Interpretação e Compilação de Linguagens– 2016-2017
Interpretation and Compilation of Programming Languages

Exam January, 9, 2017

Author: João Costa Seco
Notes: The exam is open book. Students can use any (individual) printed material that each student brings along. The exam has a maximum duration of 2h30.

Q-1 [7 val.] This question is about the definition of an abstract syntax and the operational semantics for a functional programming language with evaluated lists. Consider the programming language, called `pipes`, with the concrete syntax given by the following grammar:

$$E ::= num \mid E_1\texttt{*}E_2 \mid \mathsf{fun}\ x \Rightarrow E \mid x \mid [E_1\ ..\ E_2] \mid E_1\texttt{>>}\mathsf{map}(E_2) \mid E_1\texttt{>>}\mathsf{get} \mid E_1\texttt{>>}\mathsf{next}$$

The language comprises the base constructs for:

- **integer literals** (num), and their usual operations, represented here by operation $E_1 * E_2$;
- **functions** as first-class values ($\mathsf{fun}\ x \Rightarrow E$), and identifiers ($x$) introduced as parameters.
- finite **lists**, with the constructor ($[E_1\ ..\ E_2]$), whose result is the immutable list of consecutive integer values between the denotation of expression E_1 and the denotation of E_2 (inclusive).
- a **map** expression ($E_1\texttt{>>}\mathsf{map}(E_2)$) that iterates the list denoted by expression E_1, applying a function denoted by expression E_2 to every element of the list. The denotation of the expression is the list of all result values.
- the expression $E_1\texttt{>>}\mathsf{get}$, that denotes the first element of the list denoted by expression E_1.
- and the expression $E_1\texttt{>>}\mathsf{next}$ that denotes the list starting from the second element of the list denoted by expression E_1.

The semantics of the language follows a static resolution of names, and a *call-by-value* evaluation strategy.
Consider as an example the programming language `pipes`, with the integer value `6` as denotation:

```
[1..10]>>map(fun x=>2*x)>>next>>next>>get
```

a) [1 val.] Define the abstract syntax of all **list related operations** in language `pipes` by means of an abstract data type, defined by set of (abbreviated) Java classes and interfaces.

b) [1 val.] Define the set of values of language `pipes` by means of an abstract data type, defined by a set of (abbreviated) Java classes and interfaces.

c) [5 val.] Define the operational semantics for the **list related operations** in the language `pipes` by means of a method named **eval** of the Java classes defined in the previous question.

Q-2 [**6 val.**] This question is about the definition of the type system for language `pipes`. To answer the following questions you may use abstract data types, defined by a set of Java classes and interfaces, and the corresponding methods using Java Code.

a) [**2 val.**] **Define** the set of types used to type programs of language `pipes`.

b) [**3 val.**] **Define** the type system of language `pipes` by means of a **typecheck** function, for all **list related operations**.

c) [**1 val.**] **Enumerate** the execution errors that may occur during the execution of a program written in language `pipes`, according to the semantics defined in question **Q-1**. **Indicate and justify** which execution errors may be prevented by the type system, and those that cannot.

Q-3 [**2 val.**] This question is about the definition and use of closures and corresponding environments. It is know that the standard declaration of an identifier can be encoded as a function call and corresponding definition. Consider the following encoding:

$\textsf{decl}\ x = E_1\ \textsf{in}\ E_2 \triangleq (\textsf{fun}\ x \Rightarrow E_2)(E_1)$

Note that `pipes` includes closures, but it does not include the standard function call expression.

a) [**1 val.**] **Define** a language encoding to introduce function call in language `pipes`.

b) [**1 val.**] Consider the following example written in the language `pipes`, extended with declaration and function call. Here, a list of integer values is first mapped to a list of functions and mapped again to a list of integer values.

```
decl f = decl i = 10 in fun x => x(i) in
  (fun z => [1..z]>>map(fun x=>fun y=>x*y)>>map(f)>>next>>get)(10)
```

Present the diagram of the evaluation environment for expressions `x*y` and `x(i)`. You may pick one element of the list to illustrate.

Q-4 [**2 val.**] Mainstream languages are being extended to deal with streams and lists, by promoting lazy evaluation as a mean to implement infinite lists. For instance, the following example written in `pipes`, extended with a filter operation, denotes the list of all even integer numbers.

```
[1..]>>filter(fun x=>x%2==0)
```

Discuss the extensions and modifications to the language `pipes` necessary to implement lazy evaluation and infinite lists.

Q-5 [**2 val.**] This question is about the compilation of programs in language `pipes`. **Consider** the supporting Jasmin class `Node` listed below needed in a type preserving compilation procedure for the example above.

```
.class Node
.super java/lang/Object
.field public value I
.field public next LNode;
...
```

a) [**2 val.**] **List** the set of instructions that results from translating to Jasmin assembly code the following `pipes` expression: `[1..5]>>next>>get`.

Hint: consider building lists as linked lists from the last element to the first.