Programming Languages and Types

Exercise 10

Yi Dai

January 21, 2013

1 Abstract Syntax vs. Concrete Syntax

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

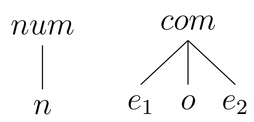
$$e \in Exp$$
$$n \in Nat$$
$$o \in Opr$$

$$\begin{array}{lcll}
\text{(expression)} \quad e & ::= & n & \text{(numeral)} \\
& | & e_1 \; o \; e_2 & \text{(compound)} \\
\\
\text{(operator)} \quad o & ::= & \texttt{+} & \text{(plus)} \\
& | & \texttt{-} & \text{(minus)} \\
& | & \texttt{*} & \text{(times)} \\
& | & \texttt{/} & \text{(divides)}
\end{array}$$

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:

$$\begin{array}{lcll}
\text{(expression)} \quad e & ::= & num\,(n) & \text{(numeral)} \\
& | & com\,(o, e_1, e_2) & \text{(compound)}
\end{array}$$

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.

$$\frac{n \in Nat}{n \in Exp}\ \textsc{Num} \qquad \frac{e_1 \in Exp \qquad e_2 \in Exp \qquad o \in Opr}{e_1\ o\ e_2 \in Exp}\ \textsc{Com}$$

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

$$\frac{}{m\ Div\ 0}\ \textsc{DNil} \qquad \frac{m\ Div\ n}{m\ Div\ n+m}\ \textsc{DSum}$$

Note that a relation is just a set of tuples. Hence $m\ Div\ n$ 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

$$\frac{premise_1 \qquad \ldots \qquad premise_n}{conclusion},$$

prove $P\,(conclusion)$ assuming $P\,(premise_1)$, ..., $P\,(premise_n)$, for every $n \in \mathbf{N}$. When $n = 0$, a rule becomes an axiom. In this case, you have to prove $P(conclusion)$ "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 $m\ Div\ n_1$ and $m\ Div\ n_2$ implies $m\ Div\ (n_1 + n_2)$.

   Proof. We prove the property

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

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

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

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

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

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

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

     assuming the inductive hypothesis

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

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

$$\frac{m\ Div\ (n_1 + n_2)}{m\ Div\ (n_1 + n_2 + m)}\ \textsc{DSum}.$$

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

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

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:

$$\begin{aligned} e &\in Exp \\ n &\in Int \\ o &\in Opr \\ v &\in Val \end{aligned}$$

$$\begin{array}{lcclr}
\text{(expression)} & e & ::= & v & \text{(value)} \\
 & & | & e_1\ o\ e_2 & \text{(compound)} \\
\\
\text{(operator)} & o & ::= & \texttt{+} & \text{(plus)} \\
 & & | & \texttt{-} & \text{(minus)} \\
 & & | & \texttt{*} & \text{(times)} \\
 & & | & \texttt{/} & \text{(divides)} \\
\\
\text{(value)} & v & ::= & n & \text{(numeral)}
\end{array}$$

The evaluation semantics for arithmetic expressions is given by a evaluation relation between expressions and values (the final form of expressions after a series of reductions), notated as $e \Longrightarrow v$, inductively defined as follows:

$$\frac{}{v \Longrightarrow v}\ \textsc{EvV} \qquad \frac{e_1 \Longrightarrow v_1 \qquad e_2 \Longrightarrow v_2}{e_1\ o\ e_2 \Longrightarrow \ulcorner op\,(o, v_1, v_2)\urcorner}\ \textsc{EvC}$$

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 numeral. The rule EvC says, to obtain the value of a compound expression $e_1\ o\ e_2$, 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.

$$\frac{\dfrac{\dfrac{}{\texttt{1} \Longrightarrow \texttt{1}}\ \textsc{EvV} \qquad \dfrac{\dfrac{}{\texttt{2} \Longrightarrow \texttt{2}}\ \textsc{EvV} \qquad \dfrac{}{\texttt{3} \Longrightarrow \texttt{3}}\ \textsc{EvV}}{\texttt{2 * 3} \Longrightarrow \ulcorner op\,(*, 2, 3)\urcorner = \texttt{6}}\ \textsc{EvC}}{\texttt{1 + 2 * 3} \Longrightarrow \ulcorner op\,(+, 1, 6)\urcorner = \texttt{7}}\ \textsc{EvC} \qquad \dfrac{}{\texttt{4} \Longrightarrow \texttt{4}}\ \textsc{EvV}}{\texttt{1 + 2 * 3 - 4} \Longrightarrow \ulcorner op\,(-, 7, 4)\urcorner = \texttt{3}}\ \textsc{EvC}$$

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:

$$\frac{}{v_1\ o\ v_2 \longrightarrow \ulcorner op\,(o, v_1, v_2)\urcorner}\ \textsc{Red}$$

$$\frac{e_1 \longrightarrow e_1'}{e_1\ o\ e_2 \longrightarrow e_1'\ o\ e_2}\ \textsc{RdL} \qquad \frac{e_2 \longrightarrow e_2'}{e_1\ o\ e_2 \longrightarrow e_1\ o\ e_2'}\ \textsc{RdR}$$

   The axiom Red says that we can invoke the primitve 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*.

$$\dfrac{\dfrac{\dfrac{}{\texttt{2 * 3} \longrightarrow \ulcorner op\,(*, 2, 3) \urcorner = \texttt{6}}\ \textsc{Red}}{\texttt{1 + 2 * 3} \longrightarrow \texttt{1 + 6}}\ \textsc{RdR}}{\texttt{1 + 2 * 3 - 4} \longrightarrow \texttt{1 + 6 - 4}}\ \textsc{RdL}$$

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

$$\dfrac{\dfrac{}{\texttt{1 + 6} \longrightarrow \ulcorner op\,(+, 1, 6) \urcorner = \texttt{7}}\ \textsc{Red}}{\texttt{1 + 6 - 4} \longrightarrow \texttt{7 - 4}}\ \textsc{RdL}$$

Go on reducing `7 - 4`:

$$\dfrac{}{\texttt{7 - 4} \longrightarrow \ulcorner op\,(-, 7, 4) \urcorner = \texttt{3}}\ \textsc{Red}$$

Now that the numeral `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 numeral `3`, that is,

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

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