Principles of Programming Languages, Spring 2016

Assignment 4

General Submission Instruction

1. Submit your answers in a zip file named `id1_id2.zip`, where `id1` and `id2` are the IDs of the students submitting the exercise (use `id.zip` if there is a single student in the group. Also, consider finding someone to work with. Contact the TAs for match-making help, if needed). A template for the archive file is provided with this exercise.
2. Please see each question for its own details.
3. A contract must be included for every implemented procedure.
4. Make sure the '`.rkt`' file is in the correct format (see the announcement about "WXME format" in the course web page).
5. As your code is tested automatically, you must use exact procedure and file names. Again, we recommend using the provided template.
6. Each directory contains a test file with some tests for your code. These tests are meant to help you write a bugless code. Use them! Make sure your code passes all the tests (of course all the regular tests of the evaluators should pass too). *Your code will be checked with another set of tests, so write a more thorough set of tests to make sure your code is correct.* Note that in some folders, the tests are in a nested folder (e.g. "substitution-interpreter" and not to top-level one).

1. Concepts

Submission instructions: Write all answers in the file `ex4_q1.pdf`

Answers to these concept questions must be submitted in the file named `ex4_q1.pdf`.

1. *Value* ADT:
    a. What is the role of the *Value* ADT in the Substitution-Evaluator need? Give an example.
    b. The Environment-Evaluator does not have a Value ADT. Why? Use the example from the previous sub-question to show that it is not needed.
    c. Why is *Procedure* not a value but an ADT of its own?
2. List the advantages and disadvantages of keeping a small language core and a large library of derived expressions.
3. What are the reasons for switching from the Substitution Model to the Environment Model?
4. What is the improvement in the Analyzer over the Environment interpreter? Give an example.
5. What is the purpose of the following condition in the function '`derive`' (in the ASP module):

```
(if (equal? exp derived-exp)
    exp
    (derive derived-exp))
```

Give an example.

6. Procedure and Primitive-Procedure:
   a. Why both ADTs, Procedure and Primitive-Procedure are needed?
   b. Look at the function `apply-primitive-procedure` (on any evaluator). What is the purpose of the `error` part? Give an example of such as error.

2. Environment Model

Submission instructions: Draw the answer in the file `ex4_q2.pdf`

1. Draw an environment diagram for the following computation in **lexical scoping**. Make sure to include:
   1. Static block markers (copy the given code and mark the block markers on it)
   2. Control links and the returned values.

```
> (define a 2)
> (define goo (λ (x)
              (λ (y)
                (/ x y))))
> (define foo (let* ((f (goo a))
                     (g (λ (x) (f x))))
              (λ (x)
                (if (= x 0)
                    x
                    (g x)))))
> (foo (foo 0))
```

2. Draw an environment model for the same computation with **dynamic scoping**.
3. Observe the diagram in (1). Explain the need for two different arrows: environment link and control link.
4. Observe the diagram in (2). Are the control link and environment link the same?

3. Derived Expressions (ASP): `and->if`

Submission instructions: The directory `ex4_q3/` in the submission template contains a copy of the substitution evaluator. You need only modify the file `asp.rkt`. (The rest of the files are there so you can test it more easily).

An '`and`' expression can be written as an '`if`' expression. For example, the expression:

```
(and 'a 'b 3)
```

Is equivalent to the expression:

```
(if 'a
    (if 'b
        3
        #f)
    #f)
```

Write the function ‘`and->if`’ that derives an ‘`and`’ expression to an ‘`if`’ expression:

```
; Signature: and->if(and-exp)
; Type: [<Scheme-exp> -> <Scheme-exp>]
```

Guidelines:

- Add the ‘`and`‘ ADT.
- Modify ‘`derived?` ‘ and ‘`shallow-derive`‘ to support the new syntax.
- Implement ‘`and->if`‘

4. Substitution Model Interpreter `defined?`

Submission instructions: The directory `ex4_q4/` in the submission template contains a copy of the substitution evaluator. Modify it. There is no need to add files.

Add a **special form** `defined?` that is applied on a variable, that returns a Value Boolean whether this variable is bound to a value in the global environment. For example:

```
> (derive-eval '(define x 5))
> (derive-eval '(defined? x))
'(value #t)
> (derive-eval '(defined? y))
'(value #f)

; Signature: defined?(var)
; Type: [Variable -> Value-Boolean-of-the-Evaluator]
; Tests:
;   (derive-eval '(defined? x)) => '(value #f)
;   (derive-eval '(define x #f)) => 'ok
;   (derive-eval '(defined? x)) => '(value #t)
;   (derive-eval '(if (defined? x) x 0) => '(value #f)
```

Guidelines:

- Define and add the needed ADT to the ASP module. Note that it is a special form.
- Modify the core where needed – don’t forget to use the DS ADT.
- **Note**: if a variable has the value false, still ‘`defined?`’ Should return true.

5. Environment Model Interpreter

Submission instructions: The directory `ex4_q5/` in the submission template contains a copy of the environment evaluator. Modify it. There is no need to add files.

Add a **special form** `defined-in-closure?` that is applied on a variable and a closure, that returns whether this variable is defined in the environment of the closure. For example:

```
> (derive-eval '(define foo (lambda (x) (lambda(y) (+ x y)))))
> (derive-eval '(defined-in-closure? x (foo 5)))
#t
> (derive-eval '(defined-in-closure? y (foo 5)))
#f

; Signature: defined-in-closure? (var, closure)
; Type: [<Variable>*<Evaluator-value-Procedure> -> Boolean]
```

Guidelines:

- Add the necessary ADT to the ASP module.
- Add '`defined-in-closure?`' as a special form.

6. Analyzer `and`

Submission instructions: The directory `ex4_q6/` in the submission template contains a copy of the analyzer. Modify it. There is no need to add files.

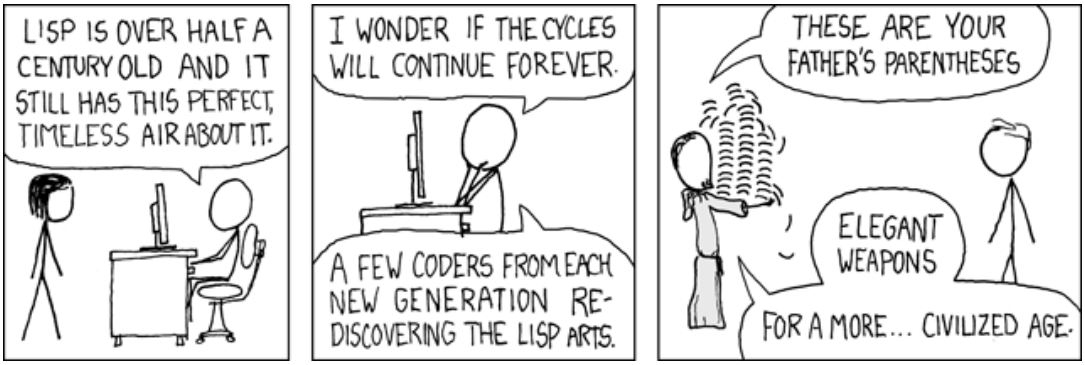
In question 3 you implemented the '`and`' expression as a derived expression. In this question you are asked to implement '`and`' as a **special form** in the analyzer.

Guidelines:

- Modify the ASP.
- Add '`and`' as a special form (This is not mandatory, but it is advised first to implement it as a special form in the evaluator, and only then in the analyzer.).
- Add the function '`analyze-and`' to the core.

Notice

- The parameters should be evaluated one-at-a-time, and not in advance. For example: `(derive-analyze-eval '(and 1 #f (/ 1 0)))` Should return `#f` and **NOT** throw an exception.
- Do not use the derived expression from question 3. It will be checked.


LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.
I WONDER IF THE CYCLES WILL CONTINUE FOREVER.
A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.
THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

בהצלחה!