

# CS 603: Programming Languages

Lecture 26

Spring 2005

Department of Computer Science

University of Alabama

Joel Jones

# Overview

- Prolog Semantics (starting at 10.2.3 of text)
  - Logical Interpretation
  - Procedural Interpretation

# Semantics: Logical Interpretation

- Prolog program: list of clauses  $C_1, \dots, C_n$  and goal  $g$ , where each clause has the form  $G :- H_1, \dots, H_n$
- Prolog logical interpretation: prove right hand side then left hand side is true, finding variables that satisfy the constraints
- In other words, to satisfy the query  $g$ , find values for the variables occurring in  $g$  such that it can be proven using the clauses as inference rules

# Example of Logical Interpretation

$\text{member}(X, [Y|M]) := \text{member}(X, M)$ .

represents the assertion: for any values  $X$ ,  $Y$ , and  $M$ , if  $X$  occurs in  $M$ , then  $X$  occurs in  $[Y|M]$ . Thus from

$\text{member}(3, [4, 3])$

we can infer

$\text{member}(3, [7, 4, 3])$ .

Similarly,

$\text{member}(X, [X|L])$ .

states that  $\text{member}(X, [X|L])$  is true no matter what the values of  $X$  and  $L$

# Logical Interpretation (cont.)

- But this is very precise, as we haven't really pinned down what is going on with variables.
- Our goal is a formulation of logical inference rules, which happen to be nondeterministic.

# Logical: Definitions

- Theorem 10.1: A *substitution*  $\theta$  is a function  $\theta$  from terms to terms that satisfies...
  - Substitution must be finite
  - Expressed as set of bindings
  - If  $\theta_1$  and  $\theta_2$  are substitutions, so is  $\theta_1 \circ \theta_2$
- Definition 10.2: The application of a substitution to a goal is the goal obtained by applying the substitution to each argument of the goal

# Logical: Precise Inference Rules

- Goal  $g$  is satisfiable using database  $D$  and substitution  $\theta^n$  as the judgment ...
- In the general case, a query can have more than one goal, so ...
- In the logical interpretation, satisfaction of different goals is independent, but requires that the same substitution satisfy them all
- See **LOGICALQUERIES** on p. 447

# Logical: Precise Inference Rules (cont.)

- A substitution  $\theta$  satisfies a query  $g$  if there is some clause in the database such that another substitution makes the conclusion the same as  $g$ , and under that substitution we can prove all the premises
- See **LOGICALQUERY** on p. 447
- Summarizing, the logical interpretation says that to satisfy query  $g$ , there must be a pair of substitutions  $\theta$  and  $\theta'$  such that  $\theta$  applied to the original query is the same as  $\theta'$  applied to the head of some clause, and the subgoals produced by applying  $\theta'$  to the subgoals of that clause are all satisfiable.

# Logical: How make deterministic?

- *Which* clause  $C \in D$  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  $\theta_1$  is *more general* than a substitution  $\theta_2$  if there exists a  $\theta_3$  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  $\theta$  that unifies  $g_1$  and  $g_2$ . Furthermore,  $\theta$  must be a *most general* substitution.
- Definition 10.5: A *renaming of variables* is a substitution  $\theta_\alpha$  in which  $\theta_\alpha(\text{VAR}(X))$  is always a variable, never an application or an integer.

# Examples of Unification (rule 10.4)

$g_1 = \text{member}(3, [3 | \text{nil}])$

$g_2 = \text{member}(X, [X | L])$

$\theta = \{X \mapsto 3, L \mapsto \text{nil}\}$

$g_1 = \text{member}(Y, [3 | \text{nil}])$

$g_2 = \text{member}(X, [X | L])$

$\theta = \{X \mapsto 3, Y \mapsto 3, L \mapsto \text{nil}\}$

~~$g_1 = \text{member}(3, [4 | \text{nil}])$~~

~~$g_2 = \text{member}(X, [X | L])$~~

~~$g_1 = \text{length}([3 | \text{nil}], N)$~~

~~$g_2 = \text{member}(X, [X | L])$~~

~~$g_1 = \text{member}(X, [X | L])$~~

~~$g_2 = \text{member}(Y, \text{cons}(\text{mkTree}(1, \text{nil}, \text{nil}), M))$~~

does not unify, as

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

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

and  $e$  can never equal  $\text{mkTree}(e, \text{nil}, \text{nil})$

# Motivation for Renaming (rule 10.5)

Given:

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

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

Can these be unified?

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.

No.

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

So, rename, then unify.

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

# Procedural Semantics, Version I

We are given database  $D = C_1, \dots, C_n$  and query  $g$ , and wish to know 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  $g :- H_1, \dots, H_m$ , we attempt to satisfy  $H_1, \dots, H_m$ , 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 45 I a for examples that work
  - both version 1 and logical
- See 45 I b 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  $g :- H_1, \dots, H_m$ , we attempt to satisfy  $H_1, \dots, H_m$ , 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