CS 603: Programming Languages

Lecture 32

Spring 2004

Department of Computer Science
University of Alabama
Joel Jones

Overview

- Questions on MP?
- Prolog Semantics continued (starting at 10.2.3 of text)
  - Logical Interpretation
  - Procedural Interpretation

Logical: How make deterministic?

- *Which* clause $C \sqcup D$ do we choose when applying rule **LogicalQuery**?
- Given g and G, how do we discover a pair of substitutions that cause a match ... ?
- Unification

Logical: Definitions for Unification

- Definition 10.3: A substitution θ_1 is *more general* than a substitution θ_2 if there exists a θ_3 such that $\theta_2 = \theta_3 \circ \theta_1$. The more general a substitution is, the fewer things it changes.
- Definition 10.4: *Unification* is the process of finding, for given goals g_1 and g_2, a substitution θ that unifies g_1 and g_2. Furthermore, θ must be a *most general* substitution.
- Definition 10.5: A *renaming of variables* is a substitution θ_ρ in which θ_ρ (VAR(*X*)) is always a variable, never an application or an integer.

Examples of Unification (rule 10.4)

g_1 = member(3, [3|nil])

g_2 = member(X, [X|L])

θ = {X ⟼ 3, L ⟼ nil}

g_1 = member(Y, [3|nil])

g_2 = member(X, [X|L])

θ = {X ⟼ 3, Y ⟼ 3, L ⟼ nil}

~~g_1 = member(3, [4|nil])~~

~~g_2 = member(X, [X|L])~~

~~g_1 = length([3|nil], N)~~

~~g_2 = member(X, [X|L])~~

~~g_1 = member(X, [X|L])~~

~~g_2 = member(Y, cons(mkTree(1, nil, nil), M))~~

does not unify, as

$\hat{\theta}(g_1)$ = member(e, [e|L])

$\hat{\theta}(g_2)$ = member(e, cons(mkTree(e, nil, nil), M))

and e can never equal mkTree(e, nil, nil)

Motivation for Renaming (rule 10.5)

Given:

g = `member(M, [1|nil])`

G = `member(X, [X|M])`

Can these be unified?

No.

Simple unification doesn't work , as there is no single θ such that $\hat{\theta}(G) = \hat{\theta}(g)$, since no single substitution can simultaneously satisfy `M = X, X = 1,` and `M = nil`

What is the problem?

`M` appears in both G and g, but we should be treating these two instances of `M` differently. If there were no variables in common, then unification would be easy.

So, rename, then unify.

Accordingly, we change **LogicalQuery** to reflect this renaming, then unification

Procedural Semantics, Version 1

We are given database $D = C_1, ..., C_n$ and query g, and wish to known whether g is satisfied. i.e. $D \vdash g$. Search clauses in order; the first time we find a clause with a left hand side matching g, say g :- $H_1, ..., H_m$, we attempt to satisfy $H_1, ..., H_m$, in that order, following the same procedure for each H_j

This only works with *ground* clauses and goals, i.e. those without variables.

Examples of Procedural Semantics, Version 1

- See 451a for examples that work
  - both version 1 and logical
- See 451b for examples that don't work
  - under version 1, but do under logical

What's missing?

Backtracking.
There may be more than one clause that applies to a given goal, and not all applicable clauses lead to a solution.

Procedural Semantics, Version 2

Search clauses in order; the first time we find a clause C_i with a left hand side matching g, say g :- H_1, ..., H_m, we attempt to satisfy H_1, ..., H_m, in that order, following the same procedure for each H_j. If all are satisfied, g has been satisfied. Otherwise, go back to the list of clauses, starting with C_{i+1}, and look for another clause whose left-hand side equals g.

This only works with *ground* clauses and goals, i.e. those without variables.

Examples of Procedural Semantics, Version 1

- See 452a for examples that work
  - both version 2 and logical, but not version 1
- See 452b for examples that don't work
  - under version 1 or 2, but do under logical

What's the problem?

Infinite loop.
But real Prolog does this too. This is one reason why Logic Programming is not quite Logic.

What else is missing?

Variables

Procedural Semantics, Final Version

- See page 453 of textbook