Elements of Programming Languages
Tutorial 1: Abstract syntax trees, evaluation and typechecking
Week 3 (October 5–9, 2015)

Starred exercises ($\star$) are more challenging. Please try all unstarred exercises before the tutorial meeting.

1. **Pattern matching.** For this problem, you should use the Scala definition of $\mathsf{L}_{\mathsf{Arith}}$ abstract syntax trees presented in the lectures:

```
abstract class Expr
case class Num(n: Integer) extends Expr
case class Plus(e1: Expr, e2: Expr) extends Expr
case class Times(e1: Expr, e2: Expr) extends Expr
```

   (a) Write a Scala function `evens[A]: List[A] => List[A]` that traverses a list and returns all of the elements in even-numbered positions. For example, `evens(List('a','b','c','d','e','f')) = List('a','c','e')`. The solution should use pattern-matching rather than indexing into the list.

   (b) Write a Scala function `allplus: Expr => Boolean` that traverses a $\mathsf{L}_{\mathsf{Arith}}$ term and returns `true` if all of the operations in it are additions, `false` otherwise. (For this problem, you may want to use the Scala Boolean AND operation `&&`.)

   (c) Write Scala function `consts: Expr => List[Int]` that traverses a $\mathsf{L}_{\mathsf{Arith}}$ expression and constructs a list containing all of the numerical constants in the expression. (For this problem, you may want to use the Scala list-append operation `++`.)

   (d) Write Scala function `revtimes: Expr => Expr` that traverses a $\mathsf{L}_{\mathsf{Arith}}$ expression and reverses the order of all multiplication operations (i.e. $e_1 \times e_2$ becomes $e_2 \times e_1$).

   (e) ($\star$) Write a Scala function `printExpr: Expr => String` that traverses an expression and converts it into a (fully parenthesised) string. For example:

```
scala> printExpr( Times(Plus(Num(1), Num(2)), Times(Num(3), Num(4))))
res0: String = ((1 + 2) * (3 * 4))
```

2. **Evaluation derivations.**

   Recall the evaluation rules covered in lectures:

   $\boxed{e \Downarrow v}$

$$\frac{}{v \Downarrow v} \qquad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{e_1 + e_2 \Downarrow v_1 +_{\mathbb{N}} v_2} \qquad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{e_1 \times e_2 \Downarrow v_1 \times_{\mathbb{N}} v_2}$$

$$\frac{e_1 \Downarrow v \quad e_2 \Downarrow v}{e_1 == e_2 \Downarrow \texttt{true}} \qquad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad v_1 \neq v_2}{e_1 == e_2 \Downarrow \texttt{false}}$$

$$\frac{e \Downarrow \texttt{true} \quad e_1 \Downarrow v_1}{\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Downarrow v_1} \qquad \frac{e \Downarrow \texttt{false} \quad e_2 \Downarrow v_2}{\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Downarrow v_2}$$

Write out derivation trees for the following expressions:

(a) 6×9

(b) $3 \times 3 + 4 \times 4 == 5 \times 5$

(c) $\texttt{if}\ 1 + 1 == 2\ \texttt{then}\ 2 + 3\ \texttt{else}\ 2 * 3$

(d) $\texttt{if}\ 1 + 1 == 2\ \texttt{then}\ 3\ \texttt{else}\ 4 + 5$

3. **Typechecking derivations.** Recall the typechecking rules covered in lectures:

$\boxed{\vdash e : \tau}$

$$\frac{n \in \mathbb{N}}{\vdash n : \texttt{int}} \qquad \frac{\vdash e_1 : \texttt{int} \quad \vdash e_2 : \texttt{int}}{\vdash e_1 + e_2 : \texttt{int}} \qquad \frac{\vdash e_1 : \texttt{int} \quad \vdash e_2 : \texttt{int}}{\vdash e_1 \times e_2 : \texttt{int}}$$

$$\frac{b \in \mathbb{B}}{\vdash b : \texttt{bool}} \qquad \frac{\vdash e_1 : \tau \quad \vdash e_2 : \tau}{\vdash e_1 == e_2 : \texttt{bool}} \qquad \frac{\vdash e : \texttt{bool} \quad \vdash e_1 : \tau \quad \vdash e_2 : \tau}{\vdash \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau}$$

Write out typing derivations for the following judgments:

(a) $\vdash 6 \times 9 : \texttt{int}$

(b) $\vdash \texttt{if}\ 1 + 1 == 2\ \texttt{then}\ 3\ \texttt{else}\ 4 + 5 : \texttt{int}$

4. **($\star$) Nondeterminism.** Suppose we add the following construct $e_1 \Box e_2$ to L_{If}:

$$\begin{array}{rcl} e & ::= & e_1 + e_2 \mid e_1 \times e_2 \mid n \in \mathbb{N} \\ & \mid & \texttt{true} \mid \texttt{false} \mid e_1 == e_2 \mid \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \\ & \mid & e_1 \Box e_2 \end{array}$$

Informally, the semantics of $e_1 \Box e_2$ is that we evaluate either e_1 or e_2 nondeterministically. To model this we extend the evaluation rules as follows:

$\boxed{e \Downarrow v}$

$$\frac{e_1 \Downarrow v}{e_1 \Box e_2 \Downarrow v} \qquad \frac{e_2 \Downarrow v}{e_1 \Box e_2 \Downarrow v}$$

(a) What property of $\mathsf{L}_{\mathsf{Arith}}$ (among those discussed in Lecture 2) is violated after we add $\Box$?

(b) Write a sensible rule for typechecking $e_1 \Box e_2$.

(c) For each of the following expressions e, list all of the possible values v such that $e \Downarrow v$ is derivable:

i. $(1 \Box 2) \times (3 \Box 4)$

ii. $\texttt{if}\ (\texttt{true} \Box \texttt{false})\ \texttt{then}\ 1\ \texttt{else}\ 2$

(d) Define an expression e and a value v such that there are two *different* derivations of the judgment $e \Downarrow v$. (What does it mean for the derivations to be different?)