# DD2481 - Principles of Programming Languages
# Lab 2 beta

Andreas Lindner

10 April 2018

## 1 Goal

The goal of this second lab is the extension of lab 1 with the untyped lambda calculus using the call-by-value evaluation strategy seen in the lecture. We provide an extended code template to enable parsing of the extended syntax. Additionally, this code template (will) contain a simple test infrastructure for you to extend.

## 2 Syntax and Semantics

The syntax and small-step evaluation rules extending the ones from lab 1 follow.

Syntax extensions:

$$
\begin{array}{rcll}
t & ::= & & \textit{terms}: \\
 & | & \ldots & \textit{(see lab 1)} \\
 & | & \lambda x.\, t & \textit{abstraction} \\
 & | & t\ t & \textit{application}
\end{array}
$$

$$
\begin{array}{rcll}
v & ::= & & \textit{values}: \\
 & | & \ldots & \textit{(see lab 1)} \\
 & | & \lambda x.\, t & \textit{abstraction value}
\end{array}
$$

Evaluation extensions:

E-App1
$$\frac{t_1 \mid \mu \rightarrow t_1' \mid \mu'}{t_1\ t_2 \mid \mu \rightarrow t_1'\ t_2 \mid \mu'}$$

E-App2
$$\frac{t_2 \mid \mu \rightarrow t_2' \mid \mu'}{v_1\ t_2 \mid \mu \rightarrow v_1\ t_2' \mid \mu'}$$

E-AppAbs
$$(\lambda x.\ t_{12})\ v_2 \mid \mu \rightarrow [x \mapsto v_2]t_{12} \mid \mu$$

## 3 Task

As in lab 1, the interpreter should be based directly on the small-step evaluation rules. In order to convince yourself that you implemented the rules correctly you will develop a small test suite containing test programs and expected outputs.

Notice the syntax difference for input programs and the abstract syntax and semantics description above. A “$\lambda$” above corresponds to a “\” character in our interpreter input files.

## 4 AST extension

The extended parts of the AST, which are also provided to you, are implemented in Scala as follows (file “parser/AST.scala”):

```
trait AST extends Positional
...
case class LamExp(id: String, e: AST) extends AST
case class AppExp(e1: AST, e2: AST) extends AST
```

## 5 Implementation

Your implementation will be in the `step` method of the `Interpreter` singleton object like in lab 1. It is located in the same file. Please refer to lab 1 for the template description.

To get started, merge the new skeleton with your existing code. For implementing the evaluation rules for lab 2, it might be helpful to start by implementing a function for determining the set of free variables in an expressions and then a function for computing a substitution in an expression.

The set of free variables is useful for identify captured variables during substitution and it can be simply computed as a list.

For this purpose the template already contains the two following method stubs:

```
/* function for determining the free variables of an expression */
def freevars (x: AST) : List[String] = errFun("implement me", x)

/* function for carrying out a substitution */
def subst (x: String, s: AST, t: AST):AST = errFun("implement me", t)
```

The course book ”Types and Programming Languages” might be a useful reference for this task.

## 6 Testing

The scala build tool `sbt` offers options for automatically rerunning the program once code changes. This also works for other files once they are specified in the “build.sbt” file using the `watchSources` option.

Using the template directly for example, you may develop interpreter input code in a file “src/input.sint” and then use the command `run src/test.sint`. Whenever you change scala code or the input in this file, your program will be compiled an rerun.

Furthermore, you will extend a simple test case framework we are going to provide later today.

## 7 Example

A lambda code example implementing integer subtraction using recursion will be provided later today as well.

Good luck!

## 8 Changelog

| Version | Contribution | Author | Contact |
|---|---|---|---|
| 0.1 | Derived from lab 1 | Philipp Haller | phaller@kth.se |
| 0.2 | Development for VT2018 | Andreas Lindner | andili@kth.se |