

1. Draw all possible parse trees for the string `aaaa` using the grammar  $S \rightarrow SS \mid a$ .

2. Write an unambiguous CFG or BNF grammar (not EBNF) for arithmetic expressions with binary operators `@`, `#`, `$`, and `&`. Operator `@` is left-associative and has highest precedence. Operator `#` is right-associative and has second highest precedence. Operator `$` is left-associative and has third highest precedence. Operator `&` is right-associative and has lowest precedence. All the operands can be denoted by `id`. Handling parentheses is not necessary. Here is an example input string: `id @ id @ id # id # id $ id $ id & id & id`.

- Write a recursive Impcore function (`binary_to_decimal m`) that converts  $m$  from binary to decimal. You may use helper functions if you wish, but do not use `while` or `set` expressions. Example: `(binary_to_decimal 1100)` should return 12.
  
- Complete this natural operational semantics rule for a C-like `DO_WHILE(e1, e2)` in Impcore. The expression is evaluated for its side-effects, so the return value is unimportant. Here is an example as it could appear in a language with an Impcore-like syntax: `(do_while (set x (* 10 x)) (x < 1000))`. The expression  $e_1$  is the body of the loop, and the expression  $e_2$  is the condition. Expression  $e_1$  must be evaluated before  $e_2$ . [Hints: Think about how to define `DO_WHILE` in terms of `WHILE`. Two premises should be sufficient.]

$$\frac{\langle \quad, \xi, \phi, \rho \rangle \Downarrow \quad \quad \langle \text{WHILE}(\quad, \quad), \quad, \quad, \quad \rangle \Downarrow}{\langle \text{DO\_WHILE}(e_1, e_2), \xi, \phi, \rho \rangle \Downarrow} \quad (\text{DO\_WHILE})$$

- Draw a diagram that illustrates the internal representation for this S-expression:  
`((a) (b c) ( ) (d e f . g))`

6. Write a Scheme function (`inner X Y f g`) that returns the inner product of the lists `X` and `Y` with respect to binary operations `f` and `g`. You may use helper functions if you wish, but do not use `while` or `set` expressions. You may assume that lists `X` and `Y` are nonempty lists of the same length, and that operation `f` is associative. Example: `(inner '(1 2 3) '(4 5 6) + *)` should return `32`, because  $1*4+2*5+3*6 = 32$ . This kind of inner product is sometimes also called a dot product. Another example: `(inner '(1 2 3) '(4 5 6) * +)` should return `315`, because  $(1+4)*(2+5)*(3+6) = 315$ .
7. Write a Scheme function (`outer X Y f`) that returns the outer product of the lists `X` and `Y` with respect to binary operation `f`. You may use helper functions if you wish, but do not use `while` or `set` expressions. The result is a list of length  $|X|$ , each of whose members is a sublist of length  $|Y|$ . The  $k^{\text{th}}$  element of the  $j^{\text{th}}$  sublist is obtained by applying operation `f` to the  $j^{\text{th}}$  element of `X` and the  $k^{\text{th}}$  element of `Y`. Example: `(outer '(1 2 3) '(4 5 6 7) *)` should return the list `((4 5 6 7) (8 10 12 14) (12 15 18 21))`. This kind of outer product is sometimes also called a multiplication table. [Hints: Write a helper function that constructs one of the sublists. Also, consider using the `map` function.]

8. Write the output that would be generated by the  $\mu$ Scheme interpreter after entering each line of code below.

```
(val m (lambda (w) (lambda (op x) (set w (op w x))))) ; _____
(val a (m 10)) ; procedure
(val b (m 20)) ; procedure
(val c (m 50)) ; procedure
(a * 3) ; _____
(b + 5) ; _____
(c + 4) ; _____
(a + 6) ; _____
(b * 2) ; _____
(c - 10) ; _____
(a - 1) ; _____
(b - 2) ; _____
(c / 2) ; _____
```

9. Consider this Scheme expression:  $(f (* a b) (* a b) (* a b))$

a. Using `let`, write an equivalent expression that only evaluates  $(* a b)$  once.

b. Using `lambda`, write an equivalent expression that only evaluates  $(* a b)$  once.

10. Complete these natural semantics rules for Scheme's primitive `car` and `cdr` functions.

$$\frac{\langle e, \rho, \sigma_0 \rangle \Downarrow \langle \text{PRIMITIVE}(\text{car}), \sigma_1 \rangle \quad \langle e_1, \rho, \sigma_1 \rangle \Downarrow}{\text{_____}} \quad (\text{CAR})$$

$$\langle \text{APPLY}(e, e_1), \rho, \sigma_0 \rangle \Downarrow$$

$$\frac{\langle e, \rho, \sigma_0 \rangle \Downarrow \quad \langle e_1, \rho, \sigma_1 \rangle \Downarrow}{\text{_____}} \quad (\text{CDR})$$

$$\langle \text{APPLY}(e, e_1), \rho, \sigma_0 \rangle \Downarrow$$