Logic Programming with Prolog

Prolog is based on three main ideas:

- Logical Horn rules (last day)
- Unification (today)
- Top-down reasoning (next day)

Prolog Notation

For convenience, we often don't write universal quantifiers, so the rule

$\forall$`X [p(X)` $\leftarrow$ `(q(X)` $\wedge$ `r(X))]`

is written as

`p(X)` $\leftarrow$ `q(X), r(X).`

We will also use the Prolog conventions:

- *variables* begin in upper case.
  e.g., `A, B, X, Y, Big, Small, ACE`
- *constants* begin in lower case.
  e.g., `a, b, x, y, abbot, costello`

Database Queries

Consider the following database of facts:

```
loves(bob,sue)
loves(sue,tony)
loves(sue,harry)
loves(X,santa)     i.e., ∀X loves(X,santa)
                   i.e., "everyone loves santa"
loves(jesus,X)     i.e., ∀X loves(jesus,X)
                   i.e., "jesus loves everyone"
```

We would like to pose questions (queries) to this database.

e.g., "Who does sue love?"

i.e., Find `Y` such that `loves(sue,Y)` is true.

answers: `Y = tony`
`Y = harry`
`Y = santa`

Another Query

<u>Same Database</u>:

```
loves(bob,sue)
loves(sue,tony)
loves(sue,harry)
loves(X,santa)
loves(jesus,X)
```

<u>Query</u>: "Who loves sue?"

i.e., Find `Y` such that `loves(Y,sue)` is true.

answers: `Y = bob`

`Y = jesus`

Conjunctive Queries

Same Database:

```
loves(bob,sue)
loves(sue,tony)
loves(sue,harry)
loves(X,santa)
loves(jesus,X)
```

Query: "Find someone who bob loves
AND who loves tony."

i.e., Find `Y` such that the formula
`loves(bob,Y)` $\wedge$ `loves(Y,tony)` is true.

answer: `Y = sue`

Disjunctive Queries

Same Database:

```
loves(bob,sue)
loves(sue,tony)
loves(sue,harry)
loves(X,santa)
loves(jesus,X)
```

Query: "Find someone who bob loves
OR who loves tony."

i.e., Find `Y` such that the formula
`loves(bob,Y)` $\vee$ `loves(Y,tony)` is true.

answers: `Y = sue`
`Y = santa`
`Y = sue`
`Y = jesus`

To answer queries, Prolog uses Unification.

Unification

Two atomic formulas with distinct variables unify if and only if they can be made syntactically identical by replacing their variables by other terms. For example,

- `loves(bob,Y)` unifies with `loves(bob,sue)` by replacing `Y` by `sue`.

- `loves(bob,Y)` unifies with `loves(X,santa)` by replacing `Y` by `santa` and `X` by `bob`.

  Both formulas become `loves(bob,santa)`.

  Formally, we use the <u>substitution</u>

  $$\{\texttt{Y}\backslash\texttt{santa},\ \texttt{X}\backslash\texttt{bob}\}$$

  which is called a <u>unifier</u> of `loves(bob,Y)` and `loves(X,santa)`.

- Note that `loves(bob,X)` does *not* unify with `loves(tony,Y)`, since no substitution for `X,Y` can make the two formulae syntactically equal.

Abstract Examples

- `p(X,X)` unifies with `p(b,b)` and with `p(c,c)`, but *not* with `p(b,c)`.

  Why? Because each occurrence of `X` must be replaced by the *same* term.

- `p(X,b)` unifies with `p(Y,Y)` with unifier $\{\mathtt{X}\backslash\mathtt{b},\ \mathtt{Y}\backslash\mathtt{b}\}$ to become `p(b,b)`.

- `p(X,Z,Z)` unifies with `p(Y,Y,b)` with unifier $\{\mathtt{X}\backslash\mathtt{b},\ \mathtt{Y}\backslash\mathtt{b},\ \mathtt{Z}\backslash\mathtt{b}\}$ to become `p(b,b,b)`.

A Negative Example

`p(X,b,X)` does *not* unify with `p(Y,Y,c)`.

Reason:

- To make the third arguments equal,
  we *must* replace `X` by `c`.

- To make the second arguments equal,
  we *must* replace `Y` by `b`.

- So, `p(X,b,X)` becomes `p(c,b,c)`,
  and `p(Y,Y,c)` becomes `p(b,b,c)`.

- However, `p(c,b,c)` and `p(b,b,c)`
  are *not* syntactically identical.

- Moreover, they cannot be made identical by additional substitutions, since they have no variables.

Function Terms

- In `love(sue,tony)`, `sue` and `tony` are *constant symbols* (very simple data structures).

- We can construct more complex data structures using *function terms*.

- *e.g.*, consider the atomic formula

  `owns(john, car(blue,corvette))`

  It represents a statement about the world. It is either `true` or `false`.

- Inside this formula, `car(blue,corvette)` is a function term. It represents an object in the world. It does not have a truth value.

- Function terms can represent complex, structured objects.

Example 1

<u>Database</u>:

```
owns(john, car(red,corvette))
owns(john, cat(black,siamese,sylvester))
owns(elvis, copyright(song,"jailhouse rock"))
owns(tolstoy, copyright(book,"war and peace"))
owns(elvis, car(red,cadillac))
```

<u>Query</u>:
"Retrieve everything that John owns."
i.e. , Find `X` such that `owns(john,X)` is true.

```
answers:  X = car(red,corvette)
          X = cat(black,siamese,sylvester)
```

Example 2

Same Database:

```
owns(john, car(red,corvette))
owns(john, cat(black,siamese,sylvester))
owns(elvis, copyright(song,"jailhouse rock"))
owns(tolstoy, copyright(book,"war and peace"))
owns(elvis, car(red,cadillac))
```

Query:

"Retrieve the colour and make of John's car."

i.e., `owns(john,car(Colour,Make))`

```
answer:  Colour = red
         Make = corvette
```

Example 3

Same Database:

```
owns(john, car(red,corvette))
owns(john, cat(black,siamese,sylvester))
owns(elvis, copyright(song,"jailhouse rock"))
owns(tolstoy, copyright(book,"war and peace"))
owns(elvis, car(red,cadillac))
```

Query:
"Who owns the copyright on the song
'Jailhouse Rock' ?"

i.e., `owns(Who,copyright(song,"jailhouse rock"))`

```
answer:  Who = elvis
```

Existential Queries

Same Database:

```
owns(john, car(red,corvette))
owns(john, cat(black,siamese,sylvester))
owns(elvis, copyright(song,"jailhouse rock"))
owns(tolstoy, copyright(book,"war and peace"))
owns(elvis, car(red,cadillac))
```

Query: "Who owns a red car?"

i.e., Find values for `Who` so that

$\exists$`Make owns(Who,car(red,Make))` is true.

```
answers:  Who = john
          Who = elvis
```

- Quantified variables (like `Make`) are said to be bound.
- Unquantified variables (like `Who`) are said to be free.
- Only the values of *free* variables are included in the answers to queries.

Another Existential Query

Same Database:

```
owns(john, car(red,corvette))
owns(john, cat(black,siamese,sylvester))
owns(elvis, copyright(song,"jailhouse rock"))
owns(tolstoy, copyright(book,"war and peace"))
owns(elvis, car(red,cadillac))
```

Query: "Who owns a copyright?"

i.e., Find values for `Who` so that

$\exists$`X,Y owns(Who,copyright(X,Y))` is true.

```
answers:  Who = elvis
          Who = tolstoy
```

Syntactic Sugar

- For convenience, Prolog has a special notation for existentially quantified variables.

- *e.g.*, `owns(Who,_)` is an abreviation for

  $\exists$`X owns(Who,X)`

  i.e., "Who owns something?"

- Each occurance of an underscore represents a *different* variable.

- *e.g.*, `owns(Who,copyright(_,_))` is an abreviation for

  $\exists$`X,Y owns(Who,copyright(X,Y))`

  i.e., "Who owns a copyright of some kind on something?"

Unification with function terms

Prolog uses unification to compute its answers.

e.g., Given the same database as before:

```
owns(john, car(red,corvette))
owns(john, cat(black,siamese,sylvester))
owns(elvis, copyright(song,"jailhouse rock"))
owns(tolstoy, copyright(book,"war and peace"))
owns(elvis, car(red,cadillac))
```

the query `owns(Who,car(red,Make))`
unifies with the following database facts:

- `owns(elvis,car(red,cadillac))`,
  with unifier {`Who\elvis`, `Make\cadillac`}

- `owns(john,car(red,corvette))`,
  with unifier {`Who\john`, `Make\corvette`}

Abstract Examples

- `p(f(X),X)` unifies with `p(Y,b)`
  with unifier $\{\mathtt{X}\backslash\mathtt{b},\ \mathtt{Y}\backslash\mathtt{f(b)}\}$
  to become `p(f(b),b)`.

- `p(b,f(X,Y),c)` unifies with `p(U,f(U,V),V)`
  with unifier $\{\mathtt{X}\backslash\mathtt{b},\ \mathtt{Y}\backslash\mathtt{c},\ \mathtt{U}\backslash\mathtt{b},\ \mathtt{V}\backslash\mathtt{c}\}$
  to become `p(b,f(b,c),c)`.

A Negative Example

`p(b,f(X,X),c)` does *not* unify with `p(U,f(U,V),V)`.

Reason:

- To make the first arguments equal, we *must* replace `U` by `b`.

- To make the third arguments equal, we *must* replace `V` by `c`.

- These substitutions convert `p(U,f(U,V),V)` into `p(b,f(b,c),c)`.

- However, *no* substitution for `X` will convert `p(b,f(X,X),c)` into `p(b,f(b,c),c)`.

Another Kind of Negative Example

`p(f(X),X)` does *not* unify with `p(Y,Y)`.

Reason:

- Any unification would require that
  $\mathtt{f(X)} = \mathtt{Y}$ and $\mathtt{Y} = \mathtt{X}$

- But no substitution can make
  $\mathtt{f(X)} = \mathtt{X}$

- For example,

  $\mathtt{f(a)} \neq \mathtt{a}$, using $\{\mathtt{X}\backslash\mathtt{a}\}$

  $\mathtt{f(b)} \neq \mathtt{b}$, using $\{\mathtt{X}\backslash\mathtt{b}\}$

  $\mathtt{f(g(a))} \neq \mathtt{g(a)}$, using $\{\mathtt{X}\backslash\mathtt{g(a)}\}$

  $\mathtt{f(f(c))} \neq \mathtt{f(c)}$, using $\{\mathtt{X}\backslash\mathtt{f(c)}\}$

  etc.

Most General Unifiers (MGU)

The atomic formulas `p(X,f(Y))` and `p(g(U),V)` have infinitely many unifiers. *e.g.*,

- {X\g(a), Y\b, U\a, V\f(b)}
  unifies them to give `p(g(a),f(b))`.

- {X\g(c), Y\d, U\c, V\f(d)}
  unifies them to give `p(g(c),f(d))`.

However, these unifiers are more specific than necessary.

The most general unifier (mgu) is

{X\g(U), V\f(Y)}

It unifies the two atomic formulas to give `p(g(U),f(Y))`.

Every other unifier results in an atomic formula of this form.

The mgu uses variables to fill in as few details as possible.

MGU Example

$$f(W,\ g(Z),\ Z)$$

$$f(X,\ Y,\ h(X))$$

To unify these two formulas, we need

$$\begin{array}{rcl} Y & = & g(Z) \\ Z & = & h(X) \\ X & = & W \end{array}$$

Working backwards from W, we get

$$\begin{array}{rcl} Y & = & g(Z) = g(h(W)) \\ Z & = & h(X) = h(W) \\ X & = & W \end{array}$$

So, the mgu is

$$\{X \backslash W,\ Y \backslash g(h(W)),\ Z \backslash h(W)\}$$

More MGU Examples

t_1	t_2	MGU
f(X,a)	f(a,Y)	
f(h(X,a),b)	f(h(g(a,b),Y),b)	
g(a,W,h(X))	g(Y,f(Y,Z),Z)	
f(X,g(X),Z)	f(Z,Y,h(Y))	
f(X,h(b,X))	f(g(P,a),h(b,g(Q,Q)))	

Syntax of Substitutions

Formally, a substitution is a set

$$\{v_1 \backslash t_1, \ldots, v_n \backslash t_n\}$$

where the	v_i's	are distinct variable names
and the	t_i's	are terms that do not use <u>any</u> of the v_j's.

Positive Examples:

$\{X \backslash a,\ Y \backslash b,\ Z \backslash f(a, b)\}$

$\{X \backslash W,\ Y \backslash f(W, V, a),\ Z \backslash W\}$

Negative Examples:

$\{f(X) \backslash a\}$

$\{X \backslash a,\ X \backslash b\}$

$\{X \backslash f(X)\}$

$\{X \backslash f(Y),\ Y \backslash g(q)\}$