HW#2: Understanding Parsong and Substitution (14 points)

***Goal: This HW#2 can help you clearly understand a parser and the substitution concepts for our WAE learned in L6 and L7.***

***All problems can be solved by the knowledge from slides in L04~L07 and information in this doc. You may discuss with your classmates and get some help from others but for each answer you must add comments whether you get help from your friends, TAs, JC or other materials except for slides and textbook. Never look at, blindly copy, and paste code from other students. So, please, add Solve by myself with additional comments and time taken as well. Please add [contract], [purpose] as comments for all functions you implement including free-ids.***

# Background

1. Postfix WAE (i.e. PWAE

<PWAE> ::= <num>

| <op>

| <id>

| <keyword>

| {<PWAE> <PWAE> <op>}

| {{<id> <PWAE>} <PWAE> <keyword>}

The original WAE introduced in L6~7 is prefix WAE. However, Postfix WAE is parsed and interpreted in postfix notation. There are some differences between prefix WAE and postfix WAE. First, There are non-terminal <op> and <keyword>. <op> means operators. It contains symbol + and symbol -. <keyword> means symbol *with* the substitution keyword. Second, of course, it is parsed and interpreted in postfix.

For example, *(parse '{{3 4 -} 7 +})*, generate

*(postfix (postfix (num 3) (num 4) (op 'sub)) (num 7) (op 'add)).*

And *(parse '{{x 5} {x x +} with})* generate

*(substitute 'x (num 5) (postfix (id 'x) (id 'x) (op 'add)) (keyword 'with))*. You can find more details about the differences prefix and postfix notation in <http://www.cs.man.ac.uk/~pjj/cs212/fix.html>.

# Problem

1. (5 points)

(a) Define type PWAE (2 point)

;Define PWAE type

;<PWAE> ::= <num>

; | <op>

; | <id>

; | <keyword>

; | {<PWAE> <PWAE> <op>}

; | {{<id> <PWAE>} <PWAE> <keyword>}

Define a type for Postfix WAE, PWAE. There are variants named *num*, *op*, *id*, and *keyword*, *postfix* and *substitute* which consumes number, symbol, symbol, and symbol. And there are *postfix* variants which consume two PWAE as operands, and *substitute* variant which consumes symbols as identifier, PWAE as named-expression, another PWAE as body, and keyword as *with* keyword. Assume that <op> allows + and - operators only.

(b) Implement Parser for PWAE (3 point)

Implement a parser for PWAE. Expressions in postfix are consumed as an input, and PWAE comes out as an output.

E.g1) Command: *(parse '{{3 4 -} 7 +})* generates

*(postfix (postfix (num 3) (num 4) (op 'sub)) (num 7) (op 'add))*

E.g2) Command: *(parse '{{x 5} {x x +} with})* generates

*(substitute 'x (num 5) (postfix (id 'x) (id 'x) (op 'add)) (keyword 'with))*

E.g3) Command: *(parse '{{x {5 4 +}} {x x +} with})* generates

*(substitute 'x (postfix (num 5) (num 4) (op 'add)) (postfix (id 'x) (id 'x) (op 'add)) (keyword 'with))*

1. (5 points)

; free-ids : WAE → list-of-sym  
Implement the function free-ids, which takes a WAE and produces a list of symbols. The list should contain a symbol for each free identifier in the given WAE, each symbol should appear at most once in the list, and the symbols should be ordered according to the following comparison procedure:

* 1. (define (symbol<? a b) (string<? (symbol->string a) (symbol->string b)))
     1. This function is used with sort function. Check the description and example of 'sort': <https://docs.racket-lang.org/reference/pairs.html#%28def._%28%28lib._racket%2Fprivate%2Flist..rkt%29._sort%29%29>
     2. You may not use symbol<? and sort function but solve this by using it. (Coding by understanding API doc is an important skill. Practice by yourself.)
  2. Use '#lang plai' as the first line in your code.
  3. Hint 1: First, write a function that generates an unordered list of symbols with duplicates, and then write separate functions to re-order the list and then remove duplicates.
  4. Hint 2: The free identifiers of a 'with' form include all of the free identifiers in the named expression, plus all of the free identifiers in the body expression except for the name bound by any 'with' in a valid scope. (See test cases for clear understanding.)
  5. Hint 3: The PLAI language includes several helpful list-processing functions: filter, sort, member, …
     1. <http://docs.racket-lang.org/reference/pairs.html>
  6. Do not forget to define the WAE data type as an instance of WAE is the input argument of the functions you have to implement.
  7. You may implement Task 2 or 3 first as this task might be the most challenging one in HW#2.

1. (2 points)  
   ; binding-ids : WAE → list-of-sym  
   Implement the function binding-ids, which is like free-ids, but the result list contains a symbol for each binding identifier in the given WAE (whether or not the binding identifier is ever referenced by a bound identifier). The result list of symbols must be sorted and have no duplicates.
2. (2 points)  
   ; bound-ids : WAE → list-of-sym  
   Implement the function bound-ids, which is like free-ids, but the result list contains a symbol for each bound identifier in the given WAE. The result list of symbols must be sorted and have no duplicates.

Here are some tests:

|  |
| --- |
| ;; free-ids  (test (free-ids (substitute 'x (num 3) (postfix (id 'x) (postfix (num 3) (id 'x) (op 'sub)) (op 'add)) (keyword 'with))) '())  (test (free-ids (substitute 'x (num 3) (postfix (id 'a) (postfix (num 4) (id 'x) (op 'add)) (op 'sub)) (keyword 'with))) '(a))  (test (free-ids (substitute 'x (num 3) (postfix (id 'b) (postfix (id 'a) (id 'x) (op 'sub)) (op 'sub)) (keyword 'with))) '(a b))  (test (free-ids (substitute 'x (num 3) (postfix (id 'a) (postfix (id 'b) (postfix (id 'x) (id 'b) (op 'add)) (op 'sub)) (op 'sub)) (keyword 'with))) '(a b))  (test (free-ids (substitute 'x (num 3) (postfix (id 'y) (substitute 'y (num 7) (postfix (id 'x) (postfix (id 'b) (id 'a) (op 'sub)) (op 'add)) (keyword 'with)) (op 'sub)) (keyword 'with))) '(a b y))  (test (free-ids (substitute 'x (id 't) (postfix (id 'x) (substitute 'y (id 'y) (postfix (id 'x) (postfix (id 'b) (id 'a) (op 'sub)) (op 'add)) (keyword 'with)) (op 'sub)) (keyword 'with))) '(a b t y))  (test (free-ids (substitute 'x (substitute 'y (num 3) (postfix (id 'x) (id 'y) (op 'sub)) (keyword 'with)) (postfix (id 'x) (id 'y) (op 'add)) (keyword 'with))) '(x y))  (test (free-ids (postfix (substitute 'x (num 10) (substitute 'x (num 3) (postfix (id 'y) (substitute 'y (num 7) (postfix (id 'x) (postfix (id 'c) (id 'b) (op 'sub)) (op 'add)) (keyword 'with)) (op 'sub)) (keyword 'with)) (keyword 'with)) (substitute 'a (id 'a) (id 'a) (keyword 'with)) (op 'add))) '(a b c y))  (test (free-ids (postfix (substitute 'x (num 10) (substitute 'x (num 3) (postfix (id 'y) (substitute 'y (num 7) (postfix (id 'x) (postfix (id 'c) (id 'b) (op 'sub)) (op 'add)) (keyword 'with)) (op 'sub)) (keyword 'with)) (keyword 'with)) (substitute 'a (id 'd) (id 'a) (keyword 'with)) (op 'add))) '(b c d y))  (test (free-ids (postfix (substitute 'x (num 10) (substitute 'x (num 3) (postfix (id 'y) (substitute 'y (num 7) (postfix (id 'x) (postfix (id 'c) (id 'b) (op 'sub)) (op 'add)) (keyword 'with)) (op 'sub)) (keyword 'with)) (keyword 'with)) (substitute 'a (id 'd) (id 'z) (keyword 'with)) (op 'add))) '(b c d y z))  ;; binding-ids  (test (binding-ids (postfix (num 3) (postfix (id 'x) (id 'y) (op 'sub)) (op 'add))) '())  (test (binding-ids (substitute 'y (num 3) (substitute 'x (id 'x) (id 'y) (keyword 'with)) (keyword 'with))) '(x y))  (test (binding-ids (substitute 'y (num 3) (substitute 'y (id 'x) (postfix (id 'x) (id 'y) (op 'add)) (keyword 'with)) (keyword 'with))) '(y))  (test (binding-ids (substitute 'y (num 3) (substitute 'y (substitute 'x (postfix (num 3) (id 'y) (op 'sub)) (postfix (id 'x) (id 'y) (op 'sub)) (keyword 'with)) (postfix (id 'x) (id 'y) (op 'add)) (keyword 'with)) (keyword 'with))) '(x y))  (test (binding-ids (substitute 'z (num 3) (substitute 'w (substitute 'z (postfix (num 3) (id 'y) (op 'add)) (postfix (id 'x) (id 'y) (op 'sub)) (keyword 'with)) (substitute 'w (id 'y) (postfix (num 7) (id 'w) (op 'add)) (keyword 'with)) (keyword 'with)) (keyword 'with))) '(w z))  ;; bound-ids  (test (bound-ids (substitute 'x (num 3) (postfix (id 'y) (num 3) (op 'add)) (keyword 'with))) '())  (test (bound-ids (substitute 'x (num 3) (postfix (id 'x) (postfix (id 'x) (id 'y) (op 'sub)) (op 'add)) (keyword 'with))) '(x))  (test (bound-ids (substitute 'x (num 3) (postfix (id 'x) (substitute 'y (num 7) (postfix (id 'x) (id 'y) (op 'sub)) (keyword 'with)) (op 'add)) (keyword 'with))) '(x y))  (test (bound-ids (substitute 'x (num 3) (substitute 'y (id 'x) (postfix (num 3) (id 'y) (op 'sub)) (keyword 'with)) (keyword 'with))) '(x y))  (test (bound-ids (substitute 'x (num 3) (postfix (id 'y) (substitute 'y (id 'x) (postfix (num 3) (num 7) (op 'sub)) (keyword 'with)) (op 'add)) (keyword 'with))) '(x))  (test (bound-ids (substitute 'x (id 'x) (postfix (id 'y) (substitute 'y (id 'y) (postfix (num 3) (substitute 'z (num 7) (postfix (id 'z) (id 'x) (op 'sub)) (keyword 'with)) (op 'sub)) (keyword 'with)) (op 'add)) (keyword 'with))) '(x z))  (test (bound-ids (substitute 'x (substitute 'y (num 3) (postfix (id 'x) (id 'y) (op 'add)) (keyword 'with)) (postfix (id 'y) (substitute 'y (id 'y) (postfix (num 3) (num 7) (op 'sub)) (keyword 'with)) (op 'add)) (keyword 'with))) '(y))  (test (bound-ids (substitute 'x (id 'a) (substitute 'y (id 'b) (substitute 'z (id 'c) (postfix (id 'd) (postfix (id 'x) (postfix (id 'y) (id 'z) (op 'add)) (op 'sub)) (op 'sub)) (keyword 'with)) (keyword 'with)) (keyword 'with))) '(x y z))  (test (bound-ids (postfix (substitute 'x (num 10) (substitute 'x (num 3) (postfix (id 'y) (substitute 'y (num 7) (postfix (id 'x) (postfix (id 'c) (id 'b) (op 'sub)) (op 'sub)) (keyword 'with)) (op 'sub)) (keyword 'with)) (keyword 'with)) (substitute 'a (id 'd) (id 'a) (keyword 'with)) (op 'add))) '(a x))  (test (bound-ids (postfix (substitute 'x (num 10) (substitute 'x (num 3) (postfix (id 'y) (substitute 'y (num 7) (postfix (id 'x) (postfix (id 'c) (id 'b) (op 'sub)) (op 'add)) (keyword 'with)) (op 'sub)) (keyword 'with)) (keyword 'with)) (substitute 'a (id 'd) (id 'z) (keyword 'with)) (op 'add))) '(x)) |

# What to submit

1. Upload the following code file that contains your answers of HW#2 in your google directory. (Write code for the three tasks in one Racket file which must be runnable.)
   * HW2\_[Student\_id]\_[name].rkt  
     e.g., HW2\_19800179\_JaechangNam.rkt
   * Do not forget to add 'Solve by myself' with additional comments and 'time taken' as well. Please add [contract], [purpose] as comments for all functions you implement including free-ids.

# Due Date

1. 22:00, Oct 10 (Sat) 2020.

# Evaluation (Full mark: 14 points)

* Late submission
  + The number of days delayed \* -2
    - 1 minute to 24-hour delay is considered a one day delay. So please be punctual!
* Missing comments required
  + -0.1 for each
* Test cases that are not passed. (We will not deduct more than 7 points.)
  + -0.2 for each test case
  + For the evaluation, we are going to use our own test cases. Even though your code can pass all test cases, it may not pass our test cases. Please, think of any other test cases for your implementation.

# TA sessions

* Venue: Online
* 20:00~21:15 every Thursday
* Access https://histutor.cafe24.com (Login with your HGU account)
* You may send emails to TAs to seek help from TAs but also use the FAQ below.

# Q&A and FAQ

Want to ask something? Use this first!

<https://docs.google.com/document/d/1A63QWsKeeEfsY3Sidfi_8oui-C0fyn86txnu1OGqJZo/edit>