Rule Ordering and Unification

1. rule ordering used in search
2. unification requires two instances of the same variable in the same rule to get the same value
3. unification does not require differently named variables to get different values: hence, sibling(edward,edward).
4. all rules searched if requested by ';'

More on unification in a lecture to come....

How Prolog Handles a Query

Example 1

Database:

```
 1)  male(tom).
 2)  male(peter).
 3)  male(doug).
 4)  female(susan).
 5)  male(david).

 6)  parent(doug, susan).
 7)  parent(tom, william).
 8)  parent(doug, david).
 9)  parent(doug, tom).

10)  grandfather(GP, GC) :- male(GP),
                            parent(GP, X) ,
                            parent(X, GC).
```

Query:

```
?- grandfather(X,Y).
```

Trace it by hand

Trace it in Prolog

```
[trace]  ?- grandfather(X,Y).
   Call: (7) grandfather(_G283, _G284) ? creep
   Call: (8) male(_G283) ? creep
   Exit: (8) male(tom) ? creep
   Call: (8) parent(tom, _L205) ? creep
   Exit: (8) parent(tom, william) ? creep
   Call: (8) parent(william, _G284) ? creep
   Fail: (8) parent(william, _G284) ? creep
   Redo: (8) male(_G283) ? creep
   Exit: (8) male(peter) ? creep
   Call: (8) parent(peter, _L205) ? creep
   Fail: (8) parent(peter, _L205) ? creep
   Redo: (8) male(_G283) ? creep
   Exit: (8) male(doug) ? creep
   Call: (8) parent(doug, _L205) ? creep
   Exit: (8) parent(doug, susan) ? creep
   Call: (8) parent(susan, _G284) ? creep
   Fail: (8) parent(susan, _G284) ? creep
   Redo: (8) parent(doug, _L205) ? creep
   Exit: (8) parent(doug, david) ? creep
   Call: (8) parent(david, _G284) ? creep
   Fail: (8) parent(david, _G284) ? creep
   Redo: (8) parent(doug, _L205) ? creep
   Exit: (8) parent(doug, tom) ? creep
   Call: (8) parent(tom, _G284) ? creep
   Exit: (8) parent(tom, william) ? creep
   Exit: (7) grandfather(doug, william) ? creep
X = doug
Y = william
Yes
```

Prolog Search Trees

- Each **node** is an ordered list of goals.
- Each **edge** is labelled with the variable bindings that occurred due to applying a rule. (The binding are in effect throughout the subtree.)
- Each **leaf** represents either success or failure.

Example 2

Database:

```
1)  male(albert).
2)  female(alice).
3)  male(edward).
4)  female(victoria).
5)  parent(albert,edward).
6)  parent(victoria,edward).
7)  parent(albert,alice).
8)  parent(victoria,alice).
9)  sibling(X, Y) :- parent(P, X), parent(P, Y).
```

Query:

```
?- sibling(alice,Asib).
Asib = edward ;
Asib = alice ;
Asib = edward;
Asib = alice ;
No
?- sibling(Asib, alice).
Asib = edward ;
Asib = edward ;
Asib = alice ;
Asib = alice ;
No
```

Trace it by hand

Trace it in Prolog

```
[trace]  ?- sibling(alice,Asib).
   Call: (7) sibling(alice, _G284) ? creep
   Call: (8) parent(_L205, alice) ? creep
   Exit: (8) parent(albert, alice) ? creep
   Call: (8) parent(albert, _G284) ? creep
   Exit: (8) parent(albert, edward) ? creep
   Exit: (7) sibling(alice, edward) ? creep

Asib = edward ;
   Redo: (8) parent(albert, _G284) ? creep
   Exit: (8) parent(albert, alice) ? creep
   Exit: (7) sibling(alice, alice) ? creep

Asib = alice ;
   Redo: (8) parent(_L205, alice) ? creep
   Exit: (8) parent(victoria, alice) ? creep
   Call: (8) parent(victoria, _G284) ? creep
   Exit: (8) parent(victoria, edward) ? creep
   Exit: (7) sibling(alice, edward) ? creep

Asib = edward ;
   Redo: (8) parent(victoria, _G284) ? creep
   Exit: (8) parent(victoria, alice) ? creep
   Exit: (7) sibling(alice, alice) ? creep

Asib = alice ;

No
```

The Anonymous Variable

If a rule has a variable that appears only once, that variable is called a "singleton variable".

Its value doesn't matter — it doesn't have to match anything elsewhere in the rule.

```
isaMother(X) :- female(X), parent(X, Y).
```

Such a variable consumes resources at run time.

We can replace it with "_", the anonymous variable. It matches anything.

If we don't, Prolog will warn us.

Procedural Semantics of Prolog

Notice the recursion in this algorithm: "find" calls "find". This reasoning is recursively applied until we reach rules that are facts.

This process is called **Backward Chaining**.

Logic Programming vs. Prolog

```
cousin(X,Y) :- parent(W,X), sister(W,Z),
               parent(Z,Y).

cousin(X,Y) :- parent(W,X), brother(W,Z),
               parent(Z,Y).

?- cousin(X,jane). % a query
```

Rule and Goal Ordering:

- There are two rules for cousin
- Which rule do we try first?
- Each rule for cousin has several subgoals
- Which subgoal do we try first?

Logic Programming vs. Prolog

Logic Programming: *Nondeterministic*

- Arbitrarily choose rule to expand first
- Arbitrarily choose subgoal to to explore first
- Results don't depend on rule and subgoal ordering

Prolog: *Deterministic*

- Expand first rule first
- Explore first subgoal first
- Results may depend on rule and subgoal ordering

Let's Write Some Code

Write these predicates:

1. uncle
2. aunt
3. nephew
4. niece
5. grandparent
6. ancestor

Code we develop in class will be posted on the Web site as text so that you can load and run it.