Implementação de Linguagens

Departamento de Ciência de Computadores
Faculdade de Ciências – Universidade do Porto
6th June 2023
Duration: 2 hours

Part I

(1) Consider a variation of the Fun language introduced in the lectures with just lambda-abstractions, applications, boolean values and conditional expressions.

$$
\begin{array}{rcll}
e & ::= & x & \text{variables} \\
 & | & \lambda x.\, e & \text{lambda abstractions} \\
 & | & e_1\ e_2 & \text{application} \\
 & | & \mathsf{true} & \text{constants} \\
 & | & \mathsf{false} & \\
 & | & \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 & \text{conditionals} \\
 & & & \\
v, u & ::= & \lambda x.\, e \mid \mathsf{true} \mid \mathsf{false} & \text{weak normal forms}
\end{array}
$$

(a) (10%) Consider the big-step evaluation rules $e \Downarrow v$ for lambda abstractions and applications:

$$\frac{}{\lambda x.\, e \Downarrow \lambda x.\, e}\ \textsc{Abs} \qquad \frac{e_1 \Downarrow \lambda x.\, e' \qquad e_2 \Downarrow v \qquad e'[v/x] \Downarrow u}{(e_1\ e_2) \Downarrow u}\ \textsc{App}$$

Define the remaining rules for booleans and if-then-else expressions.

(b) (15%) Consider now the translation scheme from expressions to instruction of a SECD machine and the transitions from configurations $(s, e, c, d, m) \longrightarrow (s', e', c', d', m')$:

$$
\begin{array}{l}
\mathsf{compile}\ x\ sym = [\mathsf{LD}\ k],\ \text{where } k = \mathsf{elemIndex}\ x\ sym \\
\mathsf{compile}\ (\lambda x.\, e)\ sym = [\mathsf{LDF}\ (\mathsf{compile}\ e\ (x : sym) +\!\!+ [\mathsf{RTN}])] \\
\mathsf{compile}\ (e_1\ e_2)\ sym = \mathsf{compile}\ e_1 +\!\!+ \mathsf{compile}\ e_2 +\!\!+ [\mathsf{AP}]
\end{array}
$$

$$
\begin{array}{ll}
(s, e, \mathsf{LD}\ k : c, d, m) \longrightarrow (v_k : s, e, c, d, m), & \text{where } e = v_0 : v_1 : \ldots : v_k : e' \\
(s, e, \mathsf{LDF}\ c' : c, d, m) \longrightarrow (a : s, e, c, d, m'), & \text{where } a \notin dom(m), m' = m[a \mapsto (c', e)] \\
(v : a : s, e, \mathsf{AP} : c, d, m) \longrightarrow ([\,], v : e', c', (s, e, c) : d, m) & \text{where } m[a] = (c', e') \\
(v : s, e, \mathsf{RTN} : c, (s', e', c') : d, m) \longrightarrow (v : s', e', c', d, m) &
\end{array}
$$

Define extra instructions, compilation scheme and transitions for handling boolean constants and if-then-else expressions. Explain your answer.

(2) (15%) Translate the following Haskell definition into the Core intermediate language. Justify your answer.

```
init :: [a] -> [a]
init [x] = []
init (x:xs) = x : init xs
```

(3) (10%) The Core and STG intermediate languages expose unboxed primitive types. Show how this feature allows optimization of the following Haskell function. Justify your answer.

```
fact :: Int -> Int
fact 0 = 1
fact n = n*fact (n-1)
```

Part II

(4) (20%) With the help of simple diagrams, describe how the "Mark and Sweep" algorithm works. If the blocks claimed by the garbage collector (implementing the aforementioned algorithm) have several different sizes and are stored in a single list, what impact can the reuse of this space have on the performance of the application? Suggest a more efficient solution.

(5) (15%) In "Generational Garbage Collectors" what is the criterion usually used to promote objects between generations? Dividing the heap into regions with different generations of objects allows the use of different algorithms to manage those regions. How can this be advantageous in terms of performance?

(6) (15%) Give three examples of code optimizations implemented by the Java Virtual Machine's JIT compiler. Note that, in this case, the optimized code may not be exclusively bytecode, it can also be binary code previously generated by the JIT compiler. In terms of performance, what is the advantage of this approach?