CSE 413 19wi Final Exam Sample Solution

Question 1. (18 points) The MUPL Memorial Question (feel free to go on to other questions and come back to this later). Also, DON'T PANIC! The question descriptions are much longer than the answers.

Several pages of MUPL information are handed out on separate pages for you to use as reference material while you answer this question. The first two pages are from the MUPL assignment (hw5), giving the details of the MUPL language. The last two pages are the starter code for hw5, including the Racket struct definitions used to implement MUPL, which may also be helpful for reference.

When we were implementing MUPL we discovered that there were two ways to extend the language. One was to directly add a new kind of expression to MUPL itself, by adding code to the interpreter. The other was to write a Racket function that acts like a MUPL macro, producing as its output a MUPL expression that can be evaluated directly by using it as an argument to the MUPL `eval-exp` interpreter function, or that can be used as parts of larger MUPL programs. In this question we want to explore both mechanisms to add a new `ifpos` expression to MUPL.

The `ifpos` expression is defined as follows:

- If e_1, e_2, and e_3 are MUPL expressions, then `(ifpos` e_1 e_2 e_3`)` is a MUPL expression. The value of `(ifpos` e_1 e_2 e_3`)` is either the value of e_2 if e_1 is a positive MUPL integer (i.e., $e_1 > 0$), or is the value of e_3 if e_1 is a MUPL integer that is not positive (i.e., $e_1 \leq 0$). Only one of e_2 or e_3 are evaluated – the other expression is not evaluated. The expression e_1 is only evaluated once. If e_1 is not a MUPL integer (i.e., something not created by `(int …)`), then execution should be terminated with a suitable error message.

(a) (MUPL) (6 points) Complete the following *Racket* function definition `(ifpos e1 e2 e3)` so it returns as its result a MUPL expression that has the same meaning as the `ifpos` expression defined above. In implementing this Racket function, do not use `closure` (which is only used internally by `eval-exp-env`). Also do not use `eval-exp` or `eval-exp-env` (we are generating a MUPL expression, not evaluating it). Hints: MUPL's `ifnz` and `isgreater` might be useful in the output of this function. Also, this is the same kind of question as "question 3 – expanding the language" in the hw5 MUPL assignment.

```
(define (ifpos e1 e2 e3) ;; add your code below
  (ifnz (isgreater e1 (int 0))
        e2
        e3)

)
```

(continued on next page)

Question 1. (cont.) (b) (12 points) The other way we could add `ifpos` to MUPL is to modify the MUPL interpreter `eval-under-env` function directly by adding this new kind of expression to the core MUPL language. Write code below to add an interpreter case for `ifpos` to `eval-under-env`. You should assume the following structure has been added to represent this expression:

```
(struct ifpos (e1 e2 e3) #:transparent) ;; if e1>0 then e2 else e3
```

The original MUPL starter code is included with the pages handed out separately for reference.

Reminder: The Racket function `(error` "message"`)` can be used to terminate evaluation with the given message.

```
(define (eval-under-env e env)
  (cond [(var? e)
         (envlookup env (var-string e))]
        ;; remaining cases omitted
        ;; CHANGE add your code for ifpos below
        [(ifpos? e)
         (let ([v (eval-under-env (ifpos-e1 e) env)])
           (if (int? v)
               (if (> (int-num v) 0)
                   (eval-under-env (ifpos-e2 e) env)
                   (eval-under-env (ifpos-e3 e) env))
               (error "MUPL ifpos applied to non-number")))]
```

CSE 413 19wi Final Exam Sample Solution

Question 2. (16 points, 4 points each) Regular expressions. For each of the following give a regular expression that generates the set of strings described and, for the last part, draw a DFA. There is lots of blank space for your answers – don't worry if you don't need nearly this much room.

Fine print: You may use basic regular expressions (sequences rs, choice r|s, repetition r*, and parentheses for grouping). You may also use + (one or more) and ? (zero or one), and character classes like [ax-z] and [^abc]. You also may use named abbreviations like "vowels ::= [aeiou]" if these help. You may not use additional regular expression operators that might be found in various programming language libraries or software tools. Use underlining if you need to distinguish a terminal symbol like * from a regular expression operator like *.

(a) All strings of 0's and 1's that do not contain the sequence "10". Includes: "" (empty), "0", "1", "011", "00001"; does not include "10", "0111011"

`0*1*`

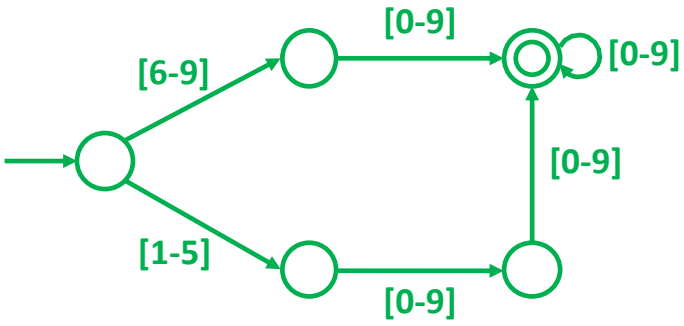
(b) All simple arithmetic expressions formed only from the single-digit number 2, and the addition (+) and subtraction (-) operators. Every expression must contain at least one number and may not include any leading or trailing or adjacent + or – operators. Includes: "2", "2+2", "2-2-2+2"; does not include: "+2", "-2", "2+", "1+2" "2+-2".

`2(+2|-2)*  or  2((+|-)2)*`

(c) All positive decimal integers with a value of at least 60, with no leading 0s. Includes: "60", "600", "413", "1742"; does not include: "", "065", "59", "17"

`([6-9][0-9]) | ([1-9][0-9][0-9]+)  or  (([1-5][0-9][0-9]+) | ([6-9][0-9]+))`

(d) Draw a DFA that accepts the set of strings described in part (c) above (decimal integers ≥ 60 with no leading 0s).



CSE 413 19wi Final Exam Sample Solution

Question 3. (14 points) Consider the following context-free grammar. For this problem, ignore any whitespace – it is not part of what the grammar generates and is only included for clarity.

```
S ::= y A x B
A ::= x y A | ε
B ::= y x B | ε
```

(a) (3 points) List all of the non-terminal symbols in the grammar:

S A B

(b) (3 points) List all of the terminal symbols in the grammar:

x y

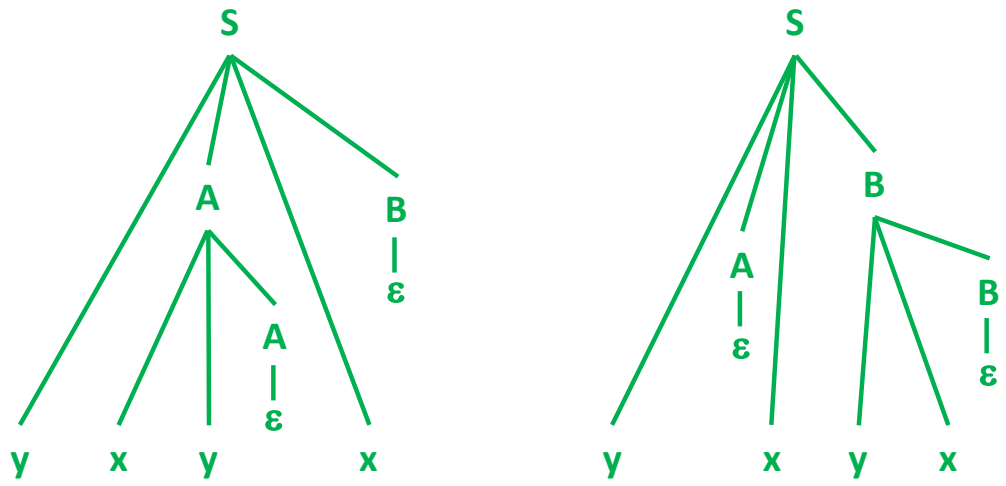
(c) (8 points) Demonstrate that this grammar is ambiguous. (Recall that you can do this with either parse trees or leftmost-only or rightmost-only derivations.)

Two leftmost derivations of yxyx:

S => yAxB => yxyAxB => yxyxB => yxyx

S => yAxB => yxB => yxyxB => yxyx

Two trees:



Question 4. (20 points) Ruby programming. Suppose we have a Ruby hash representing possible flights between airports, in which each key is a string with a 3-letter airport code and the corresponding value is an array containing the codes of all other airports reachable directly from this airport. You may assume that if airport A appears in the destinations list for airport B, then B will also appear in the destinations of A. For example, the following hash includes information that there is a flight from SEA to SFO and vice versa, but not from SEA to MSP or vice versa:

```
flights = {
   'SEA' => ['SFO', 'JFK'],
   'JFK' => ['SEA', 'MSP', 'SFO'],
   'MSP' => ['JFK'],
   'SFO' => ['SEA', 'JFK']
}
```

Write the two Ruby functions described below. The answers do not require a lot of code.

(a) (10 points) Write a Ruby method called `path_exists?` that takes a flights hash as described above and an "itinerary" array containing a sequence of airports as arguments, and returns true if it is possible to take direct flights connecting those airports in the given order. For this method you may not use any `for` or `while` loops – you may only use blocks and methods that use them like `.each`. You can assume the itinerary will have at least two elements – the starting and ending airports.

For example, using the above flights hash table, the call `path_exists?(flights, ['SEA', 'SFO', 'JFK', 'MSP'])` should return true, but `path_exists?(flights, ['SEA', 'MSP'])` should return false.

```
def path_exists?(flights, itinerary)
  last = itinerary[0]
  itinerary[1..-1].each do | city |   # [1..] works in Ruby 2.6.x
      return false unless flights[last].include? city
      last = city
  end
  true
end
```

(continued on next page)

Question 4. (cont.) (b) (10 points) Write a Ruby method called `search` that takes a flights hash argument as described on the previous page and returns information about entries in that hash table. The `search` method will be called with an associated block that basically describes a "query" in the flights table. The result of `search` should be an array containing all of the airport name strings (keys) in the original hash for which the query in the block returns true after examining the destinations array associated with that airport string.

Here are some example usages of the search method, using flights as defined on the previous page:

```
search(flights) { |destinations| destinations.length == 2 }
# => returns ['SEA', 'SFO']
search(flights) { |destinations| destinations.include? 'SFO' }
# => returns ['SEA', 'JFK']
```

Recall that you can invoke a block passed to the method using Ruby's `yield` keyword, and `yield` can include argument(s) to be passed to the block. Also, recall that when you iterate through a Ruby hash using `.each`, you can include a block with two parameters to access the (key, value) pairs in the hash.

```
def search(flights)
    airports = []
    flights.each do |source, destinations|
        if yield(destinations)
            airports.push(source)
        end
    end
    airports
end
```

Question 5. (16 points) This question concerns the calculator language from the last two assignments. If you recall, the grammar for the calculator language was as follows:

program ::= *statement* | *program statement*
statement ::= *exp* | *id* = *exp* | `clear` *id* | `list` | `quit` | `exit`
exp ::= *term* | *exp* + *term* | *exp* − *term*
term ::= *power* | *term* * *power* | *term* / *power*
power ::= *factor* | *factor* ** *power*
factor ::= *id* | *number* | `(` *exp* `)` | `sqrt (` *exp* `)`

We would like to add a new `min(`*exp*, *exp*`)` function to the language. The meaning of the new `min` function is the obvious one – it evaluates both of its operand expressions and the value of the `min` function is the smaller of those two values.

(a) (3 points) We propose to add this new function to the language by adding the following additional production to the grammar rule for factor:

factor ::= … | `min (` *exp* `,` exp `)`

Does this additional rule make the grammar ambiguous? If so, give an example that shows that it does. If not, give a short, but convincing explanation of why not.

No. The grammar rule begins with a unique keyword (`min`). The nonterminals *exp* in that production are separated by a comma, which does not appear anywhere else in the grammar, so the comma separated list of *exp* , *exp* won't be ambiguous, and it is surrounded by a balanced set of parentheses to separate it from adjacent expressions.

(b) (3 points) What changes or additions are needed in the calculator's scanner and in the `Token` class to add this new `min` function expression?

`min` and comma (,) are new lexical classes in the grammar. They need to be added as new kinds of tokens and the scanner needs to be modified to recognize them.

(continued next page)

Question 5. (cont.) (c) (10 points) Below, write the additional Ruby code needed in method `factor` needed to parse and evaluate this new `min` function. Your answer should be guided by your answers to the previous parts of the question, but adjusted as needed for use in a recursive-descent parser. You should make the following assumptions (if needed):

- The scanner and `Token` class have been modified as described above in your solution to part (b). You can call `next_token` to return the next `Token` input object whenever you need it.
- The `kind` method of a `Token` object returns a string that contains the literal text that represents the token class, like ")", or "+", or "exit". It returns "ID" or "NUMBER" for an identifier or number, respectively. For an id or number token, the `value` method returns the specific identifier or number.
- There is a global variable named `$current_token` that contains the next unprocessed `Token` read from the input at all times. Your code **must** update this variable appropriately as it parses the input and it **must** always contain the next unprocessed token. There is no "look-ahead" or "peek" function in the scanner.
- Functions like `exp` and `term` exist to parse each grammar nonterminal and return its value, if any. These functions have no parameters and expect `$current_token` to be the first token in the grammar nonterminal they are parsing when they are called.
- You may assume there are no syntax errors, missing or extra tokens or other errors in the calculator input.

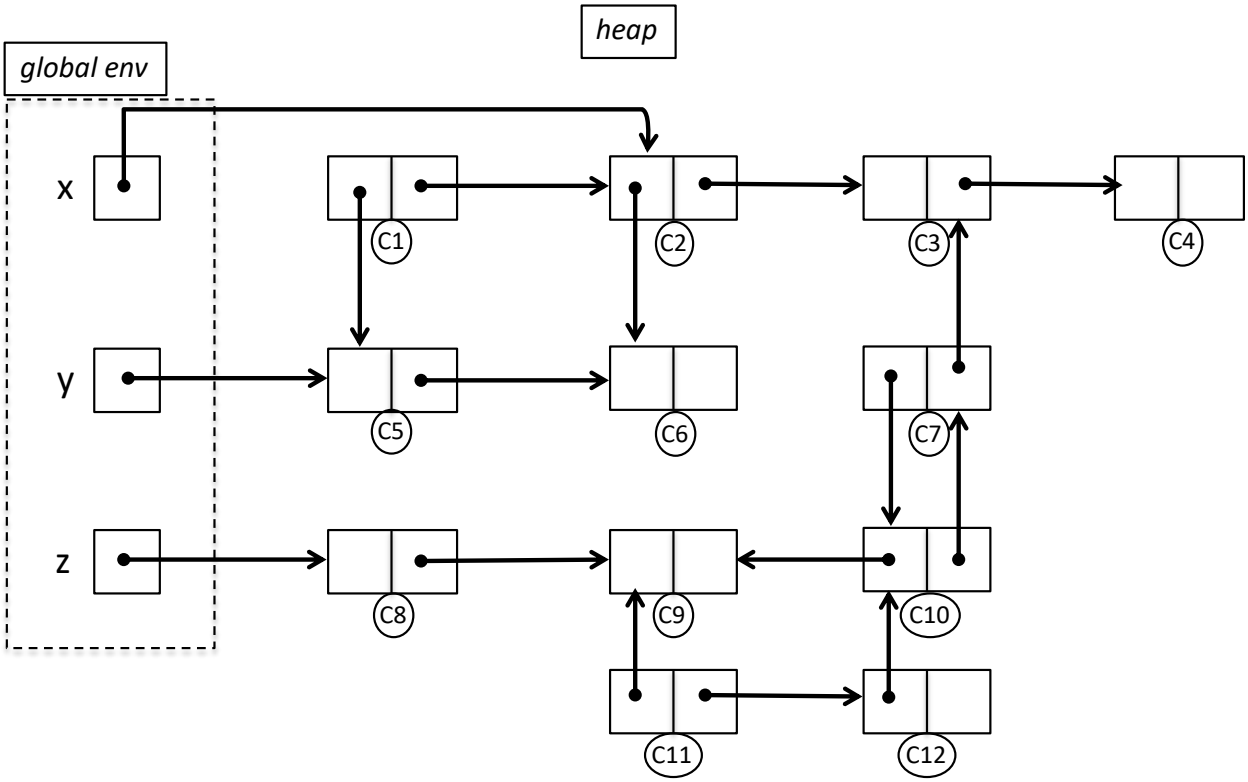
Write your additional code that needs to be added to `factor` below. If needed, state any extra assumptions you need to make in your solution.

Insert the following code into the parser method for `factor`:

```
if $current_token.kind == "min"
  next_token                    # skip min
  $current_token = next_token   # skip '('
  e1 = exp                      # first exp
  $current_token = next_token   # skip past ','
  e2 = exp                      # second exp
  $current_token = next_token   # skip ')'
  if e1 < e2 then e1 else e2 end
end
```

Question 6. (16 points) The quarter is almost over, so it's time to take out the garbage. A couple of questions here on memory management.

We ran a Racket program with all of the garbage collection turned off and observed its memory allocation. Here is a diagram of the final memory usage when the program was about to terminate. This shows the global environment, all of the allocated memory, and pointers from variables and heap objects (cons cells) to other heap objects. Parts of heap objects (cons cells) that do not contain pointers to other heap objects are blank. The different heap objects are labeled C1, C2, C3, etc.

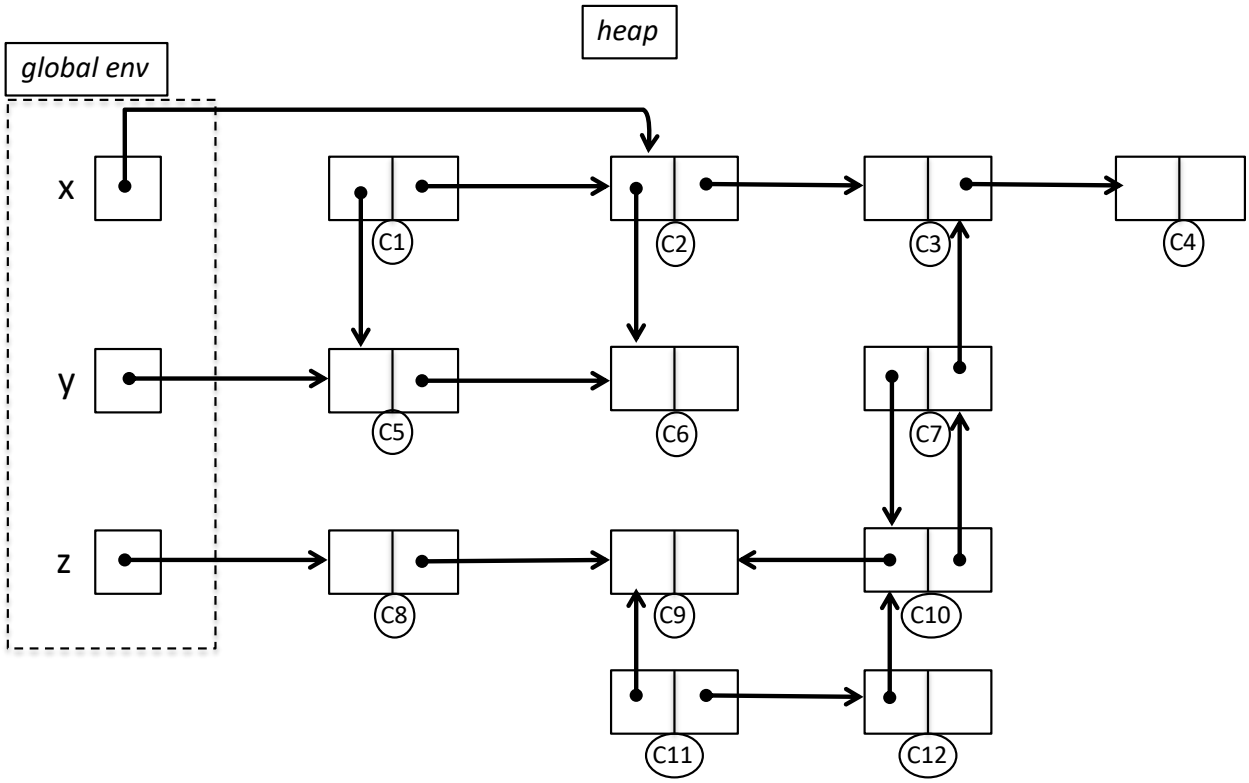


(a) (8 points) Suppose we use a *reference counting* memory manager to perform automatic deletion of appropriate heap objects (cons cells) in this memory diagram. List below the identifying numbers (e.g., c17, etc.) of heap objects that would be deleted using reference counting. Hint: remember that deleting an object that contains pointers might change the reference counts of other heap objects. Request: if you can list the deleted cells in more-or-less numeric order that will help the staff grade the question.

C1 C11 C12

(continued on next page)

Question 6. (cont) Here is the same memory diagram from part (a), above.



(b) (8 points) Suppose we instead use a *mark-sweep* garbage collector this time to preform automatic deletion of appropriate heap objects. List below the identifying numbers (e.g., c17, etc.) of heap objects that would be deleted by a mark-sweep collector. As before: if you can list the deleted cells in more-or-less numeric order that will help the staff grade the question.

C1 C7 C10 C11 C12

Have a great spring break and best wishes for the future!
The CSE 413 staff