

Midterm Exam

CSC 254

19 October 2004

Directions

This exam comprises 7 multiple-choice questions, 6 short-answer questions, and 6 brief exercises. Values are indicated for each; they total 60 points. There is also a 6-point extra credit exercise; it won't factor into your exam score, but may help to raise your letter grade at the end of the semester.

This is a *closed-book* exam. You must put away all books and notes. Please confine your answers to the space provided. For multiple choice questions, darken the circle next to the best answer. Be sure to read all candidate answers before choosing.

Put your name on every page. Scratch paper is available if you need it, but the proctor will collect **only the exams**. No partial credit will be given on the multiple-choice questions.

In the interest of fairness, the proctor has been instructed not to answer questions during the exam. If you are unsure what a question is asking, make a reasonable assumption and state it as part of your answer.

You must complete the exam in class. The proctor will collect any remaining exams promptly at 4:40 pm. Good luck!

Multiple Choice (2 points each)

1. Why must local variables in Lisp generally be allocated in the heap, rather than on the stack?
 - a. Because local variables in Lisp have unlimited extent.
 - b. Because we don't know their sizes at compile time.
 - c. Because Lisp subroutines don't follow strict LIFO calling conventions.
 - d. Because the Lisp garbage collector is unable to manipulate addresses in the stack.
2. What is the significance of L-attributed attribute grammars?
 - a. They are the most general class of attribute grammars that can be evaluated in linear time.
 - b. They can be evaluated in the course of an LL parse.
 - c. Their attribute flow is strictly bottom-up.
 - d. They can be handled by the built-in evaluator in `yacc/bison`.
3. What is the difference between *synthesized* and *inherited* attributes?
 - a. Synthesized attributes are initialized by the scanner.
 - b. Synthesized attributes depend only on information below them in the parse tree.
 - c. Inherited attributes are useful only in object-oriented languages.
 - d. Synthesized attributes are computed at run time.

4. *Closures*, which combine a subroutine pointer and a referencing environment, are required only
- a. for languages with static (lexical) scope.
 - b. for first-class subroutines.
 - c. for deep binding.
 - d. All of the above.
5. Which of the following is not a higher-order function?
- a. `map`
 - b. `member`
 - c. `apply`
 - d. `compose`
6. What is the value of `(car (cdr (map * '(1 2 3) '(4 5 6))))`?
- a. a run-time error
 - b. `()` (the empty list)
 - c. 120
 - d. 10
7. What does it mean for a method in C++ to be *virtual*?
- a. It must be defined explicitly in every derived class.
 - b. Only its declaration appears in the `.h` file; its definition is in a separate `.cc` file.
 - c. It overloads a base class method of the same name.
 - d. It employs dynamic dispatch.

Short Answer (3 points each)

8. In some languages the scope of a declaration is its entire block. In others it is the portion of the block after the declaration. Summarize the tradeoffs between these two alternatives.

Answer: Rest-of-block scope requires some special mechanism to cope with recursive definitions. Whole-block scope requires two passes of semantic analysis, and can lead to confusing error messages if the programmer attempts to use a global binding of the name near the beginning of the scope.

9. Why is tail recursion so important in functional languages?

Answer: Because it can be implemented without allocating a new stack frame, giving us the same space and time efficiency as iteration.

10. Briefly state the fundamental tradeoff between applicative- and normal-order evaluation of arguments in functional languages.

Answer: Applicative-order evaluation is usually faster, at least for arguments that are actually used. It's also easier to implement and easier to understand in the presence of side effects. Normal-order evaluation is more general: it permits the execution of certain programs that will crash or fail to terminate with applicative-order evaluation.

11. List two main problems associated with aliases in computer programs.

Answer: They can lead to subtle bugs (unexpected changes and dangling pointers, for example), and they inhibit certain important forms of code improvement.

12. Explain the difference between the *static link* and the *dynamic link* in a stack frame.

Answer: The dynamic link is the saved frame pointer; it points to the frame of the calling subroutine. In a language with lexical scope and nested subroutines, the static link points to the frame of the lexically surrounding subroutine.

13. Suppose we have a table-driven LL(1) parser with an automatically managed attribute stack, like the one in the PL/0 compiler. When the parser is about to expand a given non-terminal X , which records (parse tree nodes) will be on the attribute stack?

Answer: The records for all symbols in productions on the path between X and the root of the parse tree.

Questions 14 through 18 and the extra-credit question (number 20) use the following LL(1) grammar for a simplified subset of Scheme:

$$\begin{aligned} P &\longrightarrow E \$\$ \\ E &\longrightarrow \text{atom} \\ &\longrightarrow ' E \\ &\longrightarrow (E Es) \\ Es &\longrightarrow E Es \\ &\longrightarrow \end{aligned}$$

-
14. (5 points)

- (a) What is $\text{FIRST}(Es)$?

Answer: $\{ \text{atom}, (, ', \epsilon \}$

- (b) What is $\text{FOLLOW}(E)$?

Answer: $\{ \text{atom}, (, ',), \$\$ \}$

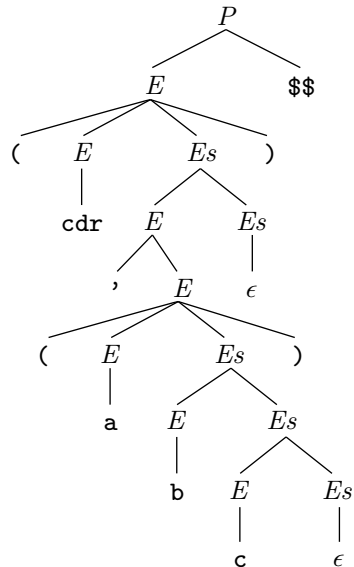
- (c) What is $\text{PREDICT}(Es \longrightarrow \epsilon)$?

Answer: $\{) \}$

(Remember to use the formal definitions of these sets, not the versions from the Scheme programming assignment. Note: you are *not* required to hand-execute the formal algorithm, or show your work. With a grammar this simple you can probably get the right answer just by staring at it for a minute or two.)

15. (5 points) Give a parse tree for the string $(\text{cdr } '(a\ b\ c))\ \$\$$.

Answer:



16. (5 points) Show the first ten lines of the leftmost derivation of $(\text{cdr } '(a\ b\ c))\ \$\$$. (The whole derivation is fourteen lines long.)

Answer: Here is the whole derivation:

```

P
E $$
( E Es ) $$
( cdr Es ) $$
( cdr E Es ) $$
( cdr ' E Es ) $$
( cdr ' ( E Es ) Es ) $$
( cdr ' ( a Es ) Es ) $$
( cdr ' ( a E Es ) Es ) $$
( cdr ' ( a b Es ) Es ) $$
( cdr ' ( a b E Es ) Es ) $$
( cdr ' ( a b c Es ) Es ) $$
( cdr ' ( a b c ) Es ) $$
( cdr ' ( a b c ) ) $$

```

17. (5 points) Here are the first seven lines of a trace of an LL(1) parse of $(\text{cdr } '(a\ b\ c))\ \$\$$.

stack	remaining input
P	$(\text{cdr } '(a\ b\ c))\ \$\$$
E $\$\$$	$(\text{cdr } '(a\ b\ c))\ \$\$$
(E Es) $\$\$$	$(\text{cdr } '(a\ b\ c))\ \$\$$
E Es) $\$\$$	$\text{cdr } '(a\ b\ c))\ \$\$$
atom Es) $\$\$$	$\text{cdr } '(a\ b\ c))\ \$\$$
Es) $\$\$$	$'(a\ b\ c))\ \$\$$
E Es) $\$\$$	$'(a\ b\ c))\ \$\$$

What are the next six lines?

Answer:

```
' E Es ) $$      '(a b c)) $$
E Es ) $$        (a b c)) $$
( E Es ) Es ) $$ (a b c)) $$
E Es ) Es ) $$   a b c)) $$
atom Es ) Es ) $$ a b c)) $$
Es ) Es ) $$     b c)) $$
```

18. (3 points) Now consider a recursive descent parser running on the same input. At the point where the quote token (') is matched, which recursive descent routines will be active (i.e., what routines will have a frame on the parser's run-time stack)?

Answer: P, E, Es, E, and match.

19. (5 points) Real number constants in Scheme are permitted to be explicitly *inexact*. A programmer who wants to express all constants using the same number of digits can use sharp signs (#) in place of any lower-significance digits whose value is not known. For example, the following are all approximations (with varying degrees of inexactness) for the constant value 100π :

```
314.16    five digits specified, all known
31#       three digits specified, one unknown, no decimal point
3##.     three digits specified, two unknown, with decimal point
3##.##   five digits specified, four unknown
314.#    four digits specified, one unknown
314.2##  six digits specified, two unknown
```

Similarly, approximations of $\pi/10$ could be written as

```
00.314   five digits specified (including two leading zeros), all known
0.3##    four digits specified, two unknown
0.##     three digits specified, two unknown
.31###   five digits specified, three unknown, with nothing to left of decimal point
```

Note that all digits must precede all sharp signs, and there must always be at least one digit.

Give a regular expression for (possibly inexact) real number constants in Scheme, as described informally above. You may use the name *digit* as shorthand for (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0). You may also use the "Kleene plus" shorthand notation if you want.

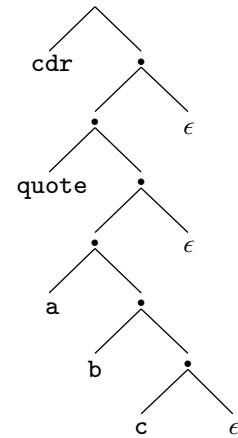
(By the way, numbers in Scheme are actually a good bit more complicated than this. Please ignore anything you may know about sign, exponent, radix, exactness and length specifiers, and complex or rational values.)

Answer: There is, of course, an infinite number of correct answers. Here is the simplest one I was able to come up with:

```
digit+ #* ( . #* |  $\epsilon$  ) | digit* . digit+ #*
```

20. (6 points **Extra Credit only**) Because programs are lists in Scheme, the natural syntax tree for a Scheme program is a tree of `cons` cells. The syntax tree for `(cdr '(a b c))` is shown at right. (Note that `'L` is semantically equivalent to `(quote L)`.)

Extend the CFG on page 3 to create an attribute grammar that will build such trees. When a parse tree has been fully decorated, the root should have an attribute `v` that refers to the syntax tree. You may assume that each atom has a synthesized attribute `v` that refers to a syntax tree node that holds information from the scanner. In your semantic functions, you may assume the availability of a `cons` function that takes two references as arguments and returns a reference to a new `cons` cell containing those references.



Answer:

- | | |
|-------------------------------------|--|
| $P \longrightarrow E \ \$\ \$$ | $\triangleright P.v = E.v$ |
| $E \longrightarrow \text{atom}$ | $\triangleright E.v = \text{atom}.v$ |
| $E_1 \longrightarrow ' E_2$ | $\triangleright E_1.v = \text{cons}(\text{quote}, \text{cons}(E_2.v, \epsilon))$ |
| $E_1 \longrightarrow (E_2 E_s)$ | $\triangleright E_1.v = \text{cons}(E_2.v, E_s.v)$ |
| $E_{s_1} \longrightarrow E E_{s_2}$ | $\triangleright E_{s_1}.v = \text{cons}(E.v, E_{s_2}.v)$ |
| $E_s \longrightarrow \epsilon$ | $\triangleright E_s.v = \text{nil}$ |