

# Programming Languages and Types

## Exercise 10

Yi Dai

January 18, 2013

### 1 Abstract Syntax vs. Concrete Syntax

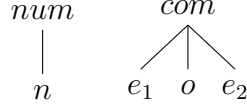
Presented below is the syntax of a language (call it AE) for (a subset of) arithmetic expressions.

	$e \in Exp$	
	$n \in Nat$	
	$o \in Opr$	
(expression)	$e ::= n$	(numeral)
	$e_1 o e_2$	(compound)
(operator)	$o ::= +$	(plus)
	$-$	(minus)
	$*$	(times)
	$/$	(divides)

In theoretical formulation, such a presentation is usually *said* to define the **abstract syntax** of a language for the following reasons: first, it captures all the structures of the language; second, the grammar is not concerned to handle ambiguity but leaves the deal to convention (like operator precedence, associativity and parentheses). However, the syntax thus defined is only *semi-abstract*. It is *not fully abstract* since it uses **infix notation**. It is *not completely concrete* either, since it does not handle the ambiguity of infix notation. A fully abstract syntax (with the declaration of syntactic categories and the definition of operators omitted just for conciseness) for AE could be:

(expression)	$e ::= num(n)$	(numeral)
	$com(o, e_1, e_2)$	(compound)

It describes the leaf and node structure that build up syntax trees



and can be readily implemented using tree-like data structures, for example, in Scala as:

```
sealed abstract class Exp
type Opr = String
type Nat = Int
case class Num(nat : Nat) extends Exp
case class Com(opr : Opr, lhs : Exp, rhs : Exp) extends Exp
```

It also gives us the freedom to choose the concrete syntax for the language, say **prefix** or **postfix notation** instead of infix notation. Usually infix notation is chosen for its readability. Thus semi-abstract syntax given above is widely spread.

## 2 Inductive Definitions and Rule Induction

### 2.1 Inductive Definitions

1. Inductively-defined sets: natural numbers, arithmetic expressions, etc.

$$\frac{n \in \textit{Nat}}{n \in \textit{Exp}} \text{NUM} \qquad \frac{e_1 \in \textit{Exp} \quad e_2 \in \textit{Exp} \quad o \in \textit{Opr}}{e_1 \, o \, e_2 \in \textit{Exp}} \text{COM}$$

2. Inductively-defined relations:  $m \, \textit{Div} \, n$ , evaluation relation of arithmetic expressions, etc.

$$\frac{}{m \, \textit{Div} \, 0} \text{DNIL} \qquad \frac{m \, \textit{Div} \, n}{m \, \textit{Div} \, n + m} \text{DSUM}$$

Note that a relation is just a set of tuples. Hence  $m \, \textit{Div} \, n$  is essentially an inductively-defined set of pairs  $(m, n)$ .

### 2.2 Rule Induction

Rule induction rules! For every *sound* inductive definition, we have a **rule induction principle** for free. The notion is simple, to prove some property  $P$  for every element of an inductively defined set, it is sufficient to prove  $P$

holds for the conclusion assuming  $P$  holds for all the premises, for every rule in the inductive definition. That is, for every rule of the form

$$\frac{\text{premise}_1 \quad \dots \quad \text{premise}_n}{\text{conclusion}},$$

prove  $P(\text{conclusion})$  assuming  $P(\text{premise}_1), \dots, P(\text{premise}_n)$ , for every  $n \in \mathbf{N}$ . When  $n = 0$ , a rule becomes an axiom. In this case, you have to prove  $P(\text{conclusion})$  “out of thin air”.

1. Prove that the sum of the first  $n$  natural numbers is  $\frac{n(n+1)}{2}$ , that is, prove

$$\sum_{i=0}^n i = \frac{n(n+1)}{2}$$

2. Prove that  $m \text{ Div } n_1$  and  $m \text{ Div } n_2$  implies  $m \text{ Div } (n_1 + n_2)$ .

*Proof.* We prove the property

$$P(m \text{ Div } n_1) = [\forall n_2 \in \mathbf{N}, m \text{ Div } n_2 \text{ implies } m \text{ Div } (n_1 + n_2)]$$

for every  $m \text{ Div } n_1$  by rule induction on  $m \text{ Div } n_1$ .

- Base case: for  $\frac{}{m \text{ Div } 0}$ , we want to prove

$$P(m \text{ Div } 0) = [\forall n_2 \in \mathbf{N}, m \text{ Div } n_2 \text{ implies } m \text{ Div } (0 + n_2)]$$

Since  $0 + n_2 = n_2$ , the goal is to prove  $m \text{ Div } n_2$  implies  $m \text{ Div } n_2$ , which is trivial.

- Inductive case: for  $\frac{m \text{ Div } n_1}{m \text{ Div } n_1 + m}$ , we want to prove

$$P(m \text{ Div } (n_1 + m)) = [\forall n_2 \in \mathbf{N}, m \text{ Div } n_2 \text{ implies } m \text{ Div } (n_1 + m + n_2)],$$

assuming the inductive hypothesis

$$P(m \text{ Div } n_1) = [\forall n_2 \in \mathbf{N}, m \text{ Div } n_2 \text{ implies } m \text{ Div } (n_1 + m)],$$

Suppose  $\forall n_2 \in \mathbf{N}, m \text{ Div } n_2$ , by the inductive hypothesis, we have  $m \text{ Div } (n_1 + n_2)$ , then apply the rule DSUM by instantiating  $n$  with  $(n_1 + n_2 + m)$ , we can conclude  $m \text{ Div } (m \text{ Div } n_1 + n_2 + m)$ , that is,

$$\frac{m \text{ Div } (n_1 + n_2)}{m \text{ Div } (n_1 + n_2 + m)} \text{ DSUM.}$$

Further, from  $m \text{ Div } (n_1 + n_2 + m)$ , by associativity and commutativity of  $+$ , we can derive  $m \text{ Div } (n_1 + m + n_2)$ , which is exactly what we want to prove.

Therefore, we have proved  $P(m \text{ Div } n_1)$  for every  $m \text{ Div } n_1$ , that is,  $\forall m, n_1, n_2 \in \mathbf{N}, m \text{ Div } n_1$  and  $m \text{ Div } n_2$  implies  $m \text{ Div } (n_1 + n_2)$ .  $\square$

### 3 Evaluation Semantics vs. Reduction Semantics

1. Give the (structural) evaluation semantics (aka. (structural) big-step (operational) semantics, natural semantics) for AE.

We first define a syntactic category for values. For convenience, we integrate it into the semi-abstract syntax given above for AE. Hereby we have the following syntax definition:

$$\begin{array}{ll}
 e \in Exp & \\
 n \in Int & \\
 o \in Opr & \\
 v \in Val & \\
 \\
 \begin{array}{llll}
 \text{(expression)} & e & ::= & v \quad \text{(value)} \\
 & | & & e_1 \ o \ e_2 \quad \text{(compound)}
 \end{array} \\
 \\
 \begin{array}{llll}
 \text{(operator)} & o & ::= & + \quad \text{(plus)} \\
 & | & & - \quad \text{(minus)} \\
 & | & & * \quad \text{(times)} \\
 & | & & / \quad \text{(divides)}
 \end{array} \\
 \\
 \begin{array}{llll}
 \text{(value)} & v & ::= & n \quad \text{(numeral)}
 \end{array}
 \end{array}$$

The evaluation semantics for arithmetic expressions is given by a evaluation relation between expressions and values (the final form of ex-

pressions after a series of reductions), notated as  $e \Longrightarrow v$ , inductively defined as follows:

$$\frac{}{v \Longrightarrow v} \text{ EvV} \quad \frac{e_1 \Longrightarrow v_1 \quad e_2 \Longrightarrow v_2}{e_1 \circ e_2 \Longrightarrow op(o, v_1, v_2)} \text{ EvC}$$

The inductive definition contains two rules, one axiom named EvV, one named EvC with two premises. The axiom EvV essentially says that a value evaluates to itself. In this example, a value can only be a number. The rule EvC says, to obtain the value of a compound expression  $e_1 \circ e_2$ , evaluate  $e_1$  to  $v_1$  and  $e_2$  to  $v_2$ , then use a primitive operation corresponding to the operator  $o$  to get the value of the whole expression from the two values  $v_1$  and  $v_2$ . Note that for simple arithmetic expressions, the order of the two premises does not matter since the evaluation of the two sub-expressions are independent of each other.

These rules should remind you of the interpreter you have crafted for arithmetic expressions.

Here is a demonstration of how to apply these evaluation rules to obtain the value of an example expression:  $1 + 2 * 3 - 4$ . Note that we assume the meta-function  $op$  can handle the operators  $+$ ,  $*$  and  $-$  correctly.

$$\frac{\frac{\frac{}{1 \Longrightarrow 1} \text{ EvV} \quad \frac{\frac{}{2 \Longrightarrow 2} \text{ EvV} \quad \frac{}{3 \Longrightarrow 3} \text{ EvV}}{2 * 3 \Longrightarrow op(*, 2, 3) = 6} \text{ EvC}}{1 + 2 * 3 \Longrightarrow op(+, 1, 6) = 7} \text{ EvC} \quad \frac{}{4 \Longrightarrow 4} \text{ EvV}}{1 + 2 * 3 - 4 \Longrightarrow op(-, 7, 4) = 3} \text{ EvC}$$

So we know the value of the expression  $1 + 2 * 3$  is 7.

2. Compare evaluation semantics and (structural) reduction semantics (aka. (structural) small-step (operational) semantics).

The reduction semantics for arithmetic expressions is given by a reduction relation between expressions, notated as  $e \longrightarrow e'$ , inductively defined as follows:

$$\frac{}{v_1 \circ v_2 \longrightarrow op(o, v_1, v_2)} \text{ RED}$$

$$\frac{e_1 \longrightarrow e'_1}{e_1 \circ e_2 \longrightarrow e'_1 \circ e_2} \text{ RDL} \quad \frac{e_2 \longrightarrow e'_2}{e_1 \circ e_2 \longrightarrow e_1 \circ e'_2} \text{ RDR}$$

The axiom RED says that we can invoke the primitive operation corresponding to the operator  $o$  only when both its operands have reduced

to values. The rule RDL covers one-step reduction of the left operand of a compound expression, while the rule RDR covers that of the right operand. Note that there is no longer a rule like  $v \longrightarrow v$ , since a value cannot be reduced in one step to anything. If such a rule is included in the definition, after an expression is reduced to a value, one can keep reducing it to itself by applying this rule till the end of the world (Who knows when it is, maybe Maya people?).

Here is a demonstration of how to apply these reduction rules to obtain the final result the same expression  $1 + 2 * 3 - 4$ . Again, we assume the correctness of the meta-function *op*.

$$\frac{\frac{2 * 3 \longrightarrow op(*, 2, 3) = 6}{1 + 2 * 3 \longrightarrow 1 + 6} \text{ RDR}}{1 + 2 * 3 - 4 \longrightarrow 1 + 6 - 4} \text{ RDL} \quad \text{RED}$$

This is just one-step reduction. To reach the final result of the expression, we need continue reducing the result expression  $1 + 6 - 4$ :

$$\frac{1 + 6 \longrightarrow op(+, 1, 6) = 7}{1 + 6 - 4 \longrightarrow 7 - 4} \text{ RDL} \quad \text{RED}$$

Go on reducing  $7 - 4$ :

$$\frac{}{7 - 4 \longrightarrow 3} \text{ RED}$$

Now that the number 3 can no longer be reduced, it is the result of the whole expression. So we have seen that the original expression  $1 + 2 * 3 - 4$  is reduced by *three* steps to the number 3, that is,

$$1 + 2 * 3 - 4 \longrightarrow 1 + 6 - 4 \longrightarrow 7 - 4 \longrightarrow 3$$

Note that the number of reduction steps is clearly indicated by the number of occurrences of the rule RED in the three derivation trees.