Interpretação e Compilação de Linguagens– 2017-2018
Interpretation and Compilation of Programming Languages

Final Test — December, 13, 2017 (17h-19h)

Author: João Costa Seco
Notes: The test is closed book with the exception of a single handwritten (cheat) sheet.

Q-1 **[6 val.]** This question is about the definition of the abstract syntax and the operational semantics for a programming language. Consider an imperative programming language, called `Nil`, with state variables, closures, and a special nil value.

$$E ::= num \mid E_1{+}E_2 \mid \mathsf{var} \mid \mathsf{nil} \mid {*}E \mid E_1 := E_2 \mid \mathsf{decl}\ x = ?E_1\ \mathsf{in}\ E_2 \mid x \mid (x) \Rightarrow E \mid E_1(E_2) \mid \mathsf{ifzero}\ E_1\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3$$

The language comprises the following constructs. **Integer literals** (num), and their corresponding operations, represented here by operation $E+E$. The **imperative constructs**, where var creates a heap allocated variable and denotes its reference. The dereferentiation $*E$ and assignment $E_1 := E_2$ expressions according to the usual semantics. The initial value of a heap allocated variable is nil. The nil value, which represents a special value with no special operation other that the test decl $x = ?E_1$ in E_2, which evaluates the declaration body (expression E_2) only if expression E_1 is different from nil. Identifier x is bound to the value denoted by E_1 as usual. If E_1 denotes nil, the whole expression denotes nil. Additionally, consider expressions for **identifiers** (x) and **anonymous functions** $(x) \Rightarrow E$, and function call according to the usual semantics. Also, we introduce a conditional expression on integers, ifzero E_1 then E_2 else E_3. The remaining aspects of the language semantics follow the semantics presented in the course lectures.
Consider the example written in the programming language `Nil`:

```
decl f = (g)=>(x)=>(y)=>if *x then 0
                                else y := *y * *x;
                                     x := *x - 1;
                                     g(g)(x)(y)
in decl x = var
            y = var
   in x := 5;
      y := 1;
      decl h =? f in h(h)(x)(y)
```

Sequence and regular declaration expressions are used in the examples, and can be obtained by encoding on other expressions, as explained in the course lectures.

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

b) **[4 val.]** **Define** the operational semantics of language `Nil`, for expressions $E_1{+}E_2$, nil, $E_1 := E_2$, decl $x = ?E_1$ in E_2, and $E_1(E_2)$, by means of a method **eval** of the corresponding AST class.

c) **[1 val.]** **State** the denotation (value) of the example above according to the semantics defined in **Q-1b)**

Q-2 **[6 val.]** This question is about the definition of a type system for language `Nil`. Consider the language `Safenil`, obtained by extending `Nil` with type annotations in the definition of functions and in the allocation of variables. To answer the following questions you should use abstract data types, defined by a set of Java classes and interfaces, and the corresponding methods using Java Code. The goal of this question is to statically **avoid execution errors due to null values**.

Consider the language types represented by `IType`, and the types `IntType`, `FunT(IType,IType)`, `Nullable(IType)`, `Ref(IType)`. Consider the example with type `Nullable(IntType)`:

```
decl x = var<int> in decl y =? *x in y+1
```

And the following ill-typed example

```
decl x = var<int>  f = (z:int)=>z+1 in f(*x)
```

a) **[4 val.]** **Define** the type system cases for expressions E_1+E_2, var, $*E$, decl $x =?E_1$ in E_2, and $E_1(E_2)$, by means of a **typecheck** method in the AST classes.

b) **[2 val.]** **Explain** if the example expression in question **Q-1** is well typed, according to the type semantics defined in question **Q-2a**. **Exhibit a modified and well typed expression** if that is not the case (use `declrec` recursive declarations).

Q-3 **[6 val.]** This question is about the compilation of programs using closures, references, null values, and nullable variables. Consider the following program written in the `Safenil` language and the compilation schema introduced in the course lectures. Consider the well-typed and type annotated (**Q-2b)**) variant of the example given in **Q-1**.

a) **[2 val.]** Explain the generated supporting structures (Jasmin class skeletons and interfaces) necessary to compile the referred example (**Q-1**).

b) **[4 val.]** **List** the set of instructions that would result from translating the final expression `decl h =? f in h(x)(y)` to the Jasmin assembly language.

Q-4 **[2 val.]** This question is about object encodings. Explain what is the advantage of using object encodings to support object-oriented features. Give an example of a well-typed functional encoding for a class **Point2D** with coordinates x and y, a translate method, and a distance method between two points.

```
class Point2D {
  int x,y;
  void translate(int dx, int dy) { x += dx; y += dy; }
  double distance(Point q) { return Math.sqrt((x-q.x)*(x-q.x)+(y-q.y)*(y-q.y)); }
}
```