CS 131

Saketh Kasibatla

Today

Prolog

This Class

homework 4 is due **Thursday, Nov. 9 at 23:55**

This homework will be graded with automated scripts

- not compiling → no credit
- your code should behave exactly according to the spec
- check Piazza for clarifications

Questions?

Install/use Prolog

We will use GProlog in the class

- http://www.gprolog.org

Make sure you're not using SWI-Prolog

Prolog Resources

gprolog manual: http://www.gprolog.org/manual/gprolog.html

prolog wikibook: https://en.wikibooks.org/wiki/Prolog

Make sure your docs apply to gprolog, not swi-prolog

prolog visualizer: www.cdglabs.org/prolog/

What is a Declarative Programming Language?

say your high level goals, not how to achieve them

leave that part to the computer

Prolog

declare a database of facts/rules

make logical queries on this database

try facts/rules in order, and *unify* query with database

Example 1

```
person(alice).

person(bob).
```

Example 1 Queries

```
consult(‘example1.pl’).

person(alice).
yes

person(john).
no

person(X).
X = alice ? ;
X = bob
yes
```

Syntax

atoms

- e.g. `alice`, `john`
- uninterpreted constant

variables

- e.g. X
- uppercase identifiers

Syntax

predicates

- e.g. `person`
- relations that are true or false
- *arity* - number of arguments it takes

facts

- e.g. `person(alice).`, `person(bob).`
- assertions that these statements are true

Example 2

```
father(orville, abe).

father(abe, homer).

father(homer, bart).

father(homer, lisa).

father(homer, maggie).

grandfather(X, Y) :-

  father(X, Z),

  father(Z, Y).
```

Example 3 Queries

```
grandfather(abe, bart).

true ?
yes

grandfather(X, Y).

X = orville
Y = homer ? ;

X = abe
Y = bart ? ;

X = abe
Y = lisa ? ;

X = abe
Y = maggie ? ;

no
```

Syntax

rule

- e.g. `grandfather(X, Y) :- father(X, Z), father(Z, Y).`

- `conclusion :- hypotheses.`
  - `conclusion` is true if the hypotheses are true
- `conclusion :- h1, h2.`
  - conculsion is true if h1 and h2 are true

Valid

```
likes(john, susie).
```

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

```
likes(X, Y) :- likes(Y, X).
```

Invalid

```
likes(john, susie)
```

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

```
not(likes(X, Y)) :-
  hates(X, Y).
```

Syntax

and operator (‘ , ’)

or operator (‘ ; ’)

Example 3

Imagine we have a `mother` predicate that defines a relation between mothers and children as well as a `father` predicate defined earlier.

Write a `birthparents` predicate that has 3 variables, `Child`, `Mother`, and `Father`, and uses the mother and father predicate.

e.g. `birthparents(homer, marge, bart).`

Complex Terms

tuples with optional tags

- e.g. cons(E, L), (E, L)

Example 4

lists: `cons(a, cons(b, empty))`

```
head(cons(X, XS), X).
```

what about `tail`?

Example 5

```
length(cons(a, cons(b, empty)), X).
```

X = 2 ?

yes

Arithmetic

is

- e.g. `X is 1 + 5`
  - term `is` expression
  - evaluated one way (expr evaluated first, bound to term).

for more details on arithmetic operators, consult the manual

Some Other Useful Predicates

(=:=)/2 - arithmetic equal,

(=\=)/2 - arithmetic not equal,

(<)/2 - arithmetic less than,

(=<)/2 - arithmetic less than or equal to,

(>)/2 - arithmetic greater than,

(>=)/2 - arithmetic greater than or equal to

Some Other Useful Predicates

(=)/2 - Prolog unification

(\=)/2 - not Prolog unifiable

(==)/2 - term identical

(\==)/2 - term not identical

Lists

e.g. `[]`, `[a, b, c, 1, 2, 3]`

destructuring

- `[X|Rest]` analogous to `X::Rest`
- `[X, Y, Z|Rest]` analogous to `X::Y::Z::Rest`

Length

```
length([], 0).
length([_|R], L) :-
  length(R, RL),
  L is RL + 1.
```

Example 6

write the append predicate

```
append(L1, L2, LFinal) :- …


append([a,b,c], [1,2,3], [a,b,c,1,2,3]).
yes
```

Example 6

write the reverse predicate

```
reverse(L1, L2) :- …


reverse([a,b,c], [c,b,a]).
yes
```

Example 7

write the `member` predicate

```
member(X, L) :- …
```

```
member(a, [c,b,a]).
yes

member(a, [d,e,f]).
no
```

Example 7

write the `remove_first` predicate

```
remove_first(X, L1, L2) :- …
```

```
remove_first(c, [c,b,a], [c,b]).
yes

remove_first(a, [a,a,a], [a,a]).
yes
```

Example 7

write the `permutation` predicate

```
permutation(L1, L2) :- …


permutation([a,b,c], [c,b,a]).
yes
```

Example 8

write the compress predicate

```
compress(L1, L2) :- …


compress([a,a,a,a,b,c,c,a,a,d,e,e,e,e],[a,b,c,a,d,e]).
yes
```