Warm up: A simple language for arithmetic expressions

Massimo Merro

5 October 2020

Logic and software engineering

- Logic is the mathematical basis for software engineering
- We can make the following statement:

$$\text{logic} \;:\; \text{sw engineering} \quad = \quad \text{calculus} \;:\; \text{mechanical/civil engineering}$$

- Induction will be a foundational concept
- For instance, inductively defined sets and relations or inductive proofs are the basis of software verification

Anatomy of an inference system:

$$(\text{Axiom}) \; \frac{-}{\mathit{Conclusion}}$$

Conclusioni

Premesse

$$(\text{Rule}) \; \frac{\mathit{Hypothesis}_1 \quad \cdots \quad \mathit{Hypothesis}_n}{\mathit{Conclusion}} \; \mathit{condition}$$

conclusioni

A Language for Arithmetic Expression: Syntax

$$E \quad ::= \quad \mathtt{n} \quad | \quad E + E \quad | \quad E \times E \quad | \quad \cdots$$

where

- $\mathtt{n}$ ranges over the domain of numerals *Num*: $\mathtt{0}$, $\mathtt{1}$, $\cdots$
- E ranges over the domain of arithmetic expressions *Exp*
- numerals $\mathtt{0}$, $\mathtt{1}$, $\cdots$ are part of the syntax of our language
- they are piece of our syntax and they should not be confused with numbers $(0, 1, 2, \ldots \in \mathbb{N})$ which are mathematical objects
- instead of $\mathtt{0}$, $\mathtt{1}$, $\cdots$ we could have used in our language the terminals $\mathtt{zero}$, $\mathtt{uno}$, $\cdots$; it would have been exactly the same.
- $+$, $\times$, $\cdots$ are symbols for operations.

We will always work with abstract syntax. We will assume that we already did the parsing of our programs. So, the grammars we will use to define our languages define syntactic trees: parentheses are only used for disambiguation - they are not part of the grammar.

Operational semantics for the language Exp

An operational semantics for Exp has the goal to *evaluate* an arithmetic expression of the language to get its associated numeral.
This can be done in two different manners:

- via a small-step (or *structural*) semantics that provides a method to evaluate an expression, step by step
- via a big-step (or *natural*) semantics that ignores the intermediate steps and directly provides the final result

In the following, we assume that there is an obvious correspondence between the numeral $\mathtt{n}$ and the number n. This is just to make things simple: In another language the numeral 3 might be associated to the number 42!

Big-Step semantics for Exp

evaluation semantic,
big step semantic

Judgements:

Final state

$\mathtt{E} \Downarrow \mathtt{n}$

Meaning:

The evaluation of expression E results in the numeral $\mathtt{n}$.

Big-Step Semantics for Exp

Axioms and Rules for Exp

$$(\text{B-Num})\ \frac{-}{\mathtt{n} \Downarrow \mathtt{n}} \qquad (\text{B-Add})\ \frac{\mathsf{E}_1 \Downarrow \mathtt{n}_1 \qquad \mathsf{E}_2 \Downarrow \mathtt{n}_2}{\mathsf{E}_1 + E_2 \Downarrow \mathtt{n}_3}\ n_3 = \mathsf{add}(n_1, n_2)$$

Similar rules for $\times$, $-$, $\cdots$

IMPORTANT: $\mathsf{add}(-,-)$ is a semantic operator on *numbers* NOT *numerals*.

How to read axioms

The axioms

$$(\text{B-Num})\ \frac{-}{\mathtt{n} \Downarrow \mathtt{n}}$$

says that:

The evaluation of numeral $\mathtt{n}$ *is* $\mathtt{n}$ *itself.*

In the axiom (B-Num) the symbol $\mathtt{n}$ is a kind of *variable* which can be replaced with any numeral $0,\ 1,\ 2,\ \cdots$. Those kinds of variables are called *meta-variables*.

How to read rules

The rule

$$(\text{B-Add}) \quad \frac{\mathsf{E}_1 \Downarrow \mathtt{n}_1 \qquad \mathsf{E}_2 \Downarrow \mathtt{n}_2}{\mathsf{E}_1 + E_2 \Downarrow \mathtt{n}_3} \quad n_3 = \text{add}(n_1, n_2)$$

should be read in the following manner:

- given two expressions E_1 and E_2
- if it is the case that $\mathsf{E}_1 \Downarrow \mathtt{n}_1$
- and it is the case that $\mathsf{E}_2 \Downarrow \mathtt{n}_2$
- then it follows that $\mathsf{E}_1 + E_2 \Downarrow \mathtt{n}_3$
- where $\mathtt{n}_3$ is the numeral associated to the number n_3, such that $n_3 = \text{add}(n_1, n_2)$
- recall that $\text{add}(-,-)$ is an operation on *numbers* NOT *numerals*.

In the rule (B-Add), E_1, E_2, $\mathtt{n}_1$, $\mathtt{n}_2$, $\mathtt{n}_3$ are meta-variables.

How to use axioms and rules

We can apply axioms and rules to *derive* judgements. Such derivations take the form of trees:

$$\text{(B-Add)}\ \frac{\text{(B-Num)}\ \dfrac{-}{3 \Downarrow 3} \qquad \text{(B-Add)}\ \dfrac{\text{(B-Num)}\ \dfrac{-}{2 \Downarrow 2} \quad \text{(B-Num)}\ \dfrac{-}{1 \Downarrow 1}}{(2+1) \Downarrow 3}}{3 + (2+1) \Downarrow 6}$$

For example, the derivation above allows us to derive the judgement:

$$3 + (2+1) \Downarrow 6$$

by applying three times the axiom (B-Num) and two times rule (B-Add).

Small-step Semantics for Exp

Continuazione del programma

Judgements:

$$E_1 \rightarrow E_2$$

Meaning:

After performing one-step of E_1 the expression E_2 remain to be evaluated.

A Left-to-right Small-step Semantics for Exp

Inference rules

$$(\text{S-Left})\ \frac{\mathsf{E}_1 \twoheadrightarrow \mathsf{E}_1'}{\mathsf{E}_1 + \mathsf{E}_2 \twoheadrightarrow \mathsf{E}_1' + \mathsf{E}_2}$$

$$(\text{S-N.Right})\ \frac{\mathsf{E}_2 \twoheadrightarrow \mathsf{E}_2'}{\mathtt{n}_1 + \mathsf{E}_2 \twoheadrightarrow \mathtt{n}_1 + \mathsf{E}_2'}$$

$$(\text{S-Add})\ \frac{-}{\mathtt{n}_1 + \mathtt{n}_2 \twoheadrightarrow \mathtt{n}_3}\ n_3 = \mathsf{add}(n_1, n_2)$$

We fix the evaluation order, from left to right. Something similar is not possible in a big-step semantics where expressions are evaluated in a single "big" step.

A Choice Small-step Semantics for Exp

A different small-step semantics for Exp is the following:

Inference rules:

$$\text{(S-Left)}\ \frac{\mathsf{E}_1 \rightharpoonup_{\mathsf{ch}} \mathsf{E}_1'}{\mathsf{E}_1 + \mathsf{E}_2 \rightharpoonup_{\mathsf{ch}} \mathsf{E}_1' + \mathsf{E}_2}$$

$$\text{(S-Right)}\ \frac{\mathsf{E}_2 \rightharpoonup_{\mathsf{ch}} \mathsf{E}_2'}{\mathsf{E}_1 + \mathsf{E}_2 \rightharpoonup_{\mathsf{ch}} \mathsf{E}_1 + \mathsf{E}_2'}$$

$$\text{(S-Add)}\ \frac{-}{\mathtt{n}_1 + \mathtt{n}_2 \rightharpoonup_{\mathsf{ch}} \mathtt{n}_3}\ n_3 = \mathsf{add}(n_1, n_2)$$

Here, no precedence is established during the evaluation.
Similar rules apply to the other operators $\times$, $-$, $\cdots$.

Executing small-step semantics

The relation $\rightarrowtriangle^k$, for $k \in \mathbb{N}$ (k may be 0)

We write $E \rightarrowtriangle^k E_k$ whenever:

$$E = E_0 \rightarrowtriangle E_1 \rightarrowtriangle E_2 \rightarrowtriangle \ldots \rightarrowtriangle E_k$$

The relation $\rightarrowtriangle^*$

Stella di Cliny, faccio n passi con n >= 0

We write E $\rightarrowtriangle^*$ F if $E \rightarrowtriangle^k F$ for some $k \in \mathbb{N}$.

The relation $\rightarrowtriangle^*$ is called *reflexive and transitive closure* of $\rightarrowtriangle$. Reflexive because k may be 0.

The final answer

We say that $\mathtt{n}$ is the final answer of E if E $\rightarrowtriangle^*$ $\mathtt{n}$.

Questions

Internal consistency of semantics

- Is it possible to derive

$$\mathsf{E} \Downarrow 3 \quad \text{and} \quad \mathsf{E} \Downarrow 7$$

  for some expression E?
- Is there some expression E which has no resulting value:

$$\mathsf{E} \twoheadrightarrow^{*} \mathtt{n} \text{ for no numeral } \mathtt{n}$$

Questions

Consistency between the different semantics

What is the relationship between the different judgements:

- $\mathsf{E} \Downarrow \mathtt{n}$
- $\mathsf{E} \rightarrow^{*} \mathtt{n}$
- $\mathsf{E} \rightarrow^{*}_{\mathsf{ch}} \mathtt{n}$

Usefulness

Can these techniques be applied to realistic programming languages?