# Interpretação e Compilação – 2013-2014
# Interpretation and Compilation of Programming Languages

Midterm April 9, 2014

Notes: The test is open book. Students can use any (individual) printed material that each one brings along. The test has a duraction of 1h30.
PS: This document was editted in order to correct typos and ambiguities in the original.

---

**Q-1** **[5 val.]** This question is about the definition of an abstract syntax for a programming language. You are certainly aware that the vast majority of software systems available in the web are based on interchangeable data formats, as the base for interoperability and extensibility. One of such formats is XML. In order to safely manipulate XML values inside a programming language we need special language constructs.

Consider the following programming language, called `XiMpL`, with the concrete syntax given by the following grammar:

$$
\begin{array}{rcl}
E & ::= & x \mid num \mid string \mid E_1 + E_2 \\
& \mid & \mathsf{decl}\ x = E_1\ \mathsf{in}\ E_2 \mid \langle label \rangle E_1 ... E_n \langle / label \rangle \mid E_1 \texttt{@} E_2 \mid \parallel E \parallel \\
& \mid & \mathsf{if}\ E_1\ \mathsf{has}\ label\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3 \\
& \mid & \mathsf{for}\ x\ \mathsf{in}\ E_1\ \mathsf{with}\ y = E_2\ \mathsf{in}\ E_3
\end{array}
$$

The language comprises the base constructs for: **integer literals** (*num*) and their usual operations on integers, represented here by operation $E + E$; **string literals** (*string*); **identifier** use ($x$) and **declaration** $\mathsf{decl}\ x = E_1\ \mathsf{in}\ E_2$. The language also manipulates XML values and terms. The **XML constructor** $\langle label \rangle E_1 ... E_n \langle / label \rangle$, containing a sequence of expression representing the child nodes, and denotes an XML value. The concatenation operation $E_1 \texttt{@} E_2$ where the XML value denoted by $E_1$ is extended with a new child node denoted by $E_2$; The selection operation $\parallel E \parallel$ denotes the child node of an XML value. Notice that this operation is only defined on a node with a single child node. Also, we have the inspection operation $\mathsf{if}\ E_1\ \mathsf{has}\ label\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3$, whose denotation is the denotation of $E_2$, in the case that $E_1$ denotes an XML value with label *label*, and the denotation of $E_3$ otherwise. Finally, we have the iteration operation $\mathsf{for}\ x\ \mathsf{in}\ E_1\ \mathsf{with}\ y = E_2\ \mathsf{in}\ E_3$ that iterates over the inner nodes of the XML value denoted by expression $E_1$, and whose denotation is given by the expression $E_3$ in the last iteration. On each iteration, $x$ denotes an element of the list and $y$ denotes the value of $E_3$ in the previous iteration, or the denotation of expression $E_2$ in the case of the first iteration.
Consider the example written in the programming language `XiMpL`:

```
decl l1 = <line>"Line 1"</line> in
decl l2 = <line>"Line 2"</line> in
decl doc = <doc><title>"Title"</title> l1 l2 </doc> in
for x in doc with y = <html></html> in
  if x has title then y @ <h1>||x||</h1> else
  if x has line then y @ <p>||x||</p> else y @ x
```

**a)** **[2 val.]** **Define** the abstract syntax of language `XiMpL` by means of an abstract data type, defined in either ML, Haskell, or a set of Java classes and interfaces.

**b)** **[2 val.]** **Define** the set of values of language `XiMpL` by means of an abstract data type, defined in either ML, Haskell, or a set of Java classes and interfaces.

**c)** **[1 val.]** The primitive operations of a programming language can be sometimes encoded inside the language. In this case, the selection operation, which is defined only for XML values with a single child node. **Define** the selection operation $\parallel E \parallel$ using other constructs of the same language.

**Q-2** [**8 val.**] This question is about the definition of the operational semantics for language `XiMpL`.

**a)** [**6 val.**] **Define** the operational semantics of language `XiMpL` by means of a method **eval** in each of the classes defined in question **Q-1a**, or a recursive function in either ML or Haskell.

**b)** [**1 val.**] **State** the denotation of the example expression in question **Q-1**, according to the semantics defined in question **Q-2a**.

**c)** [**1 val.**] **Enumerate** the execution errors that may occur during the execution of a program written in language `XiMpL`, according to the semantics defined in question **Q-2a**.

---

**Q-3** [**7 val.**] This question is about the typing semantics of language `XiMpL`, and how it can be used to avoid execution errors. Consider the abstract representation of the types necessary to type language `XiMpL`, represented by the following abstract data type written in `OCaml`:

```
type ty = IntType | StringType | XMLNode
```

**a)** [**4 val.**] **Define** the typing semantics of language `XiMpL`, for the constructs enumerated below, by means of a method **typecheck** in each of the Java classes of question **Q-1a**, or by means of a recursive function in ML or Haskell.

(a) $x$
(b) $\mathsf{decl}\ x = E_1\ \mathsf{in}\ E_2$
(c) $\langle label \rangle E_1 ... E_n \langle / label \rangle$
(d) $\mathsf{if}\ E_1\ \mathsf{has}\ label\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3$
(e) $\mathsf{for}\ x\ \mathsf{in}\ E_1\ \mathsf{with}\ y = E_2\ \mathsf{in}\ E_3$

Take the following extra restrictions that must be imposed by type checking:

(a) XML constructors with more than one child node expression, must have all child nodes of XML type.
(b) An XML constructor can only be iterated if its child nodes are of XML type.
(c) The contents of an XML value ($\| E \|$) can only be inspected if it has a single element.

**b)** [**1 val.**] **Enumerate** which runtime errors, presented in question **Q-2c**, can be avoided by the type system just defined.

**c)** [**1 val.**] **Check** if the example of question **Q-1** is well typed. **Present** the type of the expression, or justify and present an appropriate correction.

**d)** [**1 val.**] **Present** the typing environment of expression `y @ x`.