

# Elements of Programming Languages

## Tutorial 1: Abstract syntax trees, evaluation and typechecking

### Week 3 (September 30–October 4, 2024)

Starred exercises (★) 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  $L_{\text{Arith}}$  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  $L_{\text{Arith}}$  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  $L_{\text{Arith}}$  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  $L_{\text{Arith}}$  expression and reverses the order of all multiplication operations (i.e.  $e_1 \times e_2$  becomes  $e_2 \times e_1$ ).
- (e) (★) 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:

$e \Downarrow v$

$$\begin{array}{c}
\frac{}{v \Downarrow v} \quad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{e_1 + e_2 \Downarrow v_1 +_{\mathbb{N}} v_2} \quad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{e_1 \times e_2 \Downarrow v_1 \times_{\mathbb{N}} v_2} \\
\frac{e_1 \Downarrow v \quad e_2 \Downarrow v}{e_1 == e_2 \Downarrow \mathbf{true}} \quad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad v_1 \neq v_2}{e_1 == e_2 \Downarrow \mathbf{false}} \\
\frac{e \Downarrow \mathbf{true} \quad e_1 \Downarrow v_1}{\mathbf{if } e \mathbf{ then } e_1 \mathbf{ else } e_2 \Downarrow v_1} \quad \frac{e \Downarrow \mathbf{false} \quad e_2 \Downarrow v_2}{\mathbf{if } e \mathbf{ then } e_1 \mathbf{ else } e_2 \Downarrow v_2}
\end{array}$$

Write out derivation trees for the following expressions:

- (a)  $6 \times 9$
- (b)  $3 \times 3 + 4 \times 4 == 5 \times 5$
- (c)  $(\star) \mathbf{if } 1 + 1 == 2 \mathbf{ then } 2 + 3 \mathbf{ else } 2 * 3$
- (d)  $(\star) (\mathbf{if } 1 + 1 == 2 \mathbf{ then } 3 \mathbf{ else } 4) + 5$

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

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

$$\begin{array}{c}
\frac{n \in \mathbb{N}}{\vdash n : \mathbf{int}} \quad \frac{\vdash e_1 : \mathbf{int} \quad \vdash e_2 : \mathbf{int}}{\vdash e_1 + e_2 : \mathbf{int}} \quad \frac{\vdash e_1 : \mathbf{int} \quad \vdash e_2 : \mathbf{int}}{\vdash e_1 \times e_2 : \mathbf{int}} \\
\frac{b \in \mathbb{B}}{\vdash b : \mathbf{bool}} \quad \frac{\vdash e_1 : \tau \quad \vdash e_2 : \tau}{\vdash e_1 == e_2 : \mathbf{bool}} \quad \frac{\vdash e : \mathbf{bool} \quad \vdash e_1 : \tau \quad \vdash e_2 : \tau}{\vdash \mathbf{if } e \mathbf{ then } e_1 \mathbf{ else } e_2 : \tau}
\end{array}$$

Write out typing derivations for the following judgments:

- (a)  $\vdash 6 \times 9 : \mathbf{int}$
- (b)  $(\star) \vdash (\mathbf{if } 1 + 1 == 2 \mathbf{ then } 3 \mathbf{ else } 4) + 5 : \mathbf{int}$

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

$$\begin{array}{l}
e ::= e_1 + e_2 \mid e_1 \times e_2 \mid n \in \mathbb{N} \\
\mid \mathbf{true} \mid \mathbf{false} \mid e_1 == e_2 \mid \mathbf{if } e \mathbf{ then } e_1 \mathbf{ else } e_2 \\
\mid e_1 \square e_2
\end{array}$$

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

$$\boxed{e \Downarrow v}$$

$$\frac{e_1 \Downarrow v}{e_1 \square e_2 \Downarrow v} \quad \frac{e_2 \Downarrow v}{e_1 \square e_2 \Downarrow v}$$

- (a) What property of  $L_{\text{Arith}}$  (among those discussed in Lecture 2) is violated after we add  $\square$ ?
- (b) Write a sensible rule for typechecking  $e_1 \square e_2$ .
- (c) For each of the following expressions  $e$ , list all of the possible values  $v$  such that  $e \Downarrow v$  is derivable:
  - i.  $(1 \square 2) \times (3 \square 4)$
  - ii.  $\mathbf{if } (\mathbf{true} \square \mathbf{false}) \mathbf{ then } 1 \mathbf{ else } 2$
- (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?)