COMPUTER SCIENCE 320
CONCEPTS OF PROGRAMMING LANGUAGES

Problem Set 6: A Small Interpreter (Part II)

OUT: FRIDAY, FEBRUARY 24, 2006
DUE: ***4:59 PM*** ON FRIDAY, MARCH 4, 2006

There are 5 problems in this set, each worth 20 points. The harder problems are marked with a single $*$ (average difficulty) or two $**$ (higher-than-average difficulty). For the easy points, start with the unmarked problems.

$*$ **Problem 4** *(From Problem Set 5 – 20 points)* Extend the definition of `subst` in `subst.scm` to correctly perform substitutions into valid expressions in INTEX augmented with `sigma`.

Problem 5 *(From Problem Set 5 – 20 points)* Extend the definition of `subst-eval` in `subst-eval.scm` to correctly evaluate valid expressions in INTEX augmented with `sigma`.

1 Getting Familiar With INTEX+bind

We provide you with several Scheme files in the directory `Scheme-code-for-INTEX+bind`. You will also need the files in the directory `Scheme-utilities`, already provided with Problem Set 5. For easy downloading, we made a "tar" file of this directory, which can be restored to its original contents by issuing following Unix command:

```
tar -xvf Scheme-code-for-INTEX+bind.tar
```

To run and test some of the programs we provide you, we suggest that you open an Emacs window under the directory `Scheme-code-for-INTEX+bind`. After launching your Scheme interpreter, within the Emacs window, you can experiment with the INTEX+bind interpreter by executing the following Scheme code:

```
;; Load several examples of environments and expressions.
(load "examples")

;; Load several examples of programs written in INTEX+bind.
(load "intex+bind-examples")

exp2

(free-vars exp2)

(binding-vars exp2)

(bound-vars exp2)

(rename exp2 (free-vars exp2))

env1

(subst env1 exp2)
```

```
;; Perform a substitution in an expression.
(subst (env-make '(a c)
                 (map make-literal '(3 5)))
       '(bind c (+ a b) (* c d)))
```

Make sure you understand how the functions "`rename`" and "`subst`" work. Proceed further by executing the following:

```
;; 4 different modes of evaluation
calc

(subst-run-cbv calc '(2 3))

(subst-run-cbn calc '(2 3))

(env-run-cbv calc '(2 3))

(env-run-cbn calc '(2 3))
```

2 De-Sugaring

We want to extend INTEX+`bind` with two new binding mechanisms, `bindseq` and `bindpar`. The syntax of `bindseq` has the form:

$$\begin{array}{l}(\textbf{bindseq}\ (\ (var\_1\ defn\_1)\\ \qquad\qquad (var\_2\ defn\_2)\\ \qquad\qquad \ldots\\ \qquad\qquad (var\_n\ defn\_n)\ )\ body)\end{array}$$

The syntax of `bindpar` is identical to the previous one, except that keyword "`bindpar`" is substituted for "`bindseq`":

$$\begin{array}{l}(\textbf{bindpar}\ (\ (var\_1\ defn\_1)\\ \qquad\qquad (var\_2\ defn\_2)\\ \qquad\qquad \ldots\\ \qquad\qquad (var\_n\ defn\_n)\ )\ body)\end{array}$$

The semantics of `bindseq` and `bindpar` are similar to the semantics of `let*` and `let`, respectively, in Scheme.

In the case of `bindseq`, the n bindings are carried out in *sequence*, which allows $defn\_2$ to depend on var_1, then $defn\_3$ to depend on var_1 and var_2, then $defn\_4$ to depend on var_1, var_2 and var_3, etc.

By contrast, in the case of `bindpar`, the n bindings are carried out in *parallel*, so that $defn\_2$ cannot depend on var_1, nor can $defn\_3$ depend on var_1 and var_2, nor can $defn\_4$ depend on var_1, var_2 and var_3, etc.

There are two approaches to extending our interpreter for INTEX+`bind` to include `bindseq` and `bindpar`. The first approach is to make appropriate changes in each of the modules of the INTEX+`bind` system.

The second approach is to write "pre-processing" functions to de-sugar every `bindseq` expression and every `bindpar` expression into expressions that only use `bind`. This approach also requires to add a case for each of `bindseq` and `bindpar` in the package `abstract-syntax.scm`.

Problem 6. *(20 points)* Extend `abstract-syntax.scm` for the language INTEX+`bind` so that programs now containing `bindseq` and `bindpar` are also recognized as valid.

Note that you have to define an appropriate constructor and appropriate selectors for each of `bindseq` and `bindpar`. In the case of `bindseq`, for example, you have to define:

($\mathbf{make\text{-}bindseq}$ $names$ $defns$ $body$),
which returns a `bindseq` node (in the abstract syntax tree of an expression).

($\mathbf{bindseq\text{-}names}$ $bindseq\text{-}node$),
which returns the list of names in $bindseq\text{-}node$.

($\mathbf{bindseq\text{-}defns}$ $bindseq\text{-}node$)
which returns the list of definitions in $bindseq\text{-}node$.

($\mathbf{bindseq\text{-}body}$ $bindseq\text{-}node$)
which returns the body of $bindseq\text{-}node$.

($\mathbf{bindseq?}$ $node$)
which returns `#t` if $node$ is a `bindseq` node, and `#f` otherwise.

* **Problem 7.** *(20 points)* Write a de-sugaring function for programs written in INTEX+`bind`+`bindseq`. The structure of your function should be as shown below.

```
(define desugar-program
  (lambda (pgm)
    (make-program (program-formals pgm)
                  (desugar (program-body pgm)))))

(define desugar
  (lambda (exp)
    (cond
      ;; case-analysis of kernel expressions
      ((literal? exp) exp)
        .
        .
        .

      ;; case-analysis of bindseq expressions:
      ((bindseq? exp)
        .
        .
```

```
        .  )

      (else (error "DESUGAR: unrecognized expression -- " exp)))))
```

Having appropriately adjusted `abstract-syntax.scm` in Problem 6, and written the de-sugaring function in Problem 7, there is one more change to be made in each of the files: `subst-run-cbv`, `subst-run-cbn`, `env-run-cbv` and `env-run-cbn`. Namely, in the case of `subst-run-cbv` for example (and similarly in the 3 other files), it should now read:

```
(define subst-run-cbv
  (lambda (pgm ints)
    (literal-value
      (subst-eval-cbv (subst* (map make-literal ints)
                              (program-formals pgm)
                              (desugar (program-body pgm)))))))
```

** **Problem 8.** *(20 points)* Repeat Problem 7 for programs written in in INTEX+`bind`+`bindseq`+`bindpar`. You need to extend the function `desugar` to handle the case of `bindpar` expressions. *Hint*: The case of `bindpar` is more subtle than the case of `bindseq`. You will need to do some renaming of binding and bound variable-occurrences in general.