Today

- Prolog semantics (model of computation)

Assignments

- HW-11 due
- HW-12 out (due Thurs)

Announcements

- Last Quiz on Tuesday!
- Exam 3 on Thursday
What is “unfolding”?  

When we “unfold” 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:

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

\( C \) and \( \neg C \) “cancel”

Resolution is trickier with variables (predicates) …

- Unfolding/resolve
tion 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 ... recursive
- 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.
X = a,
Y = a.

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

food(X) :- edible(X), nutritious(X).
edible(twinkie).
edible(apple).
nutritious(apple).

Search for query answers in Prolog using “Backtracking”

To find more answers, keep backtracking

?- food(F)  Our goal
  F = X
  food(X)  Candidate clause

subgoals

edible(X), nutritious(X)

X = twinkie

edible(twinkie)  backtrack?

false

Found an answer!

X = apple

nutritious(apple)  Candidate clause

X = apple

edible(apple)  Candidate clause

Candidate clause

Candidate clause
Proof trees

A **proof tree** is a way to view query evaluation (Prolog computation)

- specifies a “trace” of a Prolog program
- follows evaluation order (with unification and resolution)

The KB:

1. r(a).
2. s(X) :- r(X).

![Proof tree diagram]

- List of goals to “solve”
- Result of resolution step
- One child per program clause
- Finished with clauses to solve (success!)
Proof tree definition

A proof tree ...

- Contains **nothing** and **solve** nodes
- Each nothing node is a leaf (no children)
- Each solve node contains a list of terms:
  - if empty list, the solve node is a leaf
  - otherwise the solve node has one child per program clause (in order)
  - if a clause does not unify with the head of the query list, the child is a nothing node
  - if it does unify, the query is replaced (unfolded) in the list using resolution
- The root is a solve node with the initial query terms (goal)
Another example

The KB:

1. \( p(X) :- q(X), r(X) \).
2. \( q(a) \).
3. \( q(b) \).
4. \( r(a) \).

\(?- p(X) \)."

The proof tree:

- Note we often put bindings on the edges (e.g., \( X = a \))
- Prolog evaluation performs a \textbf{depth-first left-to-right} search
And another example

The KB:

1. p(X) :- p(X), q(X).
2. q(a).
3. p(a).

?- p(X).

Q: See a potential problem?

The proof tree:

- Prolog evaluates depth-first, left-to-right ...
- So this branch will be followed infinitely, without any proofs

- Creates an **infinite** proof tree!
And one last one

The KB:

1. p(a).
2. q(a).
3. p(X) :- p(X), q(X).

?- p(X).

The proof tree:

- Also creates an **infinite** proof tree ... but generates solutions!
  - What is wrong with the recursion? ... doesn’t make “progress”
 Variable naming and proof trees

Be careful with variable names

\[ r(X,Y) : r(X,Z), s(Z,Y). \]

\[
\begin{align*}
\text{solve} [r(X,Y)] \\
\text{solve} [r(X,Z), s(Z,Y)] \\
\text{solve} [r(X,Z1), s(Z1,Z), s(Z,Y)]
\end{align*}
\]

- We cannot use Z here again!
- So we rename to Z1 ...
More examples:

- “Joins”

  \[
  r(a,b).
  s(b,c).
  q(X,Z) :- r(X,Y), s(Y,Z).
  \]

  \texttt{?- q(a,Y).}

- Transitive closure

  \[
  e(a,b).
  e(b,c).
  tc(X,Y) :- e(X,Y).
  tc(X,Y) :- e(X,Z), tc(Z,Y).
  \]

  \texttt{?- tc(a,Y).} \quad \% \text{what is reachable from } a?
**Negation**

Use `\+` for negation of a term ...

\[
p(X) :- q(X), \+ r(X). \quad \% \text{p if q and not r}
\]

- requires negated term to be completely “ground” (no uninstantiated vars)

A term is “false” if it **fails** (i.e., cannot be proved/solved)

- known as “negation as failure”

Useful for doing “forall” rules

- For example: Does this rule define the notion of an “only child”?

\[
\text{only_child}(X) :- \text{parent}(P,X), \text{parent}(P,Y), X==Y.
\]

- No! ... \( X \) and \( Y \) could be the same (recall proof tree)

- We want to say for all possible \( Y \), \( X==Y \).
- Or equivalently, there isn’t a \( Y \) different from \( X \)

\[
i.e., \, \neg \exists y(y \neq x) \equiv \forall y \neg(y \neq x) \equiv \forall y(y = x)
\]

\[
\text{only_child}(X) :- \text{parent}(P,X), \+ \text{has_sibling}(X).
\text{has_sibling}(X) :- \text{parent}(P,X), \text{parent}(P,Y), Y \neq X.
\]
Arithmetic Expressions

Prolog supports standard arithmetic operators

• But beware the definition of = (unifies)

```prolog
?- X = 2 + 3.
X = 2+3

?- X = 2 * 3 + 4.
X = 2*3+4
```

• These are not being evaluated!
  – really, we are unifying X with relations (structures)
  – e.g., we end up with X = +(2,3) in the first case

• Use “is” to evaluate expressions ...

```prolog
?- X is 2 + 3.
X = 5

?- X is 2 * 3 + 4.
X = 10
```

• Real and integer division

```prolog
?- X is 7/2.  % real division
X = 3.5

?- X is 7//2.  % integer division
X = 3

?- X is 7 mod 2.
X = 1
```
Examples of arithmetic in rules

% add one to X and "return" in R
add1(X,R) :- R is X + 1.

% add X and Y and "return" in R
add(X,Y,R) :- Z is X + Y.

% average X, Y, and Z and "return" in R
avg3(X,Y,Z,R) :- R is (X + Y + Z) / 3.