
- Prolog started as a tool to write natural language understanding systems (and is used today in controlled natural language situations).

Haskell vs Prolog Example: append (first.pl)

Haskell:

```
-- append [a1,a2,..an] [b1,..,bm] = [a1..an,b1,..,bm]
append [] l2     = l2
append (h:r) l2 = h : append r l2
```

Prolog

```
% append([a1,a2,..an], [b1,..,bm], [a1..an,b1,..,bm])
append([], L2, L2).
append([H|R], L2, [H|L3]) :-
    append(R,L2,L3).
```

Some Prolog queries:

```
append([1,2,3], [7,8,9], R).
append([1,2], X, [1,2,3,4,5]).
append(X,Y,[1,2,3,4,5]).
append(X,[3|Y],[1,2,3,4,5,4,3,2,1]).
```

Haskell vs Prolog Example: del1

Delete one instance of an element from a list. Haskell:

```
del1 :: Eq e => e -> [e] -> Maybe [e]
del1 _ [] = Nothing
del1 e (h:t)
    | e==h     = Just t
    | otherwise =  fmap (h:) (del1 e t)
```

Prolog:

```
% del1(E,L,R) is true if R is the L with one E removed
del1(E,[E|Y],Y).
del1(E,[H|T],[H|Z]) :-
    del1(E,T,Z).
```

Some Prolog queries:

```
del1(a,[a,v,a,t,a,r],A).
del1(b,[a,v,a,t,a,r],A).
```

Datalog program: family relationships (family.pl)

```
% father(X, Y) means X is the father of Y
father(pierre, justin).
father(pierre, alexandre).
father(pierre, michel).
father(justin, xavier).
father(justin, ella_grace).

% mother(X, Y) means X is the mother of Y
mother(margaret, justin).
mother(margaret, alexandre).
mother(margaret, michel).
mother(sophie, xavier).
mother(sophie, ella_grace).

% Also defined: parent, grandmother, sibling, ancestor
```

A progression of logical languages

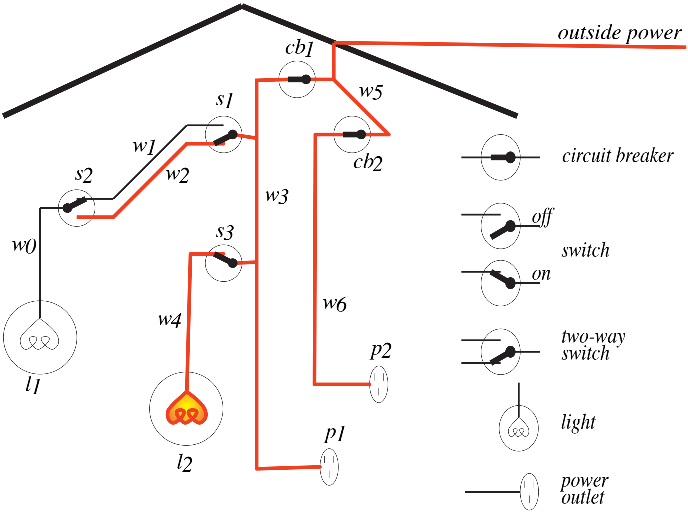
- Propositional logic programs: atoms have no arguments.
- Datalog: allow for logical variables in clauses.
- Pure Prolog: Datalog + function symbols

Propositional Logic Program Syntax

- An atom is of the form p, a word that (can contain letters, digits and underscore _) and starts with a lower-case letter.
- A body is either
  - an atom or
  - (b_1, b_2) where b_1 and b_2 are bodies. (Parentheses are optional). A comma in a body means "and".
- A definite clause is either
  - an atomic clause: an atom or
  - a rule: h :- b where h is an atom and b is a body.
    :- means "if"

  An atomic clause is treated as a rule with an empty body.
  All definite clauses ends with a period "."
- A logic program or knowledge base is a set of definite clauses
- A query is a body that is asked at the Prolog prompt (ended with a period).

Electrical Environment



Example Knowledge Base (*elect_prop.pl*)

$light\_l_1.$
$light\_l_2.$
$down\_s_1.$
$up\_s_2.$
$up\_s_3.$
$ok\_l_1.$
$ok\_l_2.$
$ok\_cb_1.$
$ok\_cb_2.$
$live\_outside.$

$lit\_l_1 \text{ :- } live\_w_0,\ ok\_l_1$
$live\_w_0 \text{ :- } live\_w_1,\ up\_s_2.$
$live\_w_0 \text{ :- } live\_w_2,\ down\_s_2.$
$live\_w_1 \text{ :- } live\_w_3,\ up\_s_1.$
$live\_w_2 \text{ :- } live\_w_3,\ down\_s_1.$
$lit\_l_2 \text{ :- } live\_w_4,\ ok\_l_2.$
$live\_w_4 \text{ :- } live\_w_3,\ up\_s_3.$
$live\_p_1 \text{ :- } live\_w_3.$
$live\_w_3 \text{ :- } live\_w_5,\ ok\_cb_1.$
$live\_p_2 \text{ :- } live\_w_6.$
$live\_w_6 \text{ :- } live\_w_5,\ ok\_cb_2.$
$live\_w_5 \text{ :- } live\_outside.$

Clicker Question

Which of the following is a clause?

A $happy \;\texttt{:-}\; Good.$

B $happy, rich \;\texttt{:-}\; good.$

C $happy \;\texttt{:-}\; .$

D $rich; sad \;\texttt{:-}\; good.$

E None of the above

Human's view of semantics

Step 1 Begin with a task domain.

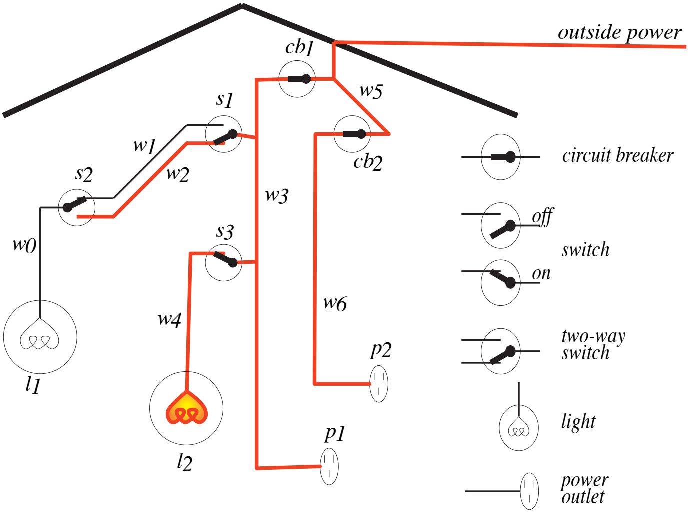
Step 2 Choose atoms in the computer to denote propositions. These atoms have meaning to the KB designer.

Step 3 Tell the system knowledge about the domain.

Step 4 Ask the system questions.

— The system gives answers.
— Person can interpret the answer with the meaning associated with the atoms.

Electrical Environment



Role of semantics in electrical domain

In user's mind:

- *light1_broken*: light 1 is broken
- *sw1_up*: switch 1 is up
- *sw2_up*: switch 2 is up
- *power*: there is power in the building
- *unlit_light1*: light 1 isn't lit
- *lit_light2*: light 2 is lit

In computer:

$$light1\_broken \;\texttt{:-}\; sw1\_up,\; sw2\_up,\; power,\; unlit\_light1.$$
$$sw1\_up.$$
$$sw2\_up.$$
$$power \;\texttt{:-}\; lit\_light2.$$
$$unlit\_light1.$$
$$lit\_light2.$$

Conclusion: *light1_broken*

- The computer doesn't know the meaning of the symbols
- The user can interpret the symbol using their meaning

Semantics

- An interpretation I assigns a truth value to each atom.
- True of compound propositions in interpretation is derived from truth table:

p	q	$p,\ q$	p :- q
true	*true*	*true*	*true*
true	*false*	*false*	*true*
false	*true*	*false*	*false*
false	*false*	*false*	*true*

- A body $(b_1,\ b_2)$ is true in I if b_1 is true in I and b_2 is true in I.
- A rule h :- b is false in I if b is true in I and h is false in I. The rule is true otherwise.
- A knowledge base KB is true in I if and only if every clause in KB is true in I.

Models and Logical Consequence

- A model of a set of clauses is an interpretation in which all the clauses are *true*.
- If KB is a set of clauses and g is a conjunction of atoms, g is a logical consequence of KB, written $KB \models g$, if g is *true* in every model of KB.
- That is, $KB \models g$ if there is no interpretation in which KB is *true* and g is *false*.

Simple Example (clicker question)

$$KB = \begin{cases} p \;\texttt{:-}\; q. \\ q. \\ r \;\texttt{:-}\; s. \end{cases}$$

A yes

B no

C I'm not sure

	p	q	r	s	model of KB?
I_1	*true*	*true*	*true*	*true*	is a model of KB
I_2	*false*	*false*	*false*	*false*	not a model of KB
I_3	*true*	*true*	*false*	*false*	is a model of KB
I_4	*true*	*true*	*true*	*false*	is a model of KB
I_5	*true*	*true*	*false*	*true*	not a model of KB

Does p, q, r, s logically follow from KB?

$KB \models p$, $KB \models q$, $KB \not\models r$, $KB \not\models s$