Specification of the Nouga DSL

Simon Cockx

November 15, 2021

The goal of this work is two-fold. On the one hand it aims to eliminate flaws from the Rosetta language by formalizing its grammar and typing system. On the other hand it seeks to give solid ground for developers of code generators, giving a single source of truth about the intended semantics of generated code. Note that these goals also constitute two different audiences; one the developers of Rosetta, the other parties interested in translating Rosetta to a new language.

Throughout this document, T and S represent basic types and C represents a cardinality.

1 Syntax

Metavariables: D and E range over entity names, F ranges over function names, a and b range over attribute and parameter names, i and j range over signed integers, k and l range over positive integers, r ranges over signed decimals. Whitespace is ignored.

⟨DD⟩ ::=		*entity declarations:*
	`type` D (`extends` E)? `:` ⟨AT⟩*	
⟨FD⟩ ::=		*function declarations:*
	`func` F `:` `inputs :` ⟨AT⟩* `output :` ⟨AT⟩ `assign-output :` ⟨E⟩	
⟨AT⟩ ::=		*attribute declarations:*
	a ⟨T⟩ ⟨CD⟩	
⟨CD⟩ ::=		*cardinalities:*
	| `(` l `..` k `)`	*bounded*
	| `(` l `.. * )`	*unbounded*
⟨E⟩ ::=		*expressions:*
	| ⟨E⟩ `or` ⟨E⟩	
	| ⟨E⟩ `and` ⟨E⟩	
	| `not` ⟨E⟩	
	| ⟨E⟩ (`single` | `multiple`)? `exists`	
	| ⟨E⟩ `is absent`	
	| ⟨E⟩ `contains` ⟨E⟩	
	| ⟨E⟩ `disjoint` ⟨E⟩	
	| ⟨E⟩ (`all` | `any`)? (`=` | `<>`) ⟨E⟩	
	| ⟨E⟩ (`+` | `-`) ⟨E⟩	
	| ⟨E⟩ (`*` | `/`) ⟨E⟩	
	| ⟨E⟩ `count`	
	| ⟨E⟩ `->` a	
	| `if` ⟨E⟩ `then` ⟨E⟩ (`else` ⟨E⟩)?	
	| F `(` (⟨E⟩ (`,` ⟨E⟩)*)? `)`	
	| a	
	| ⟨LIT⟩	
	| `(` ⟨E⟩ `)`	
	| ⟨E⟩ `->` a `only exists`	
	| ⟨E⟩ `only-element`	
	| D `{` (a `=` ⟨E⟩ (`,` b `=` ⟨E⟩)*)? `}`	
⟨LIT⟩ ::=		*literals:*
	| `True` | `False`	*booleans*
	| i	*signed integers*
	| r	*signed decimals*
	| `empty`	*empty literal*
	| `[` (⟨E⟩ (`,` ⟨E⟩)*)? `]`	*list literals*
⟨T⟩ ::=		*basic types:*
	| D | `boolean` | `int` | `number` | `nothing`	

Operator precedence (note: this differs from Rosetta. The precedence of operators common with the C language are based on https://en.cppreference.com/w/c/language/operator_precedence.)

1. `->` (projection), `->` a `only exists`
2. `only-element`
3. `exists`, `is absent`, `count`
4. `not`
5. `*` (multiplication), `/` (division)
6. `+` (addition), `-` (subtraction)
7. `=` (equality), `<>` (inequality)
8. `contains`, `disjoint`
9. `and`
10. `or`

Syntactic sugar

$$\begin{aligned} \texttt{if}\, e_1\, \texttt{then}\, e_2 &\equiv \texttt{if}\, e_1\, \texttt{then}\, e_2\, \texttt{else}\, \texttt{empty} \\ \texttt{empty} &\equiv [] \\ e\, \texttt{is}\, \texttt{absent} &\equiv \texttt{not}\, (e\, \texttt{exists}) \end{aligned}$$

Note: Nouga has a couple of differences compared to Rosetta.

1. Nouga replaces the multiple `assign-output` statements with a single statement that fully defines the output of a function. Instead of defining one attribute of the output per `assign-output` statement, you can use a record-like syntax to explicitely create an instance. (see the last option of expressions ⟨E⟩)
2. Empty list literals are allowed.
3. In Nouga you can write `not` expressions.
4. The `only-element` keyword can be written behind any expression.
5. The `only exists` is restricted to expressions that end with a projection `->` a. This simplifies the runtime model (i.e. code generators) as attributes in Nouga do not need to keep track of their parent.

2 Auxiliary definitions

Data table $\mathrm{DT}(D)$ is a mapping from data type names to data declarations. Function table $\mathrm{FT}(F)$ is a mapping from function names to function declarations.

Attribute lookup

$$\frac{\mathrm{DT}(D) = \texttt{type}\, D : a_1\ T_1\ C_1 \ldots a_n\ T_n\ C_n}{\mathrm{attrs}(D) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n}$$

$$\frac{\mathrm{DT}(D) = \texttt{type}\, D\, \texttt{extends}\, C : a_1\ T_1\ C_1 \ldots a_n\ T_n\ C_n}{\mathrm{attrs}(D) = \mathrm{attrs}(C), a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n}$$

Supertypes

$$\frac{\mathrm{DT}(D) = \texttt{type}\, D\, \texttt{extends}\, E : \ldots}{E \in \mathrm{supertypes}(D)}$$

$$\frac{A \in \mathrm{supertypes}(D) \qquad \mathrm{DT}(A) = \texttt{type}\, A\, \texttt{extends}\, B : \ldots}{B \in \mathrm{supertypes}(D)}$$

Function lookups

$$\frac{\mathrm{FT}(F) = \texttt{func}\, F : \texttt{inputs} : a_1\ T_1\ C_1 \ldots a_n\ T_n\ C_n\, \texttt{output} : \ldots}{\mathrm{inputs}(F) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n}$$

$$\frac{\mathrm{FT}(F) = \texttt{func}\, F : \texttt{inputs} : \ldots \texttt{output} : a\ T\ C \ldots}{\mathrm{output}(F) = a\ T\ C}$$

$$\frac{\mathrm{FT}(F) = \texttt{func}\, F : \ldots \texttt{assign-output} : e}{\mathrm{op}(F) = e}$$

3 Semantics

Semantic domain: $\mathbb{D}$.

Single value: $\mathbb{D}_1$

3.1 Semantics of Types

Semantics of basic types $[\![T]\!]$.

$$[\![\texttt{boolean}]\!] = \mathbb{B} = \{\, true, false \,\}$$

$$[\![\texttt{int}]\!] = \mathbb{Z}$$

$$[\![\texttt{number}]\!] = \mathbb{R}$$

$$[\![D]\!] = \{a_k = [\![T_k\ C_k]\!] \mid k \in 1..n\}$$

where $\mathrm{attrs}(D) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n$

Semantics of types $[\![T\ C]\!]$.

$$[\![T\ \texttt{(}i\,\texttt{..}\,j\texttt{)}]\!] = [\![T]\!]^{[\![i]\!] \cdot [\![j]\!]}$$

Semantics of cardinality limits $[\![c]\!] \in \mathbb{N} \cup \{\, \infty \,\}$.

$$[\![i]\!] = i$$

$$[\![\texttt{*}]\!] = \infty$$

3.2 Semantical Algebra

Semantic algebra.

$$A^0 = Unit = \{\, () \,\}$$

$$A^n = A \times A^{n-1}$$

$$A^{m:n} = \bigcup_{k \in m..n} A^k$$

$$A^* = A^{0:\infty}$$

Note: from the above definition, A^1 formally equals $A \times Unit$, so elements of this set are of the form $(a, ())$ where $a \in A$. I might sometimes write a instead of $(a, ())$ if it is clear from the context what is meant. (similar for A^n, where I will leave out the last element of the cartesian product)

$$\_\, or \,\_ : \mathbb{B}^1 \times \mathbb{B}^1 \to \mathbb{B}^1 : (a, ())\, or\, (b, ()) = (a \vee b, ())$$

$$\_\, and \,\_ : \mathbb{B}^1 \times \mathbb{B}^1 \to \mathbb{B}^1 : (a, ())\, and\, (b, ()) = (a \wedge b, ())$$

$$not\,(\_) : \mathbb{B}^1 \to \mathbb{B}^1 : not\,((a, ())) = (\neg a, ())$$

$$(\_) \to \_\, [] \,\_ : \mathbb{B}^1 \times \mathbb{D} \times \mathbb{D} \to \mathbb{D} : ((a, ())) \to b\, []\, c = \begin{cases} b, & a = true \\ c, & a = false \end{cases}$$

$$count\,(\_) : \mathbb{D} \to \mathbb{Z}^1 : count\,((a_1, \ldots, a_n, ())) = (n, ())$$

$$flatten_{A,n}\,(\_) : (A^*)^n \to A^* :$$

$$flatten_{A,n}\,((a_{11}, \ldots, a_{1m_1}, ()), \ldots, (a_{n1}, \ldots, a_{nm_n}, ()))$$

$$= (a_{11}, \ldots, a_{1m_1}, \ldots, a_{nm_n}, ())$$

$$contains\,(\_, \_) : \mathbb{D} \times \mathbb{D} \to \mathbb{B}^1 : contains\,((a_1, \ldots, a_n, ()), (b_1, \ldots, b_m, ()))$$

$$= \begin{cases} (true, ()), & \forall i \in 1..n : \exists j \in 1..m : b_i = a_j \\ (false, ()), & \text{otherwise} \end{cases}$$

$$disjoint\,(\_, \_) : \mathbb{D} \times \mathbb{D} \to \mathbb{B}^1 : disjoint\,((a_1, \ldots, a_n, ()), (b_1, \ldots, b_m, ()))$$

$$= \begin{cases} (true, ()), & \forall i \in 1..n : \forall j \in 1..m : a_i \neq b_j \\ (false, ()), & \text{otherwise} \end{cases}$$

$$onlyexists_{\{\, a_k = A_k \mid k \in 1..n \,\}, a_i}\,(\_) : \{\, a_k = A_k \mid k \in 1..n \,\}^1 \to \mathbb{B}^1 :$$

$$onlyexists_{\{\, a_k = A_k \mid k \in 1..n \,\}, a_i}\,((\{\, a_k = v_k \mid k \in 1..n \,\}, ()))$$

$$= \begin{cases} (true, ()), & v_i \neq () \wedge \forall j \in 1..n : j \neq i \Rightarrow v_j = () \\ (false, ()), & \text{otherwise} \end{cases}$$

$$project_{\left\{\, a_k = A_k^{l_k : u_k} \mid k \in 1..n \,\right\}, a_i}\,(\_) : \left\{\, a_k = A_k^{l_k : u_k} \mid k \in 1..n \,\right\}^* \to A_i^* :$$

$$project_{\left\{\, a_k = A_k^{l_k : u_k} \mid k \in 1..n \,\right\}, a_i}\,((\{\, a_k = v_{k1} \mid k \in 1..n \,\}, \ldots, \{\, a_k = v_{km} \mid k \in 1..n \,\}, ()))$$

$$= flatten_{A_i, m}\,(v_{i1}, \ldots, v_{im})$$

$$\_\, eq \,\_ : \mathbb{D} \times \mathbb{D} \to \mathbb{B}^1 : (a_1, \ldots, a_n, ())\, eq\, (b_1, \ldots, b_m, ())$$

$$= \begin{cases} (true, ()), & n = m \wedge \forall i \in 1..n : a_i = b_i \\ (false, ()), & \text{otherwise} \end{cases}$$

$$\_neq\_ : \mathbb{D} \times \mathbb{D} \to \mathbb{B}^1 : (a_1, \ldots, a_n, ())\, neq\, (b_1, \ldots, b_m, ())$$

$$= \begin{cases} (true, ()), & n \neq m \vee \forall i \in 1..n : a_i \neq b_i \\ (false, ()), & \text{otherwise} \end{cases}$$

$$\_alleq\_ : \mathbb{D} \times \mathbb{D}_1^1 \to \mathbb{B}^1 : (a_1, \ldots, a_n, ())\, alleq\, (b, ())$$

$$= \begin{cases} (true, ()), & \forall i \in 1..n : a_i = b \\ (false, ()), & \text{otherwise} \end{cases}$$

$$\_allneq\_ : \mathbb{D} \times \mathbb{D}_1^1 \to \mathbb{B}^1 : (a_1, \ldots, a_n, ())\, allneq\, (b, ())$$

$$= \begin{cases} (true, ()), & \forall i \in 1..n : a_i \neq b \\ (false, ()), & \text{otherwise} \end{cases}$$

$$\_anyeq\_ : \mathbb{D} \times \mathbb{D}_1^1 \to \mathbb{B}^1 : (a_1, \ldots, a_n, ())\, anyeq\, (b, ())$$

$$= \begin{cases} (true, ()), & \exists i \in 1..n : a_i = b \\ (false, ()), & \text{otherwise} \end{cases}$$

$$\_anyneq\_ : \mathbb{D} \times \mathbb{D}_1^1 \to \mathbb{B}^1 : (a_1, \ldots, a_n, ())\, anyneq\, (b, ())$$

$$= \begin{cases} (true, ()), & \exists i \in 1..n : a_i \neq b \\ (false, ()), & \text{otherwise} \end{cases}$$

$$\_plus_A\_ : A^1 \times A^1 \to A^1 : (a, ())\, plus_A\, (b, ()) = (a + b, ())$$

$$\_subtract_A\_ : A^1 \times A^1 \to A^1 : (a, ())\, subtract_A\, (b, ()) = (a - b, ())$$

$$\_mult_A\_ : A^1 \times A^1 \to A^1 : (a, ())\, mult_A\, (b, ()) = (a * b, ())$$

$$\_div\_ : \mathbb{R}^1 \times \mathbb{R}^1 \to \mathbb{R}^1 : (a, ())\, div\, (b, ()) = (a/b, ())$$

$$onlyelement\,(\_) : \mathbb{D} \to \mathbb{D} : onlyelement\,((a_1, \ldots, a_n, ()))$$

$$= \begin{cases} (a_1, ()), & n = 1 \\ (), & \text{otherwise} \end{cases}$$

Note: equality is checked deeply, i.e. recursively on attributes of records.
Given $f : A_1 \times \cdots \times A_n \to B$ where $A_1, \ldots, A_n \subset \mathbb{D}$ and $B \subset \mathbb{D}$, let

$$\hat{f} : \mathbb{D}_\perp^n \to \mathbb{D}_\perp : \hat{f}(a_1, \ldots, a_n) = \begin{cases} f(a_1, \ldots, a_n), & (a_1, \ldots, a_n) \in \operatorname{Dom} f \\ \perp, & \text{otherwise.} \end{cases}$$

3.3 Semantics of Expressions

Some denotations depend on the type derivation of an expression. For this reason, I will evaluate typing derivations instead of expressions. However, because I only need this in a few cases, I will often omit the derivation, i.e. I will write $[\![e]\!]$ instead of $[\![\mathcal{D} :: \emptyset \vdash e : T\ C]\!]$ if the type $T\ C$ and the derivation $\mathcal{D}$ are unimportant.

Values $[\![v]\!]$.

$$[\![True]\!] = (true, ()) \qquad \text{E-TRUE}$$

$$[\![False]\!] = (false, ()) \qquad \text{E-FALSE}$$

$$[\![i]\!] = (i, ()) \qquad \text{E-INT}$$

$$[\![r]\!] = (r, ()) \qquad \text{E-NUMBER}$$

Expressions $[\![e]\!]$.

$$[\![\mathcal{D} :: \emptyset \vdash [e_1, \dots, e_n] : T\ C]\!] = flatten_{[\![T]\!],n}\left([\![e_1]\!], \dots, [\![e_n]\!]\right) \qquad \text{E-LIST}$$

$$[\![e_1\ \texttt{or}\ e_2]\!] = [\![e_1]\!]\ \widehat{or}\ [\![e_2]\!] \qquad \text{E-OR}$$

$$[\![e_1\ \texttt{and}\ e_2]\!] = [\![e_1]\!]\ \widehat{and}\ [\![e_2]\!] \qquad \text{E-AND}$$

$$[\![\texttt{not}\ e]\!] = \widehat{not}\left([\![e]\!]\right) \qquad \text{E-NOT}$$

$$[\![e\ \texttt{exists}]\!] = \begin{cases} (false, ()), & [\![e]\!] = () \\ (true, ()), & \text{otherwise} \end{cases} \qquad \text{E-EXISTS}$$

$$[\![e\ \texttt{single exists}]\!] = \begin{cases} (true, ()), & \widehat{count}\left([\![e]\!]\right) = (1, ()) \\ (false, ()), & \text{otherwise} \end{cases} \qquad \text{E-SINGLEEXISTS}$$

$$[\![e\ \texttt{multiple exists}]\!] = \begin{cases} (true, ()), & \widehat{count}\left([\![e]\!]\right) = (k, ()) \land k \geq 2 \\ (false, ()), & \text{otherwise} \end{cases} \qquad \text{E-MULTIPLEEXISTS}$$

$$[\![e_1\ \texttt{contains}\ e_2]\!] = \widehat{contains}\left([\![e_1]\!], [\![e_2]\!]\right) \qquad \text{E-CONTAINS}$$

$$[\![e_1\ \texttt{disjoint}\ e_2]\!] = \widehat{disjoint}\left([\![e_1]\!], [\![e_2]\!]\right) \qquad \text{E-DISJOINT}$$

$$[\![e_1 \texttt{=} e_2]\!] = [\![e_1]\!]\ \widehat{eq}\ [\![e_2]\!] \qquad \text{E-EQUALS}$$

$$[\![e_1 \texttt{<>} e_2]\!] = [\![e_1]\!]\ \widehat{neq}\ [\![e_2]\!] \qquad \text{E-NOTEQUALS}$$

$$[\![e_1\ \texttt{all =}\ e_2]\!] = [\![e_1]\!]\ \widehat{alleq}\ [\![e_2]\!] \qquad \text{E-ALLEQUALS}$$

$$[\![e_1\ \texttt{all <>}\ e_2]\!] = [\![e_1]\!]\ \widehat{allneq}\ [\![e_2]\!] \qquad \text{E-ALLNOTEQUALS}$$

$$[\![e_1\ \texttt{any =}\ e_2]\!] = [\![e_1]\!]\ \widehat{anyeq}\ [\![e_2]\!] \qquad \text{E-ANYEQUALS}$$

$$[\![e_1\ \texttt{any <>}\ e_2]\!] = [\![e_1]\!]\ \widehat{anyneq}\ [\![e_2]\!] \qquad \text{E-ANYNOTEQUALS}$$

$$\left[\!\!\left[\frac{\mathcal{D}_1 :: \emptyset \vdash e_1 : T\ (1..1) \qquad \mathcal{D}_2 :: \emptyset \vdash e_2 : T\ (1..1)}{\emptyset \vdash e_1 \texttt{+} e_2 : T\ (1..1)}\right]\!\!\right] = [\![e_1]\!]\ \widehat{plus}_{[\![T]\!]}\ [\![e_2]\!] \qquad \text{E-PLUS}$$

$$\left[\!\!\left[\frac{\mathcal{D}_1 :: \emptyset \vdash e_1 : T\ (1..1) \qquad \mathcal{D}_2 :: \emptyset \vdash e_2 : T\ (1..1)}{\emptyset \vdash e_1 \texttt{-} e_2 : T\ (1..1)}\right]\!\!\right] = [\![e_1]\!]\ \widehat{subtract}_{[\![T]\!]}\ [\![e_2]\!] \qquad \text{E-Subt}$$

$$\left[\!\!\left[\frac{\mathcal{D}_1 :: \emptyset \vdash e_1 : T\ (1..1) \qquad \mathcal{D}_2 :: \emptyset \vdash e_2 : T\ (1..1)}{\emptyset \vdash e_1 \texttt{*} e_2 : T\ (1..1)}\right]\!\!\right] = [\![e_1]\!]\ \widehat{mult}_{[\![T]\!]}\ [\![e_2]\!] \qquad \text{E-Mult}$$

$$[\![e_1 \texttt{/} e_2]\!] = [\![e_1]\!]\ \widehat{div}\ [\![e_2]\!] \qquad \text{E-Div}$$

$$[\![e\ \texttt{count}]\!] = \widehat{count}\,([\![e]\!]) \qquad \text{E-Count}$$

$$\left[\!\!\left[\frac{\mathcal{D} :: \emptyset \vdash e : D\ C}{\emptyset \vdash e \texttt{->} a : T\ C_a}\right]\!\!\right] = \widehat{project}_{[\![D]\!],a}\,([\![e]\!]) \qquad \text{E-Project}$$

$$[\![\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3]\!] = ([\![e_1]\!]) \widehat{\rightarrow} [\![e_2]\!]\ []\ [\![e_3]\!] \qquad \text{E-If}$$

$$\frac{\text{inputs}(F) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n}{[\![F(e_1, \ldots, e_n)]\!] = [\![\ [\ a_1 \mapsto e_1, \ldots, a_n \mapsto e_n\ ]\ \text{op}(F)]\!]} \qquad \text{E-Func}$$

$$\frac{\text{attrs}(D) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n}{[\![D\,\{\ a_1 : e_1, \ldots, a_n : e_n\ \}]\!] = (\{\ a_1 = [\![e_1]\!], \ldots, a_n = [\![e_n]\!]\ \}\,, ())} \qquad \text{E-Construct}$$

$$\left[\!\!\left[\frac{\emptyset \vdash e : D\ (1..1)}{\emptyset \vdash e \texttt{->} a\ \texttt{only exists} : \texttt{boolean}\ \ (1..1)}\right]\!\!\right] = \widehat{onlyexists}_{[\![D]\!],a}\,([\![e]\!]) \qquad \text{E-OnlyExists}$$

$$[\![e\ \texttt{only-element}]\!] = \widehat{onlyelement}\,([\![e]\!]) \qquad \text{E-OnlyElement}$$

Note: equality between two empty lists (i.e. true) is different than the usual equality with null (i.e. always false) in other programming languages (and the official Rosetta documentation).

4 Typing

4.1 Declarative Typing

Some auxiliary definitions.

$$\begin{aligned}
\inf((l..u)) &= l \\
\sup((l..u)) &= u \\
\text{subcardinality}((l_1..u_1), (l_2..u_2)) &= l_1 \geq l_2 \wedge u_1 \leq u_2 \\
\text{comparable}(T_1, T_2) &= T_1 <: T_2 \vee T_2 <: T_1 \\
\text{overlap}((l_1..u_1), (l_2..u_2)) &= u_1 \geq l_2 \wedge u_2 \geq l_1 \\
\text{comparable}^*(T_1\ C_1, T_2\ C_2) &= \text{comparable}(T_1, T_2) \wedge \text{overlap}(C_1, C_2) \\
\text{union}((l_1..u_1), (l_2..u_2)) &= (\min(l_1, l_2)..\max(u_1, u_2))
\end{aligned}$$

Subtyping $S <: T$.

$$T <: T \qquad \text{S-Refl}$$

$$\frac{S <: U \qquad U <: T}{S <: T} \quad \textsc{S-Trans}$$

$$\texttt{int} <: \texttt{number} \quad \textsc{S-Num}$$

$$\frac{\mathrm{DT}(D) = \texttt{type}\, D\, \texttt{extends}\, E : \ldots}{D <: E} \quad \textsc{S-Extends}$$

List subtyping $T_1\ C_1 <:^* T_2\ C_2$.

$$\frac{T_1 <: T_2 \qquad \text{subcardinality}(C_1, C_2)}{T_1\ C_1 <:^* T_2\ C_2} \quad \textsc{S-Card}$$

Typing rules $\Gamma \vdash e : T\ C$.

$$\frac{\Gamma \vdash e_1 : \texttt{boolean}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{boolean}\ (1..1)}{\Gamma \vdash e_1\, \texttt{or}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-Or}$$

$$\frac{\Gamma \vdash e_1 : \texttt{boolean}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{boolean}\ (1..1)}{\Gamma \vdash e_1\, \texttt{and}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-And}$$

$$\frac{\Gamma \vdash e : T\ C \qquad \text{subcardinality}((0..1), C)}{\Gamma \vdash e\, \texttt{exists} : \texttt{boolean}\ (1..1)} \quad \textsc{T-Exists}$$

$$\frac{\Gamma \vdash e : T\ C \qquad \text{subcardinality}((1..1), C) \qquad C \neq (1..1)}{\Gamma \vdash e\, \texttt{single}\, \texttt{exists} : \texttt{boolean}\ (1..1)} \quad \textsc{T-SingleExists}$$

$$\frac{\Gamma \vdash e : T\ C \qquad \text{subcardinality}((1..2), C)}{\Gamma \vdash e\, \texttt{multiple}\, \texttt{exists} : \texttt{boolean}\ (1..1)} \quad \textsc{T-MultipleExists}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C_1 \qquad \Gamma \vdash e_2 : T_2\ C_2 \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\, \texttt{contains}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-Contains}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C_1 \qquad \Gamma \vdash e_2 : T_2\ C_2 \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\, \texttt{disjoint}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-Disjoint}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C_1 \qquad \Gamma \vdash e_2 : T_2\ C_2 \qquad \text{comparable}^*(T_1\ C_1, T_2\ C_2)}{\Gamma \vdash e_1\, \texttt{=}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-Equals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C_1 \qquad \Gamma \vdash e_2 : T_2\ C_2 \qquad \text{comparable}^*(T_1\ C_1, T_2\ C_2)}{\Gamma \vdash e_1\, \texttt{<>}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-NotEquals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C \qquad \Gamma \vdash e_2 : T_2\ (1..1) \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\, \texttt{all}\, \texttt{=}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-AllEquals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C \qquad \Gamma \vdash e_2 : T_2\ (1..1) \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\, \texttt{all}\, \texttt{<>}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-AllNotEquals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C \qquad \Gamma \vdash e_2 : T_2\ (1..1) \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\, \texttt{any}\, \texttt{=}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-AnyEquals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C \qquad \Gamma \vdash e_2 : T_2\ (1..1) \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\, \texttt{any}\, \texttt{<>}\, e_2 : \texttt{boolean}\ (1..1)} \quad \textsc{T-AnyNotEquals}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{int}\ (1..1)}{\Gamma \vdash e_1 \,\texttt{+}\, e_2 : \texttt{int}\ (1..1)} \quad \text{T-PlusInt}$$

$$\frac{\Gamma \vdash e_1 : \texttt{number}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{number}\ (1..1)}{\Gamma \vdash e_1 \,\texttt{+}\, e_2 : \texttt{number}\ (1..1)} \quad \text{T-PlusNumber}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{int}\ (1..1)}{\Gamma \vdash e_1 \,\texttt{*}\, e_2 : \texttt{int}\ (1..1)} \quad \text{T-MultInt}$$

$$\frac{\Gamma \vdash e_1 : \texttt{number}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{number}\ (1..1)}{\Gamma \vdash e_1 \,\texttt{*}\, e_2 : \texttt{number}\ (1..1)} \quad \text{T-MultNumber}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{int}\ (1..1)}{\Gamma \vdash e_1 \,\texttt{-}\, e_2 : \texttt{int}\ (1..1)} \quad \text{T-SubtInt}$$

$$\frac{\Gamma \vdash e_1 : \texttt{number}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{number}\ (1..1)}{\Gamma \vdash e_1 \,\texttt{-}\, e_2 : \texttt{number}\ (1..1)} \quad \text{T-SubtNumber}$$

$$\frac{\Gamma \vdash e_1 : \texttt{number}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{number}\ (1..1)}{\Gamma \vdash e_1 \,\texttt{/}\, e_2 : \texttt{number}\ (1..1)} \quad \text{T-Division}$$

$$\frac{\Gamma \vdash e : T\ C}{\Gamma \vdash e\,\texttt{count} : \texttt{int}\ (1..1)} \quad \text{T-Count}$$

$$\frac{\Gamma \vdash e : D\ (l..u) \qquad \text{attrs}(D) = a_1\ T_1\ (l_1..u_1), \ldots, a_n\ T_n\ (l_n..u_n)}{\Gamma \vdash e\,\texttt{->}\,a_k : T_k\ (l * l_k..u * u_k)} \quad \text{T-Project}$$

$$\frac{\Gamma \vdash e_1 : \texttt{boolean}\ (1..1) \qquad \Gamma \vdash e_2 : T\ C \qquad \Gamma \vdash e_3 : T\ C}{\Gamma \vdash \texttt{if}\, e_1\, \texttt{then}\, e_2\, \texttt{else}\, e_3 : T\ C} \quad \text{T-If}$$

$$\frac{\begin{array}{c}\forall i \in 1..n : \Gamma \vdash e_i : T_i\ C_i \\ \text{inputs}(F) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n \qquad \text{output}(F) = a\ T\ C\end{array}}{\Gamma \vdash F(e_1, \ldots, e_n) : T\ C} \quad \text{T-Func}$$

$$\frac{\forall i \in 1..n : \Gamma \vdash e_i : T_i\ C_i \qquad \text{attrs}(D) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n}{\Gamma \vdash D\,\{\, a_1 = e_1, \ldots, a_n = e_n \,\} : D\ (1..1)} \quad \text{T-Construct}$$

$$\frac{x : T\ C \in \Gamma}{\Gamma \vdash x : T\ C} \quad \text{T-Var}$$

$$\Gamma \vdash \texttt{True} : \texttt{boolean}\ (1..1) \quad \text{T-True}$$

$$\Gamma \vdash \texttt{False} : \texttt{boolean}\ (1..1) \quad \text{T-False}$$

$$\Gamma \vdash i : \texttt{int}\ (1..1) \quad \text{T-Int}$$

$$\Gamma \vdash r : \texttt{number}\ (1..1) \quad \text{T-Number}$$

$$\frac{\forall i \in 1..n : \Gamma \vdash e_i : T\ (l_i..u_i)}{\Gamma \vdash [e_1, \ldots, e_n] : T\ (\sum_{i \in 1..n} l_i\ ..\ \sum_{i \in 1..n} u_i)} \quad \text{T-List}$$

$$\frac{\Gamma \vdash e : D\ (1..1) \qquad \mathrm{attrs}(D) = a_1\ T_1\ C_1, \dots, a_n\ T_n\ C_n}{\Gamma \vdash e \,\texttt{->}\, a_k\ \texttt{only exists} : \texttt{boolean}\ (1..1)} \quad \textsc{T-OnlyExists}$$

$$\frac{\Gamma \vdash e : T\ C}{\Gamma \vdash e\ \texttt{only-element} : T\ (0..1)} \quad \textsc{T-OnlyElement}$$

$$\frac{\Gamma \vdash e : S \qquad S <: T}{\Gamma \vdash e : T} \quad \textsc{T-Sub}$$

Typing function declarations F OK.

$$\frac{\begin{array}{c}\mathrm{inputs}(F) = a_1\ T_1\ C_1, \dots, a_n\ T_n\ C_n \\ \mathrm{output}(F) = a\ T\ C \qquad a_1 : T_1\ C_1, \dots, a_n : T_n\ C_n \vdash \mathrm{op}(F) : T\ C\end{array}}{F\ \mathrm{OK}}$$

Note: for equality, there are two sensible choices as premises. Either $\Gamma \vdash e_1 : T\ C_1$ and $\Gamma \vdash e_2 : T\ C_2$ or $\Gamma \vdash e_1 : T\ C$ and $\Gamma \vdash e_2 : T\ C$. The second possibility eliminates equality checks that are always false because the operands can never have the same length.

4.2 Algorithmic Typing

These typing rules should be consistent with the declarative version, but they are defined in a way that is more straightforward to implement, because every rule is syntax-directed.

Introduce a new type `nothing` (i.e. the bottom type).
Join of basic types $\mathrm{join}(T_1, T_2)$.

$$\mathrm{join}(T, T) = T$$

$$\mathrm{join}(T_1, T_2) = \mathrm{join}(T_2, T_1)$$

$$\mathrm{join}(\texttt{int}, \texttt{number}) = \texttt{number}$$

$$\frac{E \in \mathrm{supertypes}(D)}{\mathrm{join}(D, E) = E}$$

$$\frac{\begin{array}{c}E \notin \mathrm{supertypes}(D) \\ \mathrm{DT}(E) = \texttt{type}\ E\ \texttt{extends}\ E' : \dots \qquad T = \mathrm{join}(D, E')\end{array}}{\mathrm{join}(D, E) = T}$$

$$\mathrm{join}(\texttt{nothing}, T) = T$$

Extension of join to $n \in \mathbb{N}$ basic types.

$$\mathrm{join}() = \texttt{nothing}$$

$$\mathrm{join}(T) = T$$

$$\frac{n \geq 3 \qquad T' = \mathrm{join}(T_2, \dots, T_n) \qquad T = \mathrm{join}(T_1, T')}{\mathrm{join}(T_1, \dots, T_n) = T}$$

Join for types $\text{join}^*(T_1\ C_1, T_2\ C_2)$.

$$\frac{T = \text{join}(T_1, T_2) \qquad C = \text{union}(C_1, C_2)}{\text{join}^*(T_1\ C_1, T_2\ C_2) = T\ C}$$

Basic subtyping $S <:: T$.

$$T <:: T \qquad \textsc{SA-Refl}$$

$$\texttt{int} <:: \texttt{number} \qquad \textsc{SA-Num}$$

$$\frac{E \in \text{supertypes}(D)}{D <:: E} \qquad \textsc{SA-Super!}$$

$$\texttt{nothing} <:: T \qquad \textsc{SA-Nothing}$$

Subtyping $T_1\ C_1 <::^* T_2\ C_2$.

$$\frac{T_1 <:: T_2 \qquad \text{subcardinality}(C_1, C_2)}{T_1\ C_1 <::^* T_2\ C_2} \qquad \textsc{SA-Card}$$

Typing rules $\Gamma \vdash e : T\ C$.

$$\frac{\Gamma \vdash e_1 : \texttt{boolean}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{boolean}\ (1..1)}{\Gamma \vdash e_1\, \texttt{or}\, e_2 : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-Or}$$

$$\frac{\Gamma \vdash e_1 : \texttt{boolean}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{boolean}\ (1..1)}{\Gamma \vdash e_1\, \texttt{and}\, e_2 : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-And}$$

$$\frac{\Gamma \vdash e : T\ C \qquad \text{subcardinality}((0..1), C)}{\Gamma \vdash e\, \texttt{exists} : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-Exists}$$

$$\frac{\Gamma \vdash e : T\ C \qquad \text{subcardinality}((1..1), C) \qquad C \neq (1..1)}{\Gamma \vdash e\, \texttt{single}\, \texttt{exists} : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-SingleExists}$$

$$\frac{\Gamma \vdash e : T\ C \qquad \text{subcardinality}((1..2), C)}{\Gamma \vdash e\, \texttt{multiple}\, \texttt{exists} : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-MultipleExists}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C_1 \qquad \Gamma \vdash e_2 : T_2\ C_2 \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\, \texttt{contains}\, e_2 : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-Contains}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C_1 \qquad \Gamma \vdash e_2 : T_2\ C_2 \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\, \texttt{disjoint}\, e_2 : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-Disjoint}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C_1 \qquad \Gamma \vdash e_2 : T_2\ C_2 \qquad \text{comparable}^*(T_1\ C_1, T_2\ C_2)}{\Gamma \vdash e_1\, \texttt{=}\, e_2 : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-Equals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C_1 \qquad \Gamma \vdash e_2 : T_2\ C_2 \qquad \text{comparable}^*(T_1\ C_1, T_2\ C_2)}{\Gamma \vdash e_1\, \texttt{<>}\, e_2 : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-NotEquals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C \qquad \Gamma \vdash e_2 : T_2\ (1..1) \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\, \texttt{all}\, \texttt{=}\, e_2 : \texttt{boolean}\ (1..1)} \qquad \textsc{TA-AllEquals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C \qquad \Gamma \vdash e_2 : T_2\ (1..1) \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\ \texttt{all <>}\ e_2 : \texttt{boolean}\ (1..1)} \quad \text{TA-AllNotEquals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C \qquad \Gamma \vdash e_2 : T_2\ (1..1) \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\ \texttt{any =}\ e_2 : \texttt{boolean}\ (1..1)} \quad \text{TA-AnyEquals}$$

$$\frac{\Gamma \vdash e_1 : T_1\ C \qquad \Gamma \vdash e_2 : T_2\ (1..1) \qquad \text{comparable}(T_1, T_2)}{\Gamma \vdash e_1\ \texttt{any <>}\ e_2 : \texttt{boolean}\ (1..1)} \quad \text{TA-AnyNotEquals}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{int}\ (1..1)}{\Gamma \vdash e_1\ \texttt{+}\ e_2 : \texttt{int}\ (1..1)} \quad \text{TA-PlusInt}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : T_1\ (1..1) \qquad \Gamma \vdash e_2 : T_2\ (1..1) \\ T_1 <:: \texttt{number} \qquad T_2 <:: \texttt{number} \qquad T_1 \neq \texttt{int} \vee T_2 \neq \texttt{int}\end{array}}{\Gamma \vdash e_1\ \texttt{+}\ e_2 : \texttt{number}\ (1..1)} \quad \text{TA-PlusNumber!}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{int}\ (1..1)}{\Gamma \vdash e_1\ \texttt{*}\ e_2 : \texttt{int}\ (1..1)} \quad \text{TA-MultInt}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : T_1\ (1..1) \qquad \Gamma \vdash e_2 : T_2\ (1..1) \\ T_1 <:: \texttt{number} \qquad T_2 <:: \texttt{number} \qquad T_1 \neq \texttt{int} \vee T_2 \neq \texttt{int}\end{array}}{\Gamma \vdash e_1\ \texttt{*}\ e_2 : \texttt{number}\ (1..1)} \quad \text{TA-MultNumber!}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int}\ (1..1) \qquad \Gamma \vdash e_2 : \texttt{int}\ (1..1)}{\Gamma \vdash e_1\ \texttt{-}\ e_2 : \texttt{int}\ (1..1)} \quad \text{TA-SubtInt}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : T_1\ (1..1) \qquad \Gamma \vdash e_2 : T_2\ (1..1) \\ T_1 <:: \texttt{number} \qquad T_2 <:: \texttt{number} \qquad T_1 \neq \texttt{int} \vee T_2 \neq \texttt{int}\end{array}}{\Gamma \vdash e_1\ \texttt{-}\ e_2 : \texttt{number}\ (1..1)} \quad \text{TA-SubtNumber!}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : T_1\ (1..1) \\ \Gamma \vdash e_2 : T_2\ (1..1) \qquad T_1 <:: \texttt{number} \qquad T_2 <:: \texttt{number}\end{array}}{\Gamma \vdash e_1\ \texttt{/}\ e_2 : \texttt{number}\ (1..1)} \quad \text{TA-Division!}$$

$$\frac{\Gamma \vdash e : T\ C}{\Gamma \vdash e\ \texttt{count} : \texttt{int}\ (1..1)} \quad \text{TA-Count}$$

$$\frac{\Gamma \vdash e : D\ (l..u) \qquad \text{attrs}(D) = a_1\ T_1\ (l_1..u_1), \ldots, a_n\ T_n\ (l_n..u_n)}{\Gamma \vdash e\ \texttt{->}\ a_k : T_k\ (l * l_k..u * u_k)} \quad \text{TA-Project}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \texttt{boolean}\ (1..1) \\ \Gamma \vdash e_2 : T_1\ C_1 \qquad \Gamma \vdash e_3 : T_2\ C_3 \qquad T\ C = \text{join}^*(T_1\ C_1, T_2\ C_2)\end{array}}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : T\ C} \quad \text{TA-If!}$$

$$\frac{\begin{array}{c}\text{inputs}(F) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n \qquad \forall i \in 1..n : \Gamma \vdash e_i : T_i'\ C_i' \\ \forall i \in 1..n : T_i'\ C_i' <:: T_n\ C_n \qquad \text{output}(F) = a\ T\ C\end{array}}{\Gamma \vdash F(e_1, \ldots, e_n) : T\ C} \quad \text{TA-Func!}$$

$$\frac{\mathrm{attrs}(D) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n \qquad \forall i \in 1..n : \Gamma \vdash e_i : T_i'\ C_i' \qquad \forall i \in 1..n : T_i'\ C_i' <:: T_i\ C_i}{\Gamma \vdash D\,\{\, a_1 = e_1, \ldots, a_n = e_n \,\} : D\ (1..1)} \quad \textsc{TA-Construct!}$$

$$\frac{x : T\ C \in \Gamma}{\Gamma \vdash x : T\ C} \quad \textsc{TA-Var}$$

$$\Gamma \vdash \texttt{True} : \texttt{boolean}\ \ (1..1) \quad \textsc{TA-True}$$

$$\Gamma \vdash \texttt{False} : \texttt{boolean}\ \ (1..1) \quad \textsc{TA-False}$$

$$\Gamma \vdash r : \texttt{number}\ \ (1..1) \quad \textsc{TA-Number}$$

$$\Gamma \vdash i : \texttt{int}\ \ (1..1) \quad \textsc{TA-Int}$$

$$\frac{\forall i \in 1..n : \Gamma \vdash e_i : T_i\ (l_i..u_i) \qquad T = \mathrm{join}(T_1, \ldots, T_n)}{\Gamma \vdash [e_1, \ldots, e_n] : T\ (\sum_{i\in 1..n} l_i\ ..\ \sum_{i \in 1..n} u_i)} \quad \textsc{TA-List!}$$

$$\frac{\Gamma \vdash e : D\ (1..1) \qquad \mathrm{attrs}(D) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n}{\Gamma \vdash e\,\texttt{->}\,a_k\,\texttt{only}\,\texttt{exists} : \texttt{boolean}\ \ (1..1)} \quad \textsc{TA-OnlyExists}$$

$$\frac{\Gamma \vdash e : T\ C}{\Gamma \vdash e\,\texttt{only-element} : T\ (0..1)} \quad \textsc{TA-OnlyElement}$$

Typing function declarations F OK.

$$\frac{\mathrm{inputs}(F) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n \qquad \mathrm{output}(F) = a\ T\ C \qquad a_1 : T_1\ C_1, \ldots, a_n : T_n\ C_n \vdash \mathrm{op}(F) : T'\ C' \qquad T'\ C' <:: T\ C}{F\ \mathrm{OK}} \quad !$$

5 Code Generation

Goals of target program.

- Formatting: it follows the naming conventions of the target language and is formatted in a readable, human-friendly manner.
- API: it follows the conventions of the target language for exposing an API (getters/setters, privacy, etc).

Generation of entity declarations. $[\![\texttt{type}\,D : a_1\ T_1\ C_1 \ldots a_n\ T_n\ C_n]\!]_{\mathcal{J}} =$

```
public class javaId(D) {
    ∀i ∈ 1..n :
        ⟦a_i T_i C_i⟧_J
}
```

TODO: package declaration.
Generation of extending entity declarations. $[\![\texttt{type}\,D\,\texttt{extends}\,E : a_1\ T_1\ C_1 \ldots a_n\ T_n\ C_n]\!]_{\mathcal{J}} =$

```
public class javaId(D) extends javaId(E) {
    ∀i ∈ 1..n :
        ⟦a_i T_i C_i⟧_J
```

```
}
```

Generation of attributes. $[\![a\ T\ C]\!]_{\mathcal{J}} =$

```
protected final ⟦T C⟧_𝒥 javaId(a);
```

Simple version: everything is a list.
Generation of types. $[\![T\ C]\!]_{\mathcal{J}} =$

```
List<? extends ⟦T⟧_𝒥>
```

Generation of basic types.

$$
\begin{aligned}
[\![D]\!]_{\mathcal{J}} &= \mathrm{javaId}(D) \\
[\![\texttt{boolean}]\!]_{\mathcal{J}} &= \texttt{Boolean} \\
[\![\texttt{int}]\!]_{\mathcal{J}} &= \texttt{Integer} \\
[\![\texttt{number}]\!]_{\mathcal{J}} &= \texttt{BigDecimal} \\
[\![\texttt{nothing}]\!]_{\mathcal{J}} &= \texttt{Void}
\end{aligned}
$$

Note: given this translation, the rules SA-Num and SA-Nothing have no corresponding rules in Java, so this will bring a need for coercions!
Generation of function declarations.
$[\![\texttt{func}\ F : \texttt{inputs} : a_1\ T_1\ C_1 \ldots a_n\ T_n\ C_n\ \texttt{output} : a\ T\ C\ \texttt{assign-output} : e]\!]_{\mathcal{J}} =$

```
public class javaId(F) {
  public ⟦T C⟧_𝒥 evaluate(∀i ∈ 1..n : ⟦a_i T_i C_i⟧_𝒥) {
    return ⟦e⟧_𝒥;
  }
}
```

Generation of expressions. Implementation of most expressions is straight-forward.

$$
\begin{aligned}
[\![\texttt{True}]\!]_{\mathcal{J}} &= \texttt{Collections.singletonList(true)} \\
[\![\texttt{False}]\!]_{\mathcal{J}} &= \texttt{Collections.singletonList(false)} \\
[\![i]\!]_{\mathcal{J}} &= \texttt{Collections.singletonList(}i\texttt{)} \\
[\![r]\!]_{\mathcal{J}} &= \texttt{Collections.singletonList(BigDecimal.valueOf(}r\texttt{))} \\
[\![[e_1, \ldots, e_n]]\!]_{\mathcal{J}} &= \texttt{Stream.of(}[\![e_1]\!]_{\mathcal{J}}\texttt{, ..., }[\![e_n]\!]_{\mathcal{J}}\texttt{).flatMap(Collection::stream).collect(Collectors.toList())} \\
[\![e_1\ \texttt{or}\ e_2]\!]_{\mathcal{J}} &= \texttt{Streams.zip(}[\![e_1]\!]_{\mathcal{J}}\texttt{.stream(), }[\![e_2]\!]_{\mathcal{J}}\texttt{.stream(), Boolean::logicalOr).collect(Collectors.toList(} \\
[\![e_1\ \texttt{and}\ e_2]\!]_{\mathcal{J}} &= \texttt{Streams.zip(}[\![e_1]\!]_{\mathcal{J}}\texttt{.stream(), }[\![e_2]\!]_{\mathcal{J}}\texttt{.stream(), Boolean::logicalAnd).collect(Collectors.toList} \\
[\![\texttt{not}\ e]\!]_{\mathcal{J}} &= [\![e_1]\!]_{\mathcal{J}}\texttt{.stream().map(}\mathrm{freeId}(b)\texttt{ -> !}\mathrm{freeId}(b)\texttt{).collect(Collectors.toList())}
\end{aligned}
$$

Preamble:

```
package dummypackage;

```

```java
import java.math.BigDecimal;
import java.util.Collection;
import java.util.Collections;
import java.util.stream.Stream;
import java.util.stream.Collectors;
import com.google.common.collect.Streams;
```