CS 360 Online Test 2

CS 360 Online Exam Honesty Statement
I am fully aware that once I access exam problems I am allowed to work individually only 120 minutes on them, that I may use the textbook, lecture materials, and all other resources available via course website, but neither of the following is permitted: other books or materials, personal notes, web search tools, calculators, contacting other individuals. By making a submission of my answers to the instructor I acknowledge that I followed the statement of online honesty.

You have 120 minutes for working out exam problems, and still 15 more minutes for packaging your answers into pdf format and submitting them to the instructor via e-mail.

Choose five out of the following eight problems. Each problem counts for three points. Indicate what problems you select for grading in case you submit solutions to more than five problems.

1. Express the following statements in terms of predicate logic, i.e. expressions containing quantifiers $\forall, \exists$, logical operators and predicates. Predicates may contain arithmetic variables representing positive integers, arithmetic operations and relations. You may assume that the symbol $m|n$, representing the fact that m divides n is available.

(i) p, q are relatively prime, i.e. they do not have common factors except for 1,

(ii) p, q are different and they have exactly one non-trivial common factor, i.e. different from 1,

(iii) there are infinitely many perfect squares, i.e. integers with the property that their square roots are also integers.

2. Provide a comparative analysis of the following Scheme and Haskell implementations of the merge-sort algorithm. Make sure to discuss similarities and difference on the levels of both data structures and language mechanisms.

(i) Scheme

```
(define (merge fst snd)
  (cond
    ((null? fst) snd)
    ((null? snd) fst)
    (else
     (let ((x (car fst)) (y (car snd)))
       (if (< x y) (cons x (merge (cdr fst) snd))
           (cons y (merge fst (cdr snd))))))))

(define (split lis)
  (cond
    ((null? lis) (cons '() '()))
    ((null? (cdr lis)) (cons (list (car lis)) '()))
    (else
     (let ((a (car lis)) (b (car (cdr lis))) (c (split (cdr (cdr lis)))))
       (cons (cons a (car c)) (cons b (cdr c)))))))
```

```
(define (msort lis)
  (cond
    ((null? lis) '())
    ((null? (cdr lis)) lis)
    (else
     (let* ((c (split lis)) (fst (car c)) (snd (cdr c)))
       (merge (msort fst) (msort snd))))))
```

(ii) Haskell

```
msort [] = []
msort [x] = [x]
msort lis = merge lis1 lis2
            where
            n = (length lis) / 2
            fsthalf = take n lis
            sndhalf = drop n lis
            lis1 = msort fsthalf
            lis2 = msort sndhalf

merge                :: Ord a => [a] -> [a] -> [a]
merge []     ys       = ys
merge xs     []       = xs
merge (x:xs) (y:ys)
        | x <= y      = x : merge xs (y:ys)
        | otherwise   = y : merge (x:xs) ys
```

3. A positive integer is called perfect if it is equal to the sum of its proper positive divisors, i.e. positive divisors different than itself, e.g. $6 = 1 + 2 + 3$.

(i) Express the statement that there are infinitely many perfect numbers in terms predicate logic, i.e. provide an equivalent expression containing quantifiers $\forall, \exists$, logical operators and predicates, where predicates may contain arithmetic variables representing positive integers, arithmetic operations, including Σ notation, and relations. You may assume that the symbol $m|n$, representing the fact that m divides n is available.

(ii) Use Haskell list comprehension to construct a list consisting of all perfect numbers.

4. The code and the documentation of the Metacircular Evaluator are available via the link to SICP section 4.1, they are also presented in Week 6 Part 1 file. Answer the following questions.

(i) How does *apply* operate? Does it create a new environment? If it does, what is its purpose?

(ii) What are compound procedures? Explain in detail how they are evaluated. How is the procedure body treated in the process?

5.

Restate the following Prolog rule in predicate calculus, using appropriate quantifiers:

```
sibling(X, Y) :- mother(M, X), mother(M, Y),
                 father(F, X), father(F, Y).
```

6. Design an attribute grammar recognizing the language consisting of binary strings containing the same number of 0's and 1's (order of symbols should not matter). Explain your construction. Hint: Modify the attribute grammar of PLP problem 4.1 (Quiz 4 practice problem).

7. The following attribute grammar elaborates on the solution of Problem 3 of Quiz 4. The computational target is the same, but we want to construct a grammar and its attribute rules in such a way that the *val* of an internal node is the sum of the *val*'s of its children. Fill in the four missing components of the design provided below. Provide explanations.

C ⟶ *L_digits* . *R_digits*
- ▷ R_digits.pos := −1
- ▷ C.val := L_digits.val + R_digits.val

L_digits ⟶ *digit more_L_digits*
- ▷ digit.pos := more_L_digits.len
- ▷ L_digits.len := **?**
- ▷ L_digits.val := **?**

more_L_digits ⟶ *L_digits*
- ▷ more_L_digits.len := L_digits.len
- ▷ more_L_digits.val := L_digits.val

more_L_digits ⟶ ϵ
- ▷ more_L_digits.len := 0
- ▷ more_L_digits.val := 0

R_digits ⟶ *digit more_R_digits*
- ▷ digit.pos := R_digits.pos
- ▷ more_R_digits.pos := **?**
- ▷ R_digits.val := **?**

more_R_digits ⟶ *R_digits*
- ▷ R_digits.pos := more_R_digits.pos
- ▷ more_R_digits.val := R_digits.val

more_R_digits ⟶ ϵ
- ▷ more_R_digits.val := 0

digit ⟶ 0
- ▷ digit.val := $0 \times 10^{\text{digit.pos}}$

digit ⟶ 1
- ▷ digit.val := $1 \times 10^{\text{digit.pos}}$

digit ⟶ 2
- ▷ digit.val := $2 \times 10^{\text{digit.pos}}$

digit ⟶ 3
- ▷ digit.val := $3 \times 10^{\text{digit.pos}}$

digit ⟶ 4
- ▷ digit.val := $4 \times 10^{\text{digit.pos}}$

digit ⟶ 5
- ▷ digit.val := $5 \times 10^{\text{digit.pos}}$

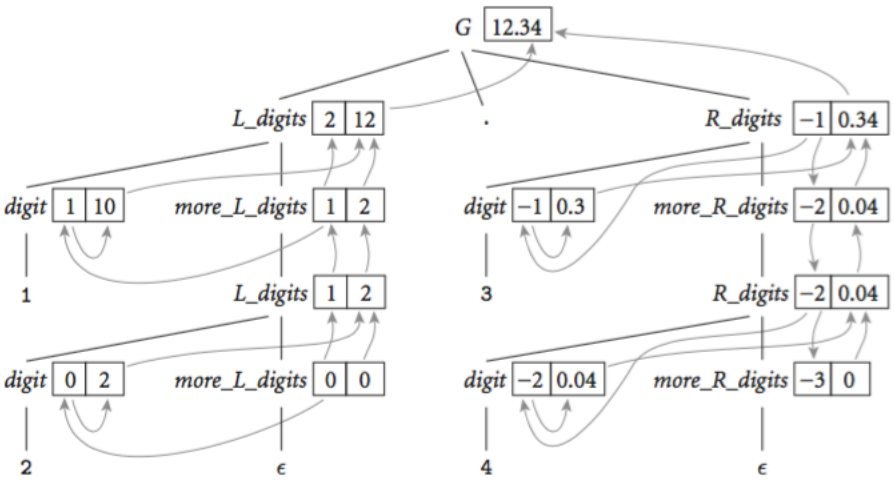
digit ⟶ 6
- ▷ digit.val := $6 \times 10^{\text{digit.pos}}$

digit ⟶ 7
- ▷ digit.val := $6 \times 10^{\text{digit.pos}}$

digit ⟶ 8
- ▷ digit.val := $8 \times 10^{\text{digit.pos}}$

digit ⟶ 9
- ▷ digit.val := $9 \times 10^{\text{digit.pos}}$

The following figure explains the expected attribute flow and it should be helpful in understanding the design.



8. Explain the treatment of types and static semantic errors in the attribute grammar of the calculator language with types of PLP Figure 4.14. Provide examples illustrating the statements you make.