

# Qualifying Exam

## Programming Languages

Adriana Compagnoni

November 26, 2007

### 1 Instructions

- This exam is open books, open notes, and open laptops.
- You can implement your solution in DrScheme and mail it to abc@cs.stevens.edu. Alternatively you can write it on paper.
- You can use the code on the appendix or you can include your own code if you already have it.

### 2 Exercises

In this exam you will be given the definition of the Post-Turing language and some basic functions, and your goal is to complete the implementation with an execution procedure `run` and a couple of auxiliary functions.

A Post-Turing program is a list of instructions that when executed modifies the contents of an infinite tape. The tape contains natural numbers or blanks, and the reading head reads one symbol at a time, and it can move left or right.

The meaning of the instruction set is as follows:

- **PRINT S**: replace the symbol on the position scanned by the reading head by `S`
- **LEFT**: move the reading head one square/symbol to the left.
- **RIGHT**: move the reading head one square/symbol to the right.
- **IF S GOTO L**: if the symbol being scanned by the reading head is `S` then jump to the first instruction with label `L`, otherwise continue with the next instruction. If there is no instruction labeled `L`, halt execution.

An example of Post-Turing code is the following that computes a function that copies its input. In this particular example, the input tape contains a natural number in unary notation ( `0` is the empty string and  $4 = 1111$ ) and

blanks (B) and it assumes that the reading head in the initial configuration stands at the first blank to the left of the input. A jump to a label that does not appear as the label on any instruction (END here) means halting execution.

```
(define copyx '(
  < A > RIGHT
    IF B GOTO END
  < B > PRINT 5
  < C > RIGHT
    IF 1 GOTO C
  < D > RIGHT
    IF 1 GOTO D
    PRINT 1
  < E > LEFT
    IF 1 GOTO E
    IF B GOTO E
    PRINT 1
    RIGHT
    IF B GOTO F
    IF 1 GOTO B
    IF B GOTO B
  < F > LEFT
    IF 1 GOTO F
  ))
```

We define symbols, labels and instructions with the following data-types:

```
(define-datatype sym sym?
  (blank)
  (num (number number?)))
```

```
(define-datatype label label?
  (lab (1 symbol?))
  (nl))
```

```
(define-datatype instr instr?
  (PRINT (1 label?) (s sym?))
  (LEFT (1 label?))
  (RIGHT (1 label?))
  (IF (1 label?) (s sym?) (goto label?)))
```

In the appendix you will find the code for `parse-instrs` that transforms a list of instructions in concrete syntax into a list of instructions in the abstract syntax given by the data-type above.

We define the data-type `program` that consists of a list of instructions `p` and a program counter `next`. The program counter indicates the instruction about to be executed. The first instruction of the program is instruction number 1.

```
(define-datatype program program?
  (prog (p (list-of instr?)) (next number?)))
```

A program executes on an infinite tape containing blanks and only a finite number of numbers. The data-type for tapes is as follows.

```
(define-datatype tape tape?
  (tp (left (list-of sym?))
      (head sym?)
      (right (list-of sym?))))
```

The code for the procedures on a tape to move left, move right, and to print are given in the appendix.

1. Write a Scheme procedure (`lookup-instr l n`) that given a list `l` of `instr` and a number `n` finds the instruction at position `n` in instruction list `l`.

```
>(lookup-instr (parse-instrs copyx) 1)
#(struct:RIGHT #(struct:lab A))

>(lookup-instr (parse-instrs copyx) 5)
#(struct:IF #(struct:nl) #(struct:num 1) #(struct:lab C))

>(lookup-instr (parse-instrs copyx) 25)
lookup-instr: Instruction not found
```

2. Write a Scheme procedure (`lookup-lab-instr instrs lb n end`) that finds the index of the first instruction labeled `lb` with offset `n` and returns `end` if there is no instruction with such a label.

```
>(lookup-lab-instr (parse-instrs copyx)(lab 'F) 1 19)
17

>(lookup-lab-instr (parse-instrs copyx)(lab 'A) 5 19)
5

>(lookup-lab-instr (parse-instrs copyx)(lab 'start) 1 19)
19
```

3. Write a Scheme procedure (`run it p`) that executes the program `p` on input tape `it`.

```
>(unparse (run (input-to-tape '(B 1 1)) (prog(parse-instrs copyx) 1)))
(< B > 1 1 B 1 1)

>(unparse (run (input-to-tape '(B 1 1 1)) (prog(parse-instrs copyx) 1)))
(< B > 1 1 1 B 1 1 1)
```

`unparse` and `input-to-tape` convert between lists of symbols and tapes. See code in the appendix.

`(B 1 1)` is short hand for `(< B > 1 1)` where `< B >` indicates the symbol being scanned by the reading head.

**Good Luck!**

## A Scheme Code

```
(define parse-instrs
  (lambda (l)
    (if (null? l)
        '()
        (if (eqv? (car l) '<)
            (let ((instr-name (caddr l)))
              (cond
                ((eqv? instr-name 'PRINT) ; it is a labeled PRINT
                 (cons (PRINT (lab(cadr l))
                       (let ((symbol (car (cddddr l))))
                         (if (number? symbol)(num symbol )(blank)))) (parse-instrs
                               ((eqv? instr-name 'RIGHT)
                                (cons (RIGHT (lab (cadr l))) (parse-instrs (cddddr l))))
                               ((eqv? instr-name 'LEFT)
                                (cons (LEFT (lab (cadr l))) (parse-instrs (cddddr l))))
                               ((eqv? instr-name 'IF)
                                (cons (IF (lab(cadr l))
                                        (let ((symbol (car (cddddr l))))
                                          (if (number? symbol)
                                              (num symbol)
                                              (blank)))
                                        (lab (car (cdr (cdr (cddddr l))))))
                                (parse-instrs (caddr (cddddr l))))
                               (else (eopl:error 'parse-instrs "unrecognized labeled instruction ~s" instr.
                                         ; it is an unlabeled instruction
                                         (let ((instr-name (car l)))
                                           (cond
                                             ((eqv? instr-name 'PRINT) ; it is an unlabeled PRINT
                                              (cons (PRINT (nl) (let ((symbol (cadr l)))
                                                            (if (number? symbol)
                                                                (num symbol)
                                                                blank)))
                                                  (parse-instrs (caddr l))))
                                             ((eqv? instr-name 'RIGHT)
                                              (cons (RIGHT (nl)) (parse-instrs (cdr l))))
                                             ((eqv? instr-name 'LEFT)
```

```

        (cons (LEFT (nl)) (parse-instrs (cdr 1))))
      ((eqv? instr-name 'IF)
       (cons (IF (nl)
                 (let ((symbol (car (cdr 1))))
                   (if (number? symbol)
                       (num symbol)
                       (blank)))
                 (lab (caddr 1)))
              (parse-instrs (cddddr 1))))
      (else (eopl:error 'parse-instrs "unrecognized unlabeled instruction ~s" instr-name)
            ))
    ))
  ))))

(define LEFTproc
  (lambda (t)
    (cases tape t
      (tp (left head right)
         (if (null? left)
             (tp '() (blank) (cons head right))
             (tp (cdr left) (car left) (cons head right)))))))

(define RIGHTproc
  (lambda (t)
    (cases tape t
      (tp (left head right)
         (if (null? right)
             (tp (cons head left) (blank) '())
             (tp (cons head left) (car right) (cdr right)))))))

(define PRINTproc
  (lambda (t s)
    (cases tape t
      (tp (left head right)
         (tp left s right))))))

(define sym-to-symbol
  (lambda (s)
    (cases sym s
      (blank{} 'B)
      (num(n) n))))

(define symbol-to-sym
  (lambda (s)
    (if (eqv? s 'B) (blank) (if (number? s)(num s)
                                  (eopl:error 'symbol-to-sym "~s is not a valid tape symbol" s))))))

```

```

(define symlist-to-list
  (lambda (sl)
    (if (null? sl) '() (cons (sym-to-symbol (car sl)) (symlist-to-list (cdr sl))))))

(define symlist1
  (list (blank) (num 3) (num 4) (num 5)))

(define list-to-symlist
  (lambda (l)
    (if (null? l) '()
        (if (eq? (car l) 'B)
            (cons (blank) (list-to-symlist (cdr l)))
            (cons (num (car l)) (list-to-symlist (cdr l)))))))

(define input-to-tape
  (lambda (l)
    (if (null? l)
        (tp '() (blank) '())
        (tp '{ } (symbol-to-sym (car l)) (list-to-symlist (cdr l))))))

(define unparse
  (lambda (t)
    (cases tape t
      (tp (left head right) (append (reverse (symlist-to-list left))
                                     (cons '< (cons (sym-to-symbol head) (cons '> (symlist

```