Sémantique des Langages de Programmation (SemLP)
DM : little-λ-ref

We consider a simple extension to the monadic *call-by-value* λ-calculus of the lecture notes (Chapter ...) with :

1. Natural values,
2. Boolean values,
3. an *if-then-else* conditional expression, and
4. references.

The syntax of this extended language is given in Figure 1. The symbol $\oplus$ represents a binary natural operator, $\otimes$ represents a natural binary comparator, and $\oslash$ represents a boolean binary operator. The special value $()$ is named *unit*, and it is the only value of type Unit.

$$
\begin{array}{lcll}
x & \in & \mathcal{V}ar & \text{(var. names)} \\
n & \in & \mathbb{N} & \text{(naturals)} \\
b & \in & \{\mathsf{true}, \mathsf{false}\} & \text{(booleans)} \\
p & \in & \mathcal{R}ef & \text{(references)} \\
\\
V & ::= & \lambda x.\ M \mid n \mid b \mid p \mid () & \text{(values)} \\
\\
M & ::= & x \mid V \mid @(M, M) & \text{(terms)} \\
 & \mid & M \oplus M \mid\ -M \mid M \otimes M \mid M \oslash M \mid \neg M & \\
 & \mid & \mathsf{ref}\ M \mid !M \mid M := M & \\
 & \mid & \mathsf{let}\ x = M\ \mathsf{in}\ M & \\
 & \mid & \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M & \\
\\
E & ::= & @(E, M) \mid @(V, E) & \text{(ev. contexts)} \\
 & \mid & E \oplus M \mid V \oplus E \mid\ -E \mid \ldots & \\
 & \mid & \mathsf{ref}\ E \mid !E \mid E := M \mid V := E & \\
 & \mid & \mathsf{let}\ x = E\ \mathsf{in}\ M \mid \mathsf{if}\ E\ \mathsf{then}\ M\ \mathsf{else}\ M &
\end{array}
$$

FIGURE 1 – Syntax for little-λ-ref.

Exercice 1 : Implementing little-λ-ref

1. Give an operational semantics similar to that of Chapter 12 of the lecture notes to little-λ-ref. Notice that you will need the *heap* to give semantics to references. [1]

2. Implement in you favorite programming language an interpreter for little-λ-ref. The input to the interpreter is an expresion in the *abstract syntax* of the language. You are not required to provide a parser. [2]

1. The evaluation contexts are given. Identifying the redexes is left as part of the exercise.
2. Providing a parser can be considered for extra credit.

Exercice 2 : Typing little-λ-ref

Provide and implement a type inference algorithm for little-λ-ref. You will need to use the type constructor Ref provided in Chapter 12 of the lecture notes. The value () is the only value with type Unit.

Exercice 3 : Transforming little-λ-ref

Implement the chain of program transformations considered in Chapter 10 of the lecture notes for little-λ-ref. You will need to consider the case of conditional expressions as well as nested arithmetic, boolean and comparator operators.

Exercice 4 : Testing little-λ-ref

Provide a test-suite showing that each of the transformations preserves the semantics of the source level program. That is, provide a set of "relevant" well-typed tests, for each of which (eg. M) you will show that :

$$(M, \epsilon) \to^* (V, h) \Rightarrow (\mathcal{C}_h \circ \mathcal{C}_{cc} \circ \mathcal{C}_{vm} \circ \mathcal{C}_{cps}(M), \epsilon) \to^* (V, h)$$

where $\circ$ represents functional composition. [3]

Can you observe differences in the performance of the programs before and after the transformations ? Can these transformations be considered as optimizations ?

3. The format for reporting the results of the tests is left open. It is expected that the results be logged or printed, and that appropriate error messages be emitted when a test fails.