

**CS 401/501 PROGRAMMING LANGUAGES**  
**TOPICS TO BE COVERED ON EXAM #2**  
**March 19, 2009**

The exam will cover *everything* we have discussed from Chapters 3 (i.e., Section 3.5), 15, and 16 in the text and all notes, but emphasizing the material covered in class. The style of the exam will be very flexible, possibly consisting of fill in the blank, true or false (possibly with justification), multiple choice, matching, short answer (e.g., definitions or listing), and discussion questions. You will not be asked to work detailed problems such as writing any actual code (i.e., code in a specific language like Lisp or Prolog) You should have some ideas about the theory behind implementation of programming languages (e.g., the relationship between formal models of computation and programming languages,  $\lambda$ -calculus etc.). Some smaller problems such as drawing a Lisp representation of a list, or tracing the execution of a Prolog program, are likely.

A brief outline of the topics we have covered is described below. (This list is intended to be as complete as possible but may not be all inclusive.)

- *Programming Languages - Semantics.* Of the three formal methods, operational, denotational and axiomatic, we concentrated on denotational. You should know the goals of formal semantics and should have specific ideas about how denotational semantics may model interpretation of programs. You need not remember any specific denotational semantics equations per se but should know enough about them to be able to explain or use them if they are given to you.
- *Functional Programming Languages.* Lambda calculus was introduced as the foundation of denotational semantics and functional programming. The LISP, ML and Haskell programming languages were discussed as examples of the functional language paradigm. You should know the general design philosophy and syntactic and semantic flavor. You do NOT need to know specific commands in detail, but should know some basic ones from LISP (e.g., `defun`, `cond`, `eq`, `setq`, `cons`, `car`, `cdr`, `append`, `list`). Examples of the most detailed information of a fundamental nature you should know would be how functions are defined, how lists are represented internally, and how functions may be manipulated as data.
- *Logic Programming.* You should know the general design philosophy and syntactic and semantic flavor of Prolog. You do NOT need to know specific commands. Examples of the most detailed information of a fundamental nature you should know would be how the rule matching works (e.g., execution of rules, instantiation of variables, backtracking, etc.) and the representation of lists.

## CS 401/501 EXAM #2 SAMPLE QUESTIONS

1. The mathematical model of computation that denotational semantics is based upon is called  
-----.
2. True or False. Denotational semantics is a method for defining the semantics of a programming language by giving an interpreter for that language.
3. Consider the following Lisp function:

```
(defun update (store V n) (prog (currentenv newbinding newstore)
  (setq currentenv (cdaddr store)))
  (setq newbinding (cons (list 'eq 'V (list 'quote V)) (list n)))
  (setq newstore (list 'lambda (list 'V) (cons 'cond (cons newbinding currentenv))))
  (return newstore))))
```

Note that the `(prog (list of local variables) (exp-1) (exp-2) ... (exp-n) (return exp))` construction declares a list of local variables used in the expressions with `exp` being the value returned. Let `S` represent the Lisp expression `(lambda (V) (cond ((eq V (quote x)) 4) (t (quote bottom))))`. What is the internal list representation that Lisp uses for `S`? What is `(cdaddr S)`? Trace the execution of `(update S 'x 5)`, showing the values of `currentenv`, `newbinding`, and `newstore`.

4. Consider the Prolog program which solves the Towers of Hanoi problem. This problem is to move a stack of blocks from one location to another without ever changing the order in which blocks are placed on top of each other. A third stack may be used but only one block may be moved at a time.

```
hanoi(N) :- move(N, left, center, right).
```

```
move(0, X, Y, Z).
```

```
move(N, X, Y, Z) :-
```

```
  N > 0, M is N - 1, move(M, X, Z, Y), display([X,Y]), move(M, Z, Y, X).
```

Trace the above program using the query `hanoi(3)`. Show all queries which are generated by the execution and all moves which are printed. You may wish to draw a tree to show this clearly.

5. Consider the denotational semantics below:

$$S[\mathbf{S}_1; \mathbf{S}_2] \text{ store} = S[\mathbf{S}_2] (S[\mathbf{S}_1] \text{ store})$$

$$S[\mathbf{V} := \mathbf{E}] \text{ store} = \text{store}[E[\mathbf{E}] \text{ store} / \mathbf{V}]$$

$$S[\mathbf{while} \ \mathbf{C} \ \mathbf{loop} \ \mathbf{S} \ \mathbf{end} \ \mathbf{loop}] \text{ store} = \\ \text{if } C[\mathbf{C}] \text{ store} \text{ then } S[\mathbf{while} \ \mathbf{C} \ \mathbf{loop} \ \mathbf{S} \ \mathbf{end} \ \mathbf{loop}] (S[\mathbf{S}] \text{ store}) \text{ else } \text{store}$$

$$C[\mathbf{E}_1 > \mathbf{E}_2] \text{ store} = \text{if } E[\mathbf{E}_1] \text{ store} > E[\mathbf{E}_2] \text{ store} \text{ then } \text{true} \text{ else } \text{false}$$

$$E[\mathbf{E}_1 + \mathbf{E}_2] \text{ store} = E[\mathbf{E}_1] \text{ store} + E[\mathbf{E}_2] \text{ store}$$

$$E[\mathbf{E}_1 - \mathbf{E}_2] \text{ store} = E[\mathbf{E}_1] \text{ store} - E[\mathbf{E}_2] \text{ store}$$

$$E[\mathbf{I}] \text{ store} = N[\mathbf{I}]$$

$$E[\mathbf{V}] \text{ store} = \text{if } \text{store}[\mathbf{V}] = \perp \text{ then } \top \text{ else } \text{store}[\mathbf{V}]$$

Assume that  $N$  returns the integer value of its argument. Given the initial store  $(\lambda \mathbf{V} . \perp) [1/u] [5/y] [5/z]$ , compute the store which results from evaluating  $S$  on the program below, showing all steps used in the evaluation.

```

while (u > 0) loop
  u := u - 1;
  z := z + y
end loop

```