UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

Prof. R. Fateman

Fall, 2002

SOLUTIONS CS 164 Final Examination: December 18, 2002, 12:30-3:30PM

1. [5 points] Recall our discussion of denotational semantics.
a. Where x is a program variable, what does the following statement mean?

$$A[\![x]\!]\sigma \;=\; \sigma(x)$$

solution: evaluate x in the environment σ

b. In this statement

$$A[\![e_1 + e_2]\!]\sigma \;=\; A[\![e_1]\!]\sigma + A[\![e_2]\!]\sigma$$

there are two occurrences of the character +. Write one or perhaps two complete English sentences explaining what they mean. Some words or phrases you might or might not use include syntax, programming language, mathematical, operational, binary, assembler, token, AST, finite state, parser, not a clue.

Solution: The + on the left is the syntactic symbol in the programming language. The + on the right means mathematical addition.

c. What kinds of statements require $\perp$ for their denotational semantics?

solution: This is used for statements that might not terminate: looping for example.

2. [3 points] Using Axiomatic Semantics notation, if $P_i(r)$ is some mathematical statement about the variable r and refers to no other variables, then what can you say at B, given:
$\{P_1(x), P_2(y), P_3(z)\}\ y := x;\ x := x + 1\{B\}$?
Circle the true assertions.
a. $P_1(x)$
b. $P_2(y)$
c. $P_3(z)$
d. $P_1(y)$
e. $P_1(x + 1)$

f. $P_1(x-1)$
g. $P_2(x)$
h. $P_3(y)$
i. $x > y$
j. $y - 1 = x$
solutions : c,d,f,i

3. [12 points] Consider the language L1 defined as follows: L1 consists of sequences of three different symbols a, b, and c. A sentence in L1 is any non-negative number of as followed by *the same number of* bs followed by a (possibly different) number of cs. In other words, L1 = $\{a^n b^n c^m \mid n,\ m \text{ integer}\}$. Here is a subset of L1: $\{\epsilon\ ab\ abcc\ aaabbb\ c\}$.

(a) Write a very simple grammar for L1. An excessively complex grammar will lose points.

```
Solution:  S->UV,  U-> ,  U->aUb,  V-> ,  V->cV
```

(b) Assume you have a lexical analysis program that will provide you, when you call it, with the next token on the input, and that it is a Lisp symbol like a, b, c, d, etc. It will return nil if there are no more tokens.

You have a choice. Do (c-1) or (c-2).
(c-1) Write an LALR grammar with augments that parses exactly this language and returns a pair of numbers n, m corresponding to the counts. Or (c-2) write a recursive descent parser that does the same thing.
If you write both, we will *not* give you the maximum grade of the two.

Here are two programs for recursive descent that you could use:

```
(defun peek()(car tokens)) ;look at next token

(defun eat(z) (or (eql z (pop tokens)) (error "illegal sentence")))
```

If you devise an LALR parser, for full credit you will have to figure out how to initialize the counts.

Use the space below or on the reverse side of this page for your answer.

```
(defparameter mcount 0)
(defparameter ncount 0)
(defparameter tokens '(a a a b b b c))

(defun peek()(car tokens)) ;look at next token

(defun eat(z) (or (eql z (pop tokens))
  (error "illegal sentence")))

(defun S()(let ((mcount 0)(ncount 0)) (U)(V) (list mcount ncount)))

(defun U()(cond ((eql (peek) 'a)
 (eat 'a)(U)
 (incf mcount)
 (eat 'b))))

(defun V()(cond ((eql (peek) 'c)
 (incf ncount)
 (eat 'c) (V))
(t (eat nil))))

An LALR solution might look like this

(defparameter lexforms '(a b c))
(defparameter n 0)
(defparameter m 0)
(defparameter gram ;; a cute trick to initialize m n
    '((S -> A U V #'(lambda(a u v)(list m n)))
      (A -> #'(lambda()(setf n 0)(setf m 0)))
     (U -> #'(lambda()nil)) ;;; or you could just return 0 and increment it as you go
     (U -> a U b #'(lambda(a u b)(incf m) u))
     (V -> #'(lambda() nil)) ;;; or return 0  etc...
     (V -> c V  #'(lambda(c v)(incf n) v))))
```

(d) It should be clear to you that L2=$\{a^n b^m c^m \mid n,\ m$ integer$\}$ *is also* a context free language. (Note that this is like L1 but has equal numbers of b and c).
(d-1) What is the intersection of the languages L1 and L2?
  solution: L3, below
(d-2) We are told the language L3=$\{a^n b^n c^n \mid n,$ integer$\}$ cannot be described by a context free grammar. What can you conclude about the intersection of context free languages?
  solution: the intersection of 2 CFL is not necessarily a CFL.
(d-3) What can you say about the union of context free languages?
  solution: the union of 2 CFL is also a CFL.

4. [3 points] While compiling the previous code, there is an error message issued that says "Error: string expected to be of same type as int". What part of the compilation process found this error, and what does it mean?

solution: The typechecker tells us that print should be given a string not an integer as argument.

5. [3 points] In the Tiger interpreter `tig-interp` there is a big case statement that looks at the operator of the expression being interpreted. If the expression is `(IfExp x y)`, the appropriate branch is

```
   (IfExp (tig-interp-if (elt x 1)(elt x 2) (elt x 3) env))
```

Explain what would be wrong (if anything) with these alternative implementations, implementing `IfExp` in place, by

```
(IfExp (if (tig-interp(elt x 1))(tig-interp (elt x 2))(tig-interp (elt x 3)) env)) ;;A
```

or

```
(IfExp (tig-interp (if (equal 0 (tig-interp(elt x 1)))(elt x 3) (elt x 2) env)) ;;B
```

Hint: You might use words or phrases like Boolean, true, nil, environment, can't tell, applicative order, normal order, correct, Go Bears.

> There are many things wrong. You had to mention at least a few of them including (a) the version of true and false used in lisp is different from Tiger, so you must change nil/ non-nil to 0/1 as in (B). You need to put environment parameters in the calls to tig-interp and not in the if. There is also a missing right paren. in B. If you said something about normal order and "if" evaluating too many things you were wrong. Go Bears.

6. [6 points] The Algol-60 language notably resembles many of its successors, but none of them have taken call by name seriously. In lecture we discussed this Algol-60 program which uses call-by-name parameters.

```
real procedure SIGMA (i,L,U,x);
     value L, U;
     integer i, L, U;
     real x;
     begin real s;  s:=0  for i:=L step 1 until U do s:= s+x;
     SIGMA :=s end.
```

(a) Not all the arguments are call by name here. Which ones are and how do you know they are call by name?

> solution: i, x because they do not appear in the value declaration are name parameters.

(b) What does `SIGMA(k,1,3,k*k)` compute?

> solution: 14

7(c) Does this call work: `SIGMA(s,1,100,A[s])`? (Note that there is a variable named `s` inside the `SIGMA` procedure.) Explain why or why not in one or more complete sentences. You may wish to use words or phrases like function, scope, think, thank, thunk, compiler, polymorphic, mysterious, scheme, ascii, aspic, not a clue.

> Solution: Yes, this works perfectly because s and A[s] are passed in to the SIGMA program as thunks, and there is no conflict with the s that is local to SIGMA. Changing the parameter i is reflected in a change in the s in the thunk, which is used to compute A[s].

7. [6 points] Recall that in the various possible designs of an object-oriented system outlined in Graham's chapter 17, he starts with a model in which every object is a hash-table which can have an entry for every possible message that the object can receive.

> Solution: look at this discussion in Graham's book. Sketch: (a) add :parent; (b) more than one parent, and requiring precedence to resolve conflicts; (c) space and speed are improved.

(a) How does Graham add inheritance to this model?
(b) What is multiple inheritance, and how does this complicate matters?
(c) In most object-oriented models there is a difference between the notion of a class and an instance. What advantages are there to making this distinction?

8. [2 points] One way of encoding pairs (like the concept of lisp's `cons`) in the lambda calculus is to take any pair (x, y) and encode it as `(lambda(b)((b x) y))`. That is, this could be the definition of `(cons x y)`. Once you have pairs, you could make lists of any length and indeed encode the rest of any computing into the lambda calculus. But you don't have to do that. All you have to do, for 1 point each, is provide definitions in the lambda calculus of `car` and `cdr` using the above definition of `cons`. That means in particular `(car (cons x y)) = x` and `(cdr (cons x y)) = y`.

```
 solution

 (car p)  = (p (lambda(z) (lambda(w) z)))
 (cdr p)  = (p (lambda(z) (lambda(w) w)))   ;; or similar
```

9. [4 points] Consider the following intermediate code:

```
    a := x    ;;; partial solution shown here...
L0: a := a*x
    n := n-1
    d := a
    f := 34
    jump L1   ;;; maybe dead code
    e := c    ;;; certainly dead
    d := a    ;;; certainly dead
L1: b := d    ;;; not dead, just pointless
    jumpz n L0
    b := f
L3: d := d - 1
```

Put a box around each basic block. Draw all control flow edges. Draw a line through dead code, assuming that only b and d are live after this portion of code has completed execution.

10. [15 points] Mark the following statements as true (T) or false (F). WARNING: points will be subtracted for incorrect answers, so don't guess.

_______ A nondeterministic finite automaton can accept a larger class of languages than a deterministic finite automaton.

_______ There is a context free grammar to describe any finite set.

_______ Some deterministic finite automata can accept an infinite language.

_______ Every LL(0) language can be described by a regular expression.

_______ Algol-60 allows strings, but has no language-defined operations on them.

_______ Algol-60 has both single and double precision floating point numbers.

_______ C and Pascal have no "power" operation because Algol-60 didn't either.

_______ Lambda calculus provides a model for dynamic scope in languages.

_______ Lambda calculus provides a model for lexical scope in languages.

_______ Lambda calculus has types int and function only.

_______ Depending on the sequence of beta-reductions, a simplification of a lambda calculus expression may terminate or not.

_______ There are LALR(1) languages which require an arbitrary size stack to parse.

_______ Optimization is possible on abstract syntax trees.

_______ Optimization is possible on assembly language code.

_______ The class assembler runs two passes over the sequence of op codes.

Solution: F T T F T F F F T F T T T T T