Today

- Prolog Semantics

Assignments

- HW-7, R-7 due
- HW-8, R-8 out
- Proj 2 out
Non-contiguous clauses warning

- SWI generally assumes your clauses are grouped together

- For example:

  mother(a,b).
  mother(a,c).
  father(d,b).

- instead of:

  mother(a,b).
  father(d,b).
  mother(a,c).

- the last one will cause a warning in SWI:

  ?- [family]
  Warning: .../family.pl:4:
  Clauses of mother/2 are not together in the source-file

- But you can just ignore this ...

- You can also tell prolog to ignore ...

  :- discontiguous
     mother/2,
     father/2.

  - a rule that is always “executed”
  - uses the “functor/arity” syntax
A **fact** is an assertion

```prolog
mother(necessity,invention).
```

- we’re asserting “*necessity is the mother of invention*” as true
- if no variables, a “**ground clause**”

A **rule** can create new facts from existing facts

```prolog
parent(X,Y) :- mother(X,Y).
human_parent(X,Y) :- parent(X,Y), human(X).
```

- The left-side of `:-` is called the rule **Head**
  - can contain only one relation
- The right-side of `:-` is called the rule **Body**
  - can contain multiple relations conjoined (ANDed) together
- A rule is also a **clause**, but with a head and body
Prolog Programming Semantics (Model of Computation)

Consider the following rule:

\[ p(X,Z) :\neg q(X,Y), r(Y,Z). \]

Two possible ways to “view” (conceptualize) a prolog program:

- **“Bottom-up”** view:
  - read rules from right to left (from current facts to derived facts)
  - “if X is q-related to Y and Y is q-related to Z, then X is p-related to Z”
  - this is the “database” perspective (like in SQL/Datalog): always deriving new facts from previous ones
  - more on this later ...

- **“Top-down”** view:
  - read rules from left to right
  - e.g., “given X, compute Z by computing Y from X via q, and then computing Z from Y via R”
  - the more standard prolog perspective (think function definitions in Haskell)

Prolog is a full-fledged programming language

1. arithmetic operations, I/O, etc.
2. sequencing via conjunction (in above example do \( q(X,Y) \) followed by \( r(Y,Z) \))
3. if-then through rules
4. looping through recursion
5. plus explicit support for lists
How it works (informal) ...

How does Prolog find answers to parent(X,Y)?

?- parent(X,Y).
  => mother(X,Y)       "goal"
  => mother(necessity,invention) "unfold" 1st parent rule
      X = necessity, Y = invention ; "unify"
  => mother(june,wally) "backtrack"
      X = june, Y = wally ; "unify"
      no more matches "backtrack"
  => father(X,Y) "unfold" 2nd parent rule
  ...

This is somewhat similar to (but not exactly) Haskell pattern matching!
What is “unfolding”? 

When we “unwrap” we apply a form of substitution

RULE 1: \( E \leftarrow C \land D \)
RULE 2: \( C \leftarrow A \land B \)

- We can substitute \( C \) in RULE 1 with \( A, B \) from RULE 2
  
RULE 1’: \( E \leftarrow A \land B \land D \)

Based on the resolution inference rule:

- For example:

\[
\begin{align*}
E & \lor \neg C \lor \neg D \\
C & \lor \neg A \lor \neg B \\
\hline
E & \lor \neg A \lor \neg B \lor \neg D
\end{align*}
\]

Resolution is trickier with variables (predicates) ...

- Unfolding/resolution requires variable “unification”

RULE 1: \( E(x) \leftarrow C(x), D(x) \)
RULE 2: \( C(y) \leftarrow A(y), B(y, z) \)

- We must unify \( x \) in RULE 1 with \( y \), and rename to \( x \)

RULE 1’: \( E(x) \leftarrow A(x), B(x, z), D(x) \)
Unification rules

- A **constant** unifies with itself
- Two **structures** unify if
  - they have the same functor
  - they have the same arity
  - and the corresponding arguments unify
- A **variable** unifies with any other term
  - if the term is a constant, the variable is **instantiated**
  - if the term is a variable, then they are associated (linked)
  - if the term is a structure, the variable is instantiated to it

Examples

?- a = a. The = symbol means "unify"
true.

?- X = a. If RHS and LHS can unify
true.

?- X = Y, Y = a. then unify and return true
X = a,
Y = a.

?- r(a, b) = r(X, Y).
?- r(a,b) = r(X, X).
false.

?- r(X, s(a, b)) = r(Y, Z).
X = Y,
Z = s(a, b).
“Proof Tree” Example ...

Another simple knowledge base

\[
\text{food}(X) :\text{- edible}(X), \text{nutritious}(X).
\text{edible}(\text{twinkie}).
\text{edible}(\text{apple}).
\text{nutritious}(\text{apple}).
\]

Search for query answers in Prolog using “Backtracking”

To find more answers, keep backtracking

?- \text{food}(F) \quad \text{Our goal}

\[ F = X \]

\[ \text{food}(X) \quad \text{Candidate clause} \]

\[ \text{edible}(X), \text{nutritious}(X) \quad \text{subgoals} \]

\[ X = \text{twinkie} \quad \text{Candidate clause} \]

\[ \text{edible}(\text{twinkie}) \]

\[ \text{false} \quad \text{backtrack} \]

\[ X = \text{apple} \quad \text{Candidate clause} \]

\[ \text{edible}(\text{apple}) \]

\[ \text{nutritious}(\text{apple}) \quad \text{Candidate clause} \]

\[ \text{Found an answer!} \]

S. Bowers 9 of 10