Announcements

- We are almost done grading HW1
- HW2 due today
- Download SWI Prolog!
- HW3 (Prolog) will be posted today, due on February 29th

Last Class

- Bottom-up (LR) parsing
  - Characteristic Finite State Machine (CFSM)
  - SLR(1) parsing table
  - Conflicts in SLR(1)
  - LR parsing variants

Today’s Lecture Outline

- Logic Programming
- Logic Programming Concepts
  - Prolog
    - Language constructs: facts, queries, rules
    - Prolog concepts: search tree, unification, backtracking, backward chaining
    - Lists

Logic Programming

- Download and install SWI Prolog on laptop!

- J.R.Fisher’s Prolog Tutorial:
  http://www.cpp.edu/~jrfisher/www/prolog_tutorial/contents.html

Logic Programming and Prolog

Read: Scott, Chapter 11.1, 11.2.1-4

Logic Programming

- Logic programming
  - Logic programming is declarative programming
  - Logic program states what (logic), not how (control)
  - Programmer declares axioms
    - In Prolog, facts and rules
    - Programmer states a theorem, or a goal (the what)
    - In Prolog, a query
  - Language implementation determines how to use the axioms to prove the goal
Logic Programming

- Logic programming style is characterized by:
  - Database of facts and rules that represent logical relations. Computation is modeled as search (queries) over this database.
  - Manipulation of lists and use of recursion, in which it is very similar to the functional programming style.

Logic Programming Concepts

- **A Horn Clause is:**
  \[ H \leftarrow B_1, B_2, \ldots, B_n \]
  - **Antecedents** (\(B_i\)’s): conjunction of zero or more terms in predicate calculus; this is the body of the horn clause.
  - **Consequent** (\(H\)): a term in predicate calculus.

- **Meaning of a Horn clause:**
  - \(H\) is true if \(B_1, B_2, \ldots, B_n\) are all true.
  - When the antecedents \(B_i\) are all true, we deduce that consequent \(H\) is true as well.

- **For example:**
  - \(\text{rainy}(\text{seattle}).\)
  - \(\text{cold}(\text{rochester}).\)
  - \(\text{snowy}(\text{rochester}) \leftarrow \text{rainy}(\text{rochester}), \text{cold}(\text{rochester}).\)

Horn Clauses in Prolog

- **Clause is composed of terms:**
  - **Constants**
    - Number, e.g., 123, etc.
    - Atoms e.g., \text{seattle}, \text{rochester}, \text{rainy}, \text{foo}
  - **Variables**
    - \(X, \text{Foo, My_var}\)
  - **Structures** consists of an atom, called a functor and a list of arguments
    - \text{rainy}(\text{seattle}), \text{snowy}(X)

- **Variables may appear in the tail and head of a rule:**
  - \(c(X) \leftarrow h(X,Y)\).
    - For all values of \(X\), \(c(X)\) is true if there exist a value of \(Y\) such that \(h(X,Y)\) is true.
  - **Call \(Y\) an auxiliary variable. Its value will be bound to make consequent true, but not reported by Prolog, because it does not appear in the head.**
Lecture Outline

- Logic Programming
- Logic Programming Concepts
- Prolog
  - Language constructs: facts, queries, rules
  - Prolog concepts: search tree, unification, backtracking, backward chaining
- Lists

Prolog

- Program has a database of clauses i.e., facts and rules; the rules help derive more facts
- We add simple queries with constants, variables, conjunctions or disjunctions
  
  rainy(seattle).
  rainy(rockefeller).
  cold(rockefeller).
  snowy(X) :- rainy(X),cold(X).

  ? - rainy(C).
  ? - snowy(C).

Facts

likes(eve, pie).
likes(al, eve).
likes(eve, tom).
likes(eve, eve).
food(pie).
food(apple).

The combination of the functor and its arity (i.e., its number of arguments) is called a predicate.

Queries

likes(eve, pie).
likes(al, eve).
likes(eve, tom).
likes(eve, eve).
food(pie).
food(apple).

?-likes(al,eve).
true.

?-likes(al,Who).
Who=eve.

?-likes(eve,W), person(W).
W=tom.

?-likes(A,B).
A=eve,B=pie ; A=al,B=eve ; A=eve,B=tom ; A=eve,B=eve.

?-likes(D,D).
D=eve.

Prolog gives us the answer precisely in this order: first W=pie then W=tom and finally W=eve.

Can you guess why?

Question

?-likes(eve,W).
W = pie ;
W = tom ;
W = eve.

Harder Queries

?-likes(al,V) / \likes(eve,V).
V=evem.

?-likes(eve,W), person(W).
W=tom.

?-likes(A,B).
A=eve,B=pie ; A=al,B=eve ; A=eve,B=tom ; A=eve,B=eve.

?-likes(D,D).
D=eve.
Harder Queries

likes(eve, pie).
likes(al, eve).
likes(eve, tom).
likes(eve, eve).

?-likes(eve, W), likes(W, V).
W = eve, V = pie ; W = eve, V = tom ; W = eve, V = eve.

?-likes(eve, W), person(W), food(V).
W = tom, V = pie ; W = tom, V = apple

?-likes(eve, V), (person(V); food(V)).
V = pie ; V = tom

Rules

likes(eve, pie).
likes(al, eve).
likes(eve, tom).
likes(eve, eve).

?-rule1.

Add a rule to the database:

rule1 :- likes(eve, V), person(V).

?-rule1.

?-rule2(pie).
false.

Lecture Outline

Logic Programming
Logic Programming Concepts

Prolog
Language constructs: facts, queries, rules
Prolog concepts: search tree, unification, backtracking, backward chaining
Lists

Logical Semantics

- Prolog program consists of facts and rules
  rainy(seattle).
  rainy(rochester).
  cold(rochester).
  snowy(X) :- rainy(X), cold(X).

- Rules like snowy(X) :- rainy(X), cold(X).
  correspond to logical formulas
  \[ \forall X [\text{snowy}(X) \leftarrow \text{rainy}(X) \land \text{cold}(X)] \]

- For every X, X is snowy if X is rainy and X is cold.
Procedural Semantics

?- snowy(C)
snowy(X) :- rainy(X), cold(X).

**First**, find the first clause in the database whose head matches the query, in our case this is clause

```
snowy(X) :- rainy(X), cold(X)
```

**Then**, find a binding for X to make rainy(X) true; then, check if cold(X) is true with that binding

- If yes, report binding as successful
- Otherwise, backtrack to the binding of X, unbind and consider the next binding

Prolog’s computation is well-defined procedurally by search tree, rule ordering, unification and backtracking

**Question**

What does this query yield?

?- snowy(C).

**Answer:**

C = rochester ;
C = troy.

Prolog Concepts:

**Search Tree**

At OR levels Prolog performs unification
- Unifies parent (goal), with child (head-of-clause)
- E.g.,
  - snowy(C) = snowy(X)
    - success, \( C = X \)
  - rainy(X) = rainy(seattle)
    - success, \( X = seattle \)
  - parents(alice,H,F) = parents(edward,victoria,albert)
    - fail
  - parents(alice,H,F) = parents(alice,victoria,albert)
    - success, \( H = victoria, F = albert \)

**Unification**

- A constant unifies only with itself
  - E.g., alice=alice, but alice=edward fails
- Two structures unify if and only if (i) they have the same functor, (ii) they have the same number of arguments, and (iii) their arguments unify recursively
  - E.g., rainy(X) = rainy(seattle)
- A variable unifies with anything. If the other thing has a value, then variable is bound to that value. If the other thing is an unbound variable, then the two variables are associated and if either one gets a value, both do.

In Prolog, = denotes unification, not assignment!
Prolog Concepts: Backtracking
If at some point, a goal fails, Prolog backtracks to the last goal (i.e., last unification point) where there is an untried binding, undoes current binding and tries new binding (an alternative OR branch), etc.

```
snowy(X):-rainy(X),cold(X).
snowy(rochester). % Backward chaining: starts from goal, towards facts
?- snowy(rochester).
snowy(rochester):-rainy(rochester), cold(rochester)
snowy(rochester).
snowy(rochester):-cold(rochester)
snowy(rochester).
snowy(rochester). % Forward chaining: starts from facts towards goal
?- snowy(rochester).
snowy(rochester):-rainy(rochester), cold(rochester)
rainy(rochester)
snowy(rochester)
snowy(rochester). % Forward chaining: starts from facts towards goal
?- snowy(rochester).
snowy(rochester):-rainy(rochester), cold(rochester)
rainy(rochester)
snowy(rochester)
snowy(rochester). % Forward chaining: starts from facts towards goal
?- snowy(rochester).
snowy(rochester):-rainy(rochester), cold(rochester)
rainy(rochester)
snowy(rochester)
snowy(rochester).
```

Exercise
```
takes(jane, his).
takes(jane, cs).
takes(ajit, art).
takes(ajit, cs).
classmates(X,Y):-takes(X,Z),takes(Y,Z).
?- classmates(jane,C).
```

Running Prolog Programs
```
Enter the database of facts into a file. E.g., we can enter the classmates database into file classmates.pl
At the interpreter prompt, load the file then run queries. E.g.,
?- [classmates].
... true.
?- takes(ajit,C).
C = art ; C = cs.
```

Lecture Outline
- Logic Programming
- Logic programming concepts
- Prolog
  - Language constructs: facts, queries, rules
  - Prolog concepts: search tree, unification, backtracking, backward chaining
- Lists

Lists
```
lst head tail
[a,b,c] a [b,c] [c]
[X, [cat], Y] X [[cat], Y] [ ]
[a, [b,c], d] a [[b,c], d]
[X | Y] X Y a b c d [ ]
```

Prolog Concepts: Backward Chaining
- Backward chaining: starts from goal, towards facts
- Forward chaining: starts from facts towards goal
Lists: Unification

- \([ H1 | T1 ] = [ H2 | T2 ]\)
  - Head \(H1\) unifies with \(H2\), possibly recursively
  - Tail \(T1\) unifies with \(T2\), possibly recursively

- E.g., \([ a | [b, c] ] = [ X | Y ]\)
  - \(X = a\)
  - \(Y = [b, c]\)

- NOTE: In Prolog, \(=\) denotes unification, not assignment!

Question

- \([X,Y,Z] = [john, likes, fish]\)
  - \(X = john\), \(Y = likes\), \(Z = fish\)

- \([\text{cat}] = [X | Y]\)
  - \(X = \text{cat}\), \(Y = [ ]\)

- \([[\text{the}, Y]|Z] = [[X, hare]|\{\text{is, here}]\)
  - \(X = \text{the}\), \(Y = \text{hare}\), \(Z = \{\text{is, here}\}\)

Lists: Unification

- Sequence of comma separated terms, or
  - \([\text{first term} | \text{rest_of_list}]\)

\([\text{the} | Y] | Z] = [X, hare] | [\text{is, here}]\)

Lists Unification

look at the trees to see how this works!

\([a, b, c] = [X | Y]\)
- \(X = a\), \(Y = [b, c]\).

\([a | Z] =? [X | Y]\)
- \(X = a\), \(Y = Z\).

Improper and Proper Lists

\([1 | 2]\) versus \([1, 2]\)

Improper and Proper Lists

- \([abc, Y] =? [abc | Y]\)

What happens here?

Answer: No. There is no value binding for \(Y\), that makes these two trees isomorphic.
**Member_of “Procedure”**

?- member(a,[a,b]).
true.
?- member(a,[b,c]).
false.
?- member(X,[a,b,c]).
X = a ;
X = b ;
X = c ;
false

2. member(A, [B | C]) :- member (A, C).

**Example**

?- member(a,(b,c,X)).

2. member(A, [B | C]) :- member (A, C).

**Logical semantics:** For every value assignment of A, B and C, we have member(A, [B | C]) if member(A, C);
**procedural semantics:** Head of clause is procedure entry. Procedure body consists of calls within this procedure.

**Append “Procedure”**

append([], A, A).
append([A|B], C, [A|D]) :- append(B,C,D).

- Build a list
  ?- append([], [b], Y).
  Y = [ a,b ]
- Break a list into constituent parts
  ?- append(X,[b],[a,b]).
  X = [ a ]
  ?- append([a],Y,[a,b]).
  Y = [ b ]
More Append

?- append(X,Y,[a,b]).
X = [ ]
Y = [a,b] ;
X = [a]
Y = [b] ;
X = [a,b]
Y = [ ] ;
false.

More Append

Generating an unbounded number of lists
?- append(X,[b],Y).
X = [ ]
Y = [b] ;
X = [ _169]
Y = [ _169, b] ;
X = [ _169, _170 ]
Y = [ _169, _170, b] ;
etc.

Group Exercise

Create a file lists.pl, add member and append to it
1. member(A, [A | B]).
2. member(A, [B | C]) :- member(A, C).

Write a few queries with member and append

Now add another “procedure” brackets which encloses each element in brackets
e.g., brackets([a,[b],c],R).
returns R = [[a],[[b]],[c]]