Some notes on presenting *formal languages.*

Jan de Muijnck-Hughes

29th January 2019

This document presents a rough guide on formal programming language descriptions. We do this through consideration of a simple language for integer and boolean arithmetic. The original reduced version of this language was presented by Jeremy Siek in his blog post:

`http://siek.blogspot.com/2012/07/crash-course-on-notation-in-programming.html`

Jeremy Siek has since released more material from a two part seminar series originally presented at λ Conf. 2018.

- Part 1: `https://youtu.be/vU3caZPtT2I`
- Part 2: `https://youtu.be/MhuK_aepu1Y`
- Accompanying notes: `https://t.co/nvWmlkPewl`

Reading Guide

- Sections 1 to 5: If you are interested in describing an executable policy language.
- Sections 1 to 3 and 5: If you are interested in describing a modelling language that goes from one modelling language to another.

Contents

1 Abstract Syntax, Types, & Contexts **2**

2 Typing Rules **2**

3 Substitution **2**

4 Big Step Semantics **3**

5 Interpretation **3**

6 Note **4**

1 Abstract Syntax, Types, & Contexts

$$
\begin{array}{rll}
n & ::= \text{Integers} & \text{Constants} \\
b & ::= \mathsf{False} \mid \mathsf{True} & \\
e & ::= \mu \mid n \mid b & \text{Variables/Constants} \\
 & \mid \mathsf{add}\,(e,e) \mid \mathsf{sub}\,(e,e) \mid \mathsf{div}\,(e,e) \mid \mathsf{mul}\,(e,e) \mid \mathsf{neg}\,(e) & \text{Expressions} \\
 & \mid \mathsf{greaterThan}\,(e,e) \mid \mathsf{lessThan}\,(e,e) \mid \mathsf{equal}\,(e,e) & \\
 & \mid \mathsf{or}\,(e,e) \mid \mathsf{and}\,(e,e) \mid \mathsf{xor}\,(e,e) \mid \mathsf{not}\,(e) & \\
 & \mid \underline{\mathsf{let}}\ \mu\ \underline{\mathsf{be}}\ e\ \underline{\mathsf{in}}\ e \mid \underline{\mathsf{if}}\ e\ \underline{\mathsf{then}}\ e\ \underline{\mathsf{else}}\ e & \text{Statements} \\
T : \mathcal{T} & ::= \mathbb{Z} \mid \mathsf{Bool} & \text{Types} \\
\Gamma & ::= \Gamma, (x : T) \mid \emptyset & \text{Context}
\end{array}
$$

Figure 1: Our language's abstract systanx, types, and context.

Figure 1 presents the syntactical structure, and types for our language. Our language contains Integers and Boolean values, we leave abstract how integers are written. Variables (μ) are constructed using Let-bindings, and conditional statements take the well-known form of `if-then-else`.

Core expressions allow for addition, substraction, division, multiplication, and negation of integers. Integers can also be compared using standard comparison operations of: greater-than, less-than, and equals-to. Boolean operators provide logical negation, conjunction, disjunction, and, exclusive disjunction.

A context (Γ) keeps track of well-typed expressions, and our context can be expanded.

2 Typing Rules

Figure 2 present's our language's typing rules. These rules dictate what it means for an expression/statement to be well-formed. We do this by assigning types to expressions/statements. We read typing rules as follows: Things above the lines are premises such that if all premises are true then the judgement (below the line) will also be true. When given any expression/statement in our language we can use the typing rules to construct a derivation that provides proof that the expression/statement is well-typed, that is we can apply each rule and form a derivation tree. If we cannot construct this tree then the expression is ill-typed and syntactically not valid. Typing rules are a compile time static check.

We can only proceed to computation/evaluation of our language iff it is well-typed.

3 Substitution

Figure 3 presents a standard set of substitution rules for our language. These rules describe how we can interate over our expressions/statements and swap variables for values. Note how they form a recursive call. We will use these rules to help us describe how we can transform our program instances.

$$\textsc{Intro-Nat}\ \frac{}{n : \mathbb{Z}} \qquad \textsc{Intro-F}\ \frac{}{\mathsf{False} : \mathsf{Bool}} \qquad \textsc{Intro-T}\ \frac{}{\mathsf{True} : \mathsf{Bool}} \qquad \textsc{VAR}\ \frac{\mu : T \in \Gamma}{\mu : T}$$

$$\textsc{ADD}\ \frac{\Gamma \vdash a : \mathbb{Z} \qquad \Gamma \vdash b : \mathbb{Z}}{\Gamma \vdash \mathsf{add}\,(a, b) : \mathbb{Z}} \qquad \textsc{Sub}\ \frac{\Gamma \vdash a : \mathbb{Z} \qquad \Gamma \vdash b : \mathbb{Z}}{\Gamma \vdash \mathsf{sub}\,(a, b) : \mathbb{Z}} \qquad \textsc{DIV}\ \frac{\Gamma \vdash a : \mathbb{Z} \qquad \Gamma \vdash b : \mathbb{Z}}{\Gamma \vdash \mathsf{div}\,(a, b) : \mathbb{Z}}$$

$$\textsc{MUL}\ \frac{\Gamma \vdash a : \mathbb{Z} \qquad \Gamma \vdash b : \mathbb{Z}}{\Gamma \vdash \mathsf{mul}\,(a, b) : \mathbb{Z}} \qquad \textsc{NEG}\ \frac{\Gamma \vdash n : \mathbb{Z}}{\Gamma \vdash \mathsf{neg}\,(n) : \mathbb{Z}} \qquad \textsc{OR}\ \frac{\Gamma \vdash a : \mathsf{Bool} \qquad \Gamma \vdash b : \mathsf{Bool}}{\Gamma \vdash \mathsf{or}\,(a, b) : \mathsf{Bool}}$$

$$\textsc{AND}\ \frac{\Gamma \vdash a : \mathsf{Bool} \qquad \Gamma \vdash b : \mathsf{Bool}}{\Gamma \vdash \mathsf{and}\,(a, b) : \mathsf{Bool}} \qquad \textsc{Xor}\ \frac{\Gamma \vdash a : \mathsf{Bool} \qquad \Gamma \vdash b : \mathsf{Bool}}{\Gamma \vdash \mathsf{xor}\,(a, b) : \mathsf{Bool}} \qquad \textsc{NOT}\ \frac{\Gamma \vdash n : \mathsf{Bool}}{\Gamma \vdash \mathsf{not}\,(n) : \mathsf{Bool}}$$

$$\textsc{GT}\ \frac{\Gamma \vdash a : \mathbb{Z} \qquad \Gamma \vdash b : \mathbb{Z}}{\Gamma \vdash \mathsf{greaterThan}\,(a, b) : \mathsf{Bool}} \qquad \textsc{LT}\ \frac{\Gamma \vdash a : \mathbb{Z} \qquad \Gamma \vdash b : \mathbb{Z}}{\Gamma \vdash \mathsf{lessThan}\,(a, b) : \mathsf{Bool}} \qquad \textsc{EQ}\ \frac{\Gamma \vdash a : \mathbb{Z} \qquad \Gamma \vdash b : \mathbb{Z}}{\Gamma \vdash \mathsf{equal}\,(a, b) : \mathsf{Bool}}$$

$$\textsc{Let}\ \frac{\mu : T_1 \qquad \Gamma \vdash e_1 : T_1 \qquad \Gamma, (\mu : T_1) \vdash e_2 : T_2}{\Gamma \vdash \underline{\mathsf{let}}\ \mu\ \underline{\mathsf{be}}\ e_1\ \underline{\mathsf{in}}\ e_2 : T_2} \qquad \textsc{IF}\ \frac{\Gamma \vdash e : \mathsf{Bool} \qquad \Gamma \vdash l : T \qquad \Gamma \vdash r : T}{\Gamma \vdash \underline{\mathsf{if}}\ e\ \underline{\mathsf{then}}\ l\ \underline{\mathsf{else}}\ r : T}$$

Figure 2: Typing Rules

4 Big Step Semantics

Operational semantics describe how we evaluate our programs. This describes how we can *reduce*/evaluate our language expressions and statements to a single value. There are generally two common styles of operational semantics: Big-Step, and Small-Step. There are more formal names given but we generally refer to the styles using these names.

Big-Step semantics are concerned with what the final result is; we can skip description of intermediate computations. Small-Step semantics are concerned with how we get to the final result; we cannot skip intermediate computations. Both have pros and cons.

Figure 4 presents Big-Step style semantics, here we use *real* operations to show how an expression is reduced using *real* integer and boolean operators. Interestingly are the rules for IF-True and IF-False, they describe the branching that occurs with use of conditional statements.

5 Interpretation

In this final section we describe how we can interpret our language to another form, in this case concrete Java expressions. Like substitution, interpretation recursively operates over each language statement and expression. At each step if replaces the expression from our language with it's equivalent Java form. If we interpret a language we do not need to provide operational semantics, the target language provides this.

$$
\begin{aligned}
(\mu)[e/x] &::= \begin{cases} e & x \equiv \mu \\ x & x \not\equiv \mu \end{cases} \\
(\mathsf{add}\,(a,b))[e/x] &::= \mathsf{add}\,((a)[e/x],(b)[e/x]) \\
(\mathsf{sub}\,(a,b))[e/x] &::= \mathsf{sub}\,((a)[e/x],(b)[e/x]) \\
(\mathsf{div}\,(a,b))[e/x] &::= \mathsf{div}\,((a)[e/x],(b)[e/x]) \\
(\mathsf{mul}\,(a,b))[e/x] &::= \mathsf{mul}\,((a)[e/x],(b)[e/x]) \\
(\mathsf{neg}\,(a))[e/x] &::= \mathsf{neg}\,((a)[e/x]) \\
(\mathsf{or}\,(a,b))[e/x] &::= \mathsf{or}\,((a)[e/x],(b)[e/x]) \\
(\mathsf{and}\,(a,b))[e/x] &::= \mathsf{and}\,((a)[e/x],(b)[e/x]) \\
(\mathsf{xor}\,(a,b))[e/x] &::= \mathsf{xor}\,((a)[e/x],(b)[e/x]) \\
(\mathsf{not}\,(a))[e/x] &::= \mathsf{not}\,((a)[e/x]) \\
(\mathsf{greaterThan}\,(a,b))[e/x] &::= \mathsf{greaterThan}\,((a)[e/x],(b)[e/x]) \\
(\mathsf{lessThan}\,(a,b))[e/x] &::= \mathsf{lessThan}\,((a)[e/x],(b)[e/x]) \\
(\mathsf{equal}\,(a,b))[e/x] &::= \mathsf{equal}\,((a)[e/x],(b)[e/x]) \\
(\underline{\mathsf{if}}\ e_1\ \underline{\mathsf{then}}\ l\ \underline{\mathsf{else}}\ r)[e/x] &::= \underline{\mathsf{if}}\ (e_1)[e/x]\ \underline{\mathsf{then}}\ (l)[e/x]\ \underline{\mathsf{else}}\ (r)[e/x] \\
(\underline{\mathsf{let}}\ \mu\ \underline{\mathsf{be}}\ e_1\ \underline{\mathsf{in}}\ e_2)[e/x] &::= \underline{\mathsf{let}}\ \mu\ \underline{\mathsf{be}}\ (e_1)[e/x]\ \underline{\mathsf{in}}\ (e_2)[e/x]
\end{aligned}
$$

Figure 3: Substitution Rules

6 Note

We can use substitution, operational semantics, and interpretation forms more than one. It is common to provide a rich, somewhat complex, abstract syntax for a language that has simpler (reduced) forms. We can use interpretation, big-step semantics, and substitution to describe how we go from one form to another.

$$\textsc{Nat}\ \frac{}{n \Downarrow n} \qquad \textsc{Bool}\ \frac{}{b \Downarrow b} \qquad \text{ADD}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{add}\,(a,b) \Downarrow a' + b'} \qquad \textsc{Sub}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{sub}\,(a,b) \Downarrow a' - b'}$$

$$\text{DIV}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{div}\,(a,b) \Downarrow \frac{a'}{b'}} \qquad \text{MUL}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{mul}\,(a,b) \Downarrow a' \times b'} \qquad \text{NEG}\ \frac{n \Downarrow n'}{\mathsf{neg}\,(n) \Downarrow (-1) \times n'}$$

$$\text{OR}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{or}\,(a,b) \Downarrow a' \vee b'} \qquad \text{AND}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{and}\,(a,b) \Downarrow a' \wedge b'} \qquad \textsc{Xor}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{xor}\,(a,b) \Downarrow a' \oplus b'}$$

$$\text{NOT}\ \frac{n \Downarrow n'}{\mathsf{not}\,(n) \Downarrow \neg n} \qquad \text{GT}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{greaterThan}\,(a,b) \Downarrow a' > b'} \qquad \text{LT}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{lessThan}\,(a,b) \Downarrow a' < b'}$$

$$\text{EQ}\ \frac{a \Downarrow a' \qquad b \Downarrow b'}{\mathsf{equal}\,(a,b) \Downarrow a' \equiv b'} \qquad \textsc{Let}\ \frac{e_1 \Downarrow e_1' \qquad (e_2)[\mu/e_1'] \Downarrow e_2'}{\underline{\mathsf{let}}\ \mu\ \underline{\mathsf{be}}\ e_1\ \underline{\mathsf{in}}\ e_2 \Downarrow e_2'} \qquad \text{IF-}\textsc{True}\ \frac{e \Downarrow \mathsf{True} \qquad l \Downarrow l'}{\underline{\mathsf{if}}\ e\ \underline{\mathsf{then}}\ l\ \underline{\mathsf{else}}\ r \Downarrow l'}$$

$$\text{IF-}\textsc{False}\ \frac{e \Downarrow \mathsf{False} \qquad r \Downarrow r'}{\underline{\mathsf{if}}\ e\ \underline{\mathsf{then}}\ l\ \underline{\mathsf{else}}\ r \Downarrow r'}$$

Figure 4: Big Step Semantics

$$
\begin{array}{rl}
\textsc{ArithLang} \rightarrow \textsc{Java} & \\
[\![\ \mathbb{Z}\ ]\!] & ::= \texttt{Integer} \\
[\![\ \mathsf{Bool}\ ]\!] & ::= \texttt{Boolean} \\
[\![\ n\ ]\!] & ::= \texttt{new Integer(}n\texttt{);} \\
[\![\ \mathsf{False}\ ]\!] & ::= \texttt{False} \\
[\![\ \mathsf{True}\ ]\!] & ::= \texttt{True} \\
[\![\ \mathsf{add}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{+} [\![\ b\ ]\!] \\
[\![\ \mathsf{sub}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{-} [\![\ b\ ]\!] \\
[\![\ \mathsf{div}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{/} [\![\ b\ ]\!] \\
[\![\ \mathsf{mul}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{*} [\![\ b\ ]\!] \\
[\![\ \mathsf{neg}\,(a)\ ]\!] & ::= \texttt{-1*} [\![\ a\ ]\!] \\
[\![\ \mathsf{or}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{||} [\![\ b\ ]\!] \\
[\![\ \mathsf{and}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{\&\&} [\![\ b\ ]\!] \\
[\![\ \mathsf{xor}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{\^{}} [\![\ b\ ]\!] \\
[\![\ \mathsf{not}\,(a)\ ]\!] & ::= \texttt{!} [\![\ a\ ]\!] \\
[\![\ \mathsf{greaterThan}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{>} [\![\ b\ ]\!] \\
[\![\ \mathsf{lessThan}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{<} [\![\ b\ ]\!] \\
[\![\ \mathsf{equal}\,(a,b)\ ]\!] & ::= [\![\ a\ ]\!] \texttt{==} [\![\ b\ ]\!] \\
[\![\ \underline{\mathsf{if}}\ e_1\ \underline{\mathsf{then}}\ l\ \underline{\mathsf{else}}\ r\ ]\!] & ::= [\![\ e_1\ ]\!] \texttt{ ? } [\![\ l\ ]\!] \texttt{ : } [\![\ r\ ]\!] \\
[\![\ \underline{\mathsf{let}}\ \mu\ \underline{\mathsf{be}}\ e_1\ \underline{\mathsf{in}}\ e_2\ ]\!] & ::= [\![\ (e_2)[\mu/e_1]\ ]\!]
\end{array}
$$

Figure 5: Interpretation Rules