Logic Programming

Top-Down Execution

Michael Genesereth
Computer Science Department
Stanford University
Bottom-Up Evaluation
   Starts with data
   Applies rules to produce new data
   (within strata or working up the stratum hierarchy)
   Until no new results can be added

Top-Down Evaluation
   Starts with query to be answered
   Applies rules to reduce to subqueries
   Continues until reaches data level
Given a query, a dataset, and a ruleset, do the following.

(1) If the predicate in the query is a base predicate, succeed if and only if query is in dataset.

(2) If the query is a negative literal, evaluate target and succeed if and only if fail to prove.

(3) If the query is a conjunction of literals, succeed iff succeed on all conjuncts.

(4) If the predicate in the query is a view predicate, evaluate the body of each rule defining that predicate and succeed if and only if succeed on at least one rule.
Facts
- $p(a)$
- $p(b)$
- $p(c)$

Rules
- $s(c) :- p(a) \& q(b)$
- $s(c) :- p(b) \& r(c)$
- $s(c) :- p(c) \& \neg q(c)$
- $r(c) :- p(a) \& p(d)$

Top Down Evaluation

```
                s(c)?
               /    \
              /      \ 
         p(a) & q(b)?  p(b) & r(a)?  p(c) & ~q(c)?
               \    /        \            /
      p(a) & p(d)?
```
Procedure without variables uses *equality* tests.

\[
\begin{align*}
p(a,b) \\
p(b,c) \\
\text{query}(a,c) & : = p(a,b) \land p(b,c) \\
\end{align*}
\]

query(a,c)?

Procedure with variables uses *unification*.

\[
\begin{align*}
p(a,b) \\
p(b,c) \\
\text{query}(X,Z) & : = p(X,Y) \land p(Y,Z) \\
\end{align*}
\]

query(a,c)?

*Details of unification tedious. Read in notes.*
**Facts**

- $p(a, b)$
- $p(b, c)$

**Rules**

- $s(X, Z) :\text{:-} p(X, Y) \land p(Y, Z)$

**Top Down Evaluation**

1. $s(X, Z)$?
2. $p(X, Y) \land p(Y, Z)$?
3. $X=a, Y=b$
   - $p(b, Z)$?
4. $X=b, Y=c$
   - $p(c, Z)$?
Facts and Rules

\[ p(a, b) \]
\[ p(b, c) \]
\[ s(X, Z) := p(X, Y) \land p(Y, Z) \]

Trace

Call: \( s(X, Z) \)
  | Call: \( p(X, Y) \)
  | Exit: \( p(a, b) \)
  | Call: \( p(b, Z) \)
  | Exit: \( p(b, c) \)
Exit: \( s(a, c) \)
Facts and Rules

\[
p(a,b) \\
p(b,c) \\
s(X,Z) :\quad p(X,Y) \land p(Y,Z)
\]

Trace

Call: \(s(X,Z)\) \quad Redo: \(s(X,Z)\)
\[
\quad \mid \quad \text{Call: } p(X,Y) \quad \mid \quad \text{Redo: } p(b,Z) \\
\quad \mid \quad \text{Exit: } p(a,b) \quad \mid \quad \text{Fail: } p(b,Z) \\
\quad \mid \quad \text{Call: } p(b,Z) \quad \mid \quad \text{Redo: } p(X,Y) \\
\quad \mid \quad \text{Exit: } p(b,c) \quad \mid \quad \text{Exit: } p(b,c) \\
\quad \text{Exit: } s(a,c) \quad \mid \quad \text{Call: } p(c,Z) \\
\quad \mid \quad \text{Fail: } p(c,Z) \\
\quad \text{Fail: } s(X,Z)
\]
Comparison of Evaluation Strategies

**Bottom-Up Evaluation**
- Easy to understand
- Computes all results
- Computes subresults just once
- Wasteful when want just one or a few answers, not all
- Problematic on logic programs with infinite models

**Top-Down Evaluation**
- Less waste when want one or a few answers
- More complicated
- Subqueries evaluated multiple times
- Possibility of infinite loops on programs with finite models
Bottom-Up Evaluation
   Can be focussed using Magic Sets

Top-Down Evaluation
   Top-Down can avoid duplication through caching
   Infinite Loops can be avoided using iterative deepening

*The arms race continues.*