Specification of the Nouga DSL

Simon Cockx

January 31, 2022

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.

⟨MODEL⟩::= *root model:*
(⟨ED⟩ | ⟨FD⟩)*

⟨ED⟩::= *entity declarations:*
`type` *D* (`extends` *E*)? `:`
⟨AD⟩*

⟨FD⟩::= *function declarations:*
`func` *F* `:`
`inputs :` ⟨AD⟩*
`output :` ⟨AD⟩
`assign-output :` ⟨E⟩

⟨AD⟩::= *attribute declarations:*
a ⟨T⟩ ⟨CC⟩

⟨CC⟩::= *cardinality constraints:*
| `(` *l* `..` *k* `)` *bounded*
| `(` *l* `.. * )` *unbounded*

⟨E⟩::= *expressions:*
| ⟨LIT⟩
| *a*
| ⟨E⟩ `or` ⟨E⟩
| ⟨E⟩ `and` ⟨E⟩
| `not` ⟨E⟩
| ⟨E⟩ (`+` | `-`) ⟨E⟩
| ⟨E⟩ (`*` | `/`) ⟨E⟩
| *D* `{` (*a* `:` ⟨E⟩ (`,` *b* `:` ⟨E⟩)*)? `}`
| ⟨E⟩ `->` *a*
| ⟨E⟩ (`single` | `multiple`)? `exists`
| ⟨E⟩ `is absent`
| ⟨E⟩ `->` *a* `only exists`
| ⟨E⟩ `count`
| ⟨E⟩ `only-element`
| ⟨E⟩ (`all` | `any`)? (`=` | `<>`) ⟨E⟩
| ⟨E⟩ `contains` ⟨E⟩
| ⟨E⟩ `disjoint` ⟨E⟩
| `[` (⟨E⟩ (`,` ⟨E⟩)*)? `]`
| `if` ⟨E⟩ `then` ⟨E⟩ (`else` ⟨E⟩)?
| *F* `(` (⟨E⟩ (`,` ⟨E⟩)*)? `)`
| `(` ⟨E⟩ `)`

⟨LIT⟩::= *literals:*
| `True` | `False` *booleans*
| *i* *signed integers*
| *r* *signed decimals*
| `empty` *empty literal*

⟨T⟩::= *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

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

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 $\langle \mathrm{E} \rangle$)
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

Entity table $\mathrm{ET}(D)$ 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.

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

$$\frac{\mathrm{ET}(D) = \texttt{type}\, D : \ldots}{\mathrm{allattrs}(D) = \mathrm{attrs}(D)}$$

$$\frac{\mathrm{ET}(D) = \texttt{type}\, D \,\texttt{extends}\, E : \ldots}{\mathrm{allattrs}(D) = \mathrm{allattrs}(E), \mathrm{attrs}(D)}$$

Ancestors.

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

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

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

3.1 Semantics of Types

Semantics of item types $\mathcal{T}[\![T]\!]$.

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

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

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

$$\mathcal{T}[\![D]\!] = \bigcup_{E <: D} \mathcal{D}[\![E]\!]$$

Semantics of entities $\mathcal{D}[\![D]\!]$.

$$\begin{aligned}\mathcal{D}[\![D]\!] = \textbf{let } & a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n = \mathrm{allattrs}(D) \\ \textbf{in } & \{\, D \,\} \times \prod_{i \in 1..n} \mathcal{T}^*[\![T_i\ C_i]\!]\end{aligned}$$

Semantics of list types $\mathcal{T}^*[\![T\ C]\!]$.

$$\mathcal{T}^*[\![T\ (l..k)]\!] = (\mathcal{T}[\![T]\!])^{l:k}$$

3.2 Semantical Algebra

Semantic algebra.

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

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

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

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

Note: from the above definition, A^1 formally equals $A \times \mathit{Unit}$, 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)

$$\begin{aligned}
\mathrm{count}(_) &: \mathbb{D}^* \to \mathbb{Z} : \mathrm{count}([a_1, \ldots, a_n]) = n \\
\mathrm{flatten}_n(_, \ldots, _) &: (\mathbb{D}^*)^n \to \mathbb{D}^* : \mathrm{flatten}_n([a_{11}, \ldots, a_{1m_1}], \ldots, [a_{n1}, \ldots, a_{nm_n}]) \\
&= [a_{11}, \ldots, a_{1m_1}, \ldots, a_{nm_n}] \\
\mathrm{equals}(_, _) &: \mathbb{D} \times \mathbb{D} \to \mathbb{B} : \mathrm{equals}(x, y) \\
&= \begin{cases} \mathrm{equals}_{\mathbb{P}}(x, y), & \text{if } x \in \mathbb{P} \wedge y \in \mathbb{P} \\ \mathrm{equals}_{\mathbb{E}}(x, y), & \text{if } x \in \mathbb{E} \wedge y \in \mathbb{E} \\ \mathit{false}, & \text{otherwise} \end{cases} \\
\mathrm{equals}_{\mathbb{P}}(_, _) &: \mathbb{P} \times \mathbb{P} \to \mathbb{B} : \mathrm{equals}_{\mathbb{P}}(x, y)
\end{aligned}$$

$$= \begin{cases} \mathit{true}, & \text{if } x = y \\ \mathit{false}, & \text{otherwise} \end{cases}$$

$$\text{equals}_{\mathbb{E}}(\_,\_) : \mathbb{E} \times \mathbb{E} \to \mathbb{B} : \text{equals}_{\mathbb{E}}((D, v_1, \ldots, v_m), (E, w_1, \ldots, w_n))$$

$$= \begin{cases} \mathit{true}, & \text{if } D = E \wedge \forall i \in 1..n : \text{equals}^*(v_i, w_i) \\ \mathit{false}, & \text{otherwise} \end{cases}$$

$$\text{equals}^*(\_,\_) : \mathbb{D}^* \times \mathbb{D}^* \to \mathbb{B} : \text{equals}^*([x_1, \ldots, x_m], [y_1, \ldots, y_n])$$

$$= \begin{cases} \mathit{true}, & \text{if } m = n \wedge \forall i \in 1..n : \text{equals}(x_i, y_i) \\ \mathit{false}, & \text{otherwise} \end{cases}$$

$$\text{project}_a(\_) : \mathbb{E} \to \mathbb{D}^* : \text{project}_{a_i}((D, v_1, \ldots, v_n))$$

$$= \textbf{let } a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n = \text{allattrs}(D) \quad \textbf{in } v_i$$

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.

$$\mathcal{E}\,[\![\texttt{True}]\!]\,S = [\mathit{true}] \qquad \text{E-True}$$

$$\mathcal{E}\,[\![\texttt{False}]\!]\,S = [\mathit{false}] \qquad \text{E-False}$$

$$\mathcal{E}\,[\![i]\!]\,S = [i] \qquad \text{E-Int}$$

$$\mathcal{E}\,[\![r]\!]\,S = [r] \qquad \text{E-Number}$$

$$\mathcal{E}\,[\![x]\!]\,S = S(x) \qquad \text{E-Var}$$

$$\mathcal{E}\,[\![e_1 \texttt{ or } e_2]\!]\,S = \textbf{let } [x] = \mathcal{E}\,[\![e_1]\!]\,S,\ [y] = \mathcal{E}\,[\![e_2]\!]\,S \ \textbf{in } [x \vee y] \qquad \text{E-Or}$$

$$\mathcal{E}\,[\![e_1 \texttt{ and } e_2]\!]\,S = \textbf{let } [x] = \mathcal{E}\,[\![e_1]\!]\,S,\ [y] = \mathcal{E}\,[\![e_2]\!]\,S \ \textbf{in } [x \wedge y] \qquad \text{E-And}$$

$$\mathcal{E}\,[\![\texttt{not } e]\!]\,S = \textbf{let } [x] = \mathcal{E}\,[\![e]\!]\,S \ \textbf{in } [\neg x] \qquad \text{E-Not}$$

$$\mathcal{E}\,[\![e_1 \,\texttt{+}\, e_2]\!]\,S = \begin{aligned}[t] &\textbf{let } [x] = \mathcal{E}\,[\![e_1]\!]\,S, \\ &\qquad [y] = \mathcal{E}\,[\![e_2]\!]\,S \\ &\textbf{in } [x+y] \end{aligned} \qquad \text{E-PLUS}$$

$$\mathcal{E}\,[\![e_1 \,\texttt{-}\, e_2]\!]\,S = \begin{aligned}[t] &\textbf{let } [x] = \mathcal{E}\,[\![e_1]\!]\,S, \\ &\qquad [y] = \mathcal{E}\,[\![e_2]\!]\,S \\ &\textbf{in } [x-y] \end{aligned} \qquad \text{E-SUBT}$$

$$\mathcal{E}\,[\![e_1 \,\texttt{*}\, e_2]\!]\,S = \begin{aligned}[t] &\textbf{let } [x] = \mathcal{E}\,[\![e_1]\!]\,S, \\ &\qquad [y] = \mathcal{E}\,[\![e_2]\!]\,S \\ &\textbf{in } [x*y] \end{aligned} \qquad \text{E-MULT}$$

$$\mathcal{E}\,[\![e_1 \,\texttt{/}\, e_2]\!]\,S = \begin{aligned}[t] &\textbf{let } [x] = \mathcal{E}\,[\![e_1]\!]\,S, \\ &\qquad [y] = \mathcal{E}\,[\![e_2]\!]\,S \\ &\textbf{in } [x/y] \end{aligned} \qquad \text{E-DIV}$$

$$\mathcal{E}\,[\![D\,\{\,a_1 : e_1, \ldots, a_n : e_n\,\}]\!]\,S = [(D, \mathcal{E}\,[\![e_1]\!]\,S, \ldots, \mathcal{E}\,[\![e_n]\!]\,S)] \qquad \text{E-INSTANTIATE}$$

$$\mathcal{E}\,[\![e \,\texttt{->}\, a]\!]\,S = \begin{aligned}[t] &\textbf{let } [x_1, \ldots, x_n] = \mathcal{E}\,[\![e]\!]\,S \\ &\textbf{in } \operatorname{flatten}_n(\operatorname{project}_a(x_1), \ldots, \operatorname{project}_a(x_n)) \end{aligned} \qquad \text{E-PROJECT}$$

$$\mathcal{E}\,[\![e \,\texttt{exists}]\!]\,S = \begin{cases} [\mathit{true}], & \text{if } \operatorname{count}(\mathcal{E}\,[\![e]\!]\,S) \geq 1 \\ [\mathit{false}], & \text{otherwise} \end{cases} \qquad \text{E-EXISTS}$$

$$\mathcal{E}\,[\![e \,\texttt{single exists}]\!]\,S = \begin{cases} [\mathit{true}], & \text{if } \operatorname{count}(\mathcal{E}\,[\![e]\!]\,S) = 1 \\ [\mathit{false}], & \text{otherwise} \end{cases} \qquad \text{E-SINGLEEXISTS}$$

$$\mathcal{E}\,[\![e \,\texttt{multiple exists}]\!]\,S = \begin{cases} [\mathit{true}], & \text{if } \operatorname{count}(\mathcal{E}\,[\![e]\!]\,S) \geq 2 \\ [\mathit{false}], & \text{otherwise} \end{cases} \qquad \text{E-MULTIPLEEXISTS}$$

$$\mathcal{E}\,[\![e \,\texttt{->}\, a_i \,\texttt{only exists}]\!]\,S = \begin{aligned}[t] &\textbf{let } [(D, v_1, \ldots, v_n)] = \mathcal{E}\,[\![e]\!]\,S, \\ &\qquad a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n = \operatorname{allattrs}(D) \\ &\textbf{in } \begin{cases} [\mathit{true}], & \text{if } \operatorname{count}(v_i) \geq 1 \ \wedge\ \forall j \in 1..n : \\ & i \neq j \Rightarrow \operatorname{count}(v_j) = 0 \\ [\mathit{false}], & \text{otherwise} \end{cases} \end{aligned} \qquad \text{E-ONLYEXISTS}$$

$$\mathcal{E}\,[\![e \,\texttt{count}]\!]\,S = [\operatorname{count}(\mathcal{E}\,[\![e]\!]\,S)] \qquad \text{E-COUNT}$$

$$\mathcal{E}\,[\![e \,\texttt{only-element}]\!]\,S = \begin{aligned}[t] &\textbf{let } v = \mathcal{E}\,[\![e]\!]\,S \\ &\textbf{in } \begin{cases} v, & \text{if } \operatorname{count}(v) = 1 \\ [\,], & \text{otherwise} \end{cases} \end{aligned} \qquad \text{E-ONLYELEMENT}$$

$$\mathcal{E}\,[\![e_1 \,\texttt{=}\, e_2]\!]\,S = \operatorname{equals}^*(\mathcal{E}\,[\![e_1]\!]\,S, \mathcal{E}\,[\![e_2]\!]\,S) \qquad \text{E-EQUALS}$$

$$\mathcal{E}\,[\![e_1 \,\texttt{<>}\, e_2]\!]\,S = \begin{aligned}[t] &\textbf{let } [x_1, \ldots, x_m] = \mathcal{E}\,[\![e_1]\!]\,S, \\ &\qquad [y_1, \ldots, y_n] = \mathcal{E}\,[\![e_2]\!]\,S \\ &\textbf{in } \begin{cases} [\mathit{true}], & \text{if } m \neq n \ \vee \\ & \forall i \in 1..n : \neg \operatorname{equals}_(\, x_i, y_i) \\ [\mathit{false}], & \text{otherwise} \end{cases} \end{aligned} \qquad \text{E-NOTEQUALS}$$

$$\mathcal{E}\,[\![e_1\,\texttt{all = }e_2]\!]\,S = \textbf{let}\ \begin{array}[t]{l}[x_1,\dots,x_n] = \mathcal{E}\,[\![e_1]\!]\,S,\\ [y] = \mathcal{E}\,[\![e_2]\!]\,S\end{array}\quad \text{E-AllEquals}$$
$$\textbf{in}\ \begin{cases}[\mathit{true}], & \text{if } \forall i \in 1..n : \text{equals}_(\,x_i, y)\\ [\mathit{false}], & \text{otherwise}\end{cases}$$

$$\mathcal{E}\,[\![e_1\,\texttt{all <> }e_2]\!]\,S = \textbf{let}\ \begin{array}[t]{l}[x_1,\dots,x_n] = \mathcal{E}\,[\![e_1]\!]\,S,\\ [y] = \mathcal{E}\,[\![e_2]\!]\,S\end{array}\quad \text{E-AllNotEquals}$$
$$\textbf{in}\ \begin{cases}[\mathit{true}], & \text{if } \forall i \in 1..n : \neg\,\text{equals}(x_i, y)\\ [\mathit{false}], & \text{otherwise}\end{cases}$$

$$\mathcal{E}\,[\![e_1\,\texttt{any = }e_2]\!]\,S = \textbf{let}\ \begin{array}[t]{l}[x_1,\dots,x_n] = \mathcal{E}\,[\![e_1]\!]\,S,\\ [y] = \mathcal{E}\,[\![e_2]\!]\,S\end{array}\quad \text{E-AnyEquals}$$
$$\textbf{in}\ \begin{cases}[\mathit{true}], & \text{if } \exists i \in 1..n : \text{equals}(x_i, y)\\ [\mathit{false}], & \text{otherwise}\end{cases}$$

$$\mathcal{E}\,[\![e_1\,\texttt{any <> }e_2]\!]\,S = \textbf{let}\ \begin{array}[t]{l}[x_1,\dots,x_n] = \mathcal{E}\,[\![e_1]\!]\,S,\\ [y] = \mathcal{E}\,[\![e_2]\!]\,S\end{array}\quad \text{E-AnyNotEquals}$$
$$\textbf{in}\ \begin{cases}[\mathit{true}], & \text{if } \exists i \in 1..n : \neg\,\text{equals}(x_i, y)\\ [\mathit{false}], & \text{otherwise}\end{cases}$$

$$\mathcal{E}\,[\![e_1\,\texttt{contains}\,e_2]\!]\,S = \textbf{let}\ \begin{array}[t]{l}[x_1,\dots,x_m] = \mathcal{E}\,[\![e_1]\!]\,S,\\ [y_1,\dots,y_n] = \mathcal{E}\,[\![e_2]\!]\,S\end{array}\quad \text{E-Contains}$$
$$\textbf{in}\ \begin{cases}[\mathit{true}], & \text{if } \forall j \in 1..n :\\ & \quad \exists i \in 1..m : \text{equals}(x_i, y_j)\\ [\mathit{false}], & \text{otherwise}\end{cases}$$

$$\mathcal{E}\,[\![e_1\,\texttt{disjoint}\,e_2]\!]\,S = \textbf{let}\ \begin{array}[t]{l}[x_1,\dots,x_m] = \mathcal{E}\,[\![e_1]\!]\,S,\\ [y_1,\dots,y_n] = \mathcal{E}\,[\![e_2]\!]\,S\end{array}\quad \text{E-Disjoint}$$
$$\textbf{in}\ \begin{cases}[\mathit{true}], & \text{if } \forall i \in 1..m :\\ & \quad \forall j \in 1..n : \neg\,\text{equals}(x_i, y_j)\\ [\mathit{false}], & \text{otherwise}\end{cases}$$

$$\mathcal{E}\,[\![[e_1,\dots,e_n]]\!]\,S = \text{flatten}_n(\mathcal{E}\,[\![e_1]\!]\,S,\dots,\mathcal{E}\,[\![e_n]\!]\,S)\quad \text{E-List}$$

$$\mathcal{E}\,[\![\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3]\!]\,S = \textbf{let}\ [x] = \mathcal{E}\,[\![e_1]\!]\,S\quad \text{E-If}$$
$$\textbf{in}\ \begin{cases}\mathcal{E}\,[\![e_1]\!]\,S, & \text{if } x = \mathit{true}\\ \mathcal{E}\,[\![e_2]\!]\,S, & \text{otherwise}\end{cases}$$

$$\mathcal{E}\,[\![F(e_1,\dots,e_n)]\!]\,S = \textbf{let}\ a_1\ T_1\ C_1,\dots,a_n\ T_n\ C_n = \text{inputs}(F)\quad \text{E-Func}$$
$$\textbf{in}\ \mathcal{E}\,[\![\text{op}(F)]\!]\,[\,a_1 \mapsto \mathcal{E}\,[\![e_1]\!]\,S,\dots,\ a_n \mapsto \mathcal{E}\,[\![e_n]\!]\,S\,]$$

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.

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

Subtyping $S <: T$.

$$\frac{}{T <: T} \quad \textsc{S-Refl}$$

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

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

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

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

$$\frac{T_1 <: T_2 \qquad C_1 \subseteq C_2}{T_1\ C_1 <:^* T_2\ C_2} \quad \textsc{S-Card}$$

Typing rules $\Gamma \vdash e : T\ C$. Subtyping $S <: T$.

$$\frac{}{T <: T} \quad \textsc{S-Refl}$$

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

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

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

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

$$\frac{T_1 <: T_2 \qquad C_1 \subseteq C_2}{T_1\ C_1 <:^* T_2\ C_2} \quad \textsc{S-List}$$

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

$$\frac{\Gamma \vdash e : T_1\ C_1 \qquad T_1\ C_1 <:^* T_2\ C_2}{\Gamma \vdash e : T_2\ C_2} \quad \textsc{T-Sub}$$

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash e : \texttt{boolean}\ (1..1)}{\Gamma \vdash \texttt{not}\, e : \texttt{boolean}\ (1..1)}\ \textsc{T-Not}$$

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

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash e : D\ C \qquad \mathrm{allattrs}(D) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n}{\Gamma \vdash e \,\texttt{->}\, a_k : T_k\ C * C_k}\ \textsc{T-Project}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Typing function declarations F OK.

$$\frac{\text{inputs}(F) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n \\ \text{output}(F) = a\ T\ C \qquad a_1 : T_1\ C_1, \ldots, a_n : T_n\ C_n \vdash \text{op}(F) : T\ C}{F \text{ OK}}$$

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 $\text{join}(T_1, T_2)$.

$$\frac{}{\text{join}(T, T) = T}$$

$$\frac{}{\text{join}(T_1, T_2) = \text{join}(T_2, T_1)}$$

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

$$\frac{E \in \text{ancestors}(D)}{\text{join}(D, E) = E}$$

$$\frac{E \notin \text{ancestors}(D) \qquad \text{ET}(E) = \texttt{type}\, E\, \texttt{extends}\, E' : \ldots \qquad T = \text{join}(D, E')}{\text{join}(D, E) = T}$$

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

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

$$\frac{}{\text{join}() = \texttt{nothing}}$$

$$\frac{}{\text{join}(T) = T}$$

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

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

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

Subtyping $S <: T$.

$$\frac{}{T <: T} \quad \text{SA-Refl}$$

$$\frac{}{\texttt{int} <: \texttt{number}} \quad \text{SA-NUM}$$

$$\frac{E \in \text{ancestors}(D)}{D <: E} \quad \text{SA-ANCESTOR}$$

$$\frac{}{\texttt{nothing} <: T} \quad \text{SA-NOTHING}$$

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

$$\frac{T_1 <: T_2 \qquad C_1 \subseteq C_2}{T_1\ C_1 <:^* T_2\ C_2} \quad \text{SA-LIST}$$

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

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

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

$$\frac{}{\Gamma \vdash r : \texttt{number}\ \ (1..1)} \quad \text{TA-NUMBER}$$

$$\frac{}{\Gamma \vdash i : \texttt{int}\ \ (1..1)} \quad \text{TA-INT}$$

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

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

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

$$\frac{\Gamma \vdash e : \texttt{boolean}\ \ (1..1)}{\Gamma \vdash \texttt{not}\, e : \texttt{boolean}\ \ (1..1)} \quad \text{TA-NOT}$$

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

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

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

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

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

$$\frac{\Gamma \vdash e_1 : T_1\ (1..1) \qquad \Gamma \vdash e_2 : T_2\ (1..1) \qquad T_1 <: \texttt{number} \qquad T_2 <: \texttt{number} \qquad T_1 = \texttt{number} \lor T_2 = \texttt{number}}{\Gamma \vdash e_1 \,\texttt{-}\, e_2 : \texttt{number}\ (1..1)} \quad \text{TA-SubtNumber}$$

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

$$\frac{\text{allattrs}(D) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n \qquad \forall i \in 1..n : \Gamma \vdash e_i : T_i'\ C_i' \qquad \forall i \in 1..n : T_i'\ C_i' <:^* T_i\ C_i}{\Gamma \vdash D\,\{\, a_1 : e_1, \ldots, a_n : e_n \,\} : D\ (1..1)} \quad \text{TA-Instantiate}$$

$$\frac{\Gamma \vdash e : D\ C \qquad \text{allattrs}(D) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n}{\Gamma \vdash e \,\texttt{->}\, a_k : T_k\ C * C_k} \quad \text{TA-Project}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\forall i \in 1..n : \Gamma \vdash e_i : T_i\ C_i \qquad T = \text{join}(T_1, \ldots, T_n)}{\Gamma \vdash [e_1, \ldots, e_n] : T\ \sum_{i \in 1..n} C_i} \quad \text{TA-LIST}$$

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

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

Typing function declarations F OK.

$$\frac{\begin{array}{c}\text{inputs}(F) = a_1\ T_1\ C_1, \ldots, a_n\ T_n\ C_n \qquad \text{output}(F) = a\ T\ C \\ a_1 : T_1\ C_1, \ldots, a_n : T_n\ C_n \vdash \text{op}(F) : T'\ C' \qquad T'\ C' <: T\ C\end{array}}{F\ \text{OK}}$$

5 Code Generation

Java representation of list types $\mathcal{T}^*_{\mathcal{J}} [\![T\ C]\!]$.

$$\mathcal{T}^*_{\mathcal{J}} [\![T\ (0..1)]\!] = \mathcal{T}_{\mathcal{J}} [\![T]\!] \quad \text{J-OPTIONAL}$$

$$\mathcal{T}^*_{\mathcal{J}} [\![T\ (1..1)]\!] = \begin{cases} \texttt{boolean}, & \text{if } T = \texttt{boolean} \\ \texttt{int}, & \text{if } T = \texttt{int} \\ \mathcal{T}_{\mathcal{J}} [\![T]\!], & \text{otherwise} \end{cases} \quad \text{J-SINGULAR}$$

$$\mathcal{T}^*_{\mathcal{J}} [\![T\ (k..l)]\!] = \mathsf{List}{<}?\ \texttt{extends}\ \mathcal{T}_{\mathcal{J}} [\![T]\!]{>}, \quad \text{if } l > 1 \quad \text{J-PLURAL}$$

$$\mathcal{T}^*_{\mathcal{J}} [\![T\ (0..0)]\!] = \mathcal{T}_{\mathcal{J}} [\![T]\!] \quad \text{J-EMPTY}$$

Java reference type of each item type $\mathcal{T}_{\mathcal{J}} [\![T]\!]$.

$$\begin{aligned} \mathcal{T}_{\mathcal{J}} [\![\texttt{boolean}]\!] &= \mathsf{Boolean} \\ \mathcal{T}_{\mathcal{J}} [\![\texttt{int}]\!] &= \mathsf{Integer} \\ \mathcal{T}_{\mathcal{J}} [\![\texttt{number}]\!] &= \mathsf{NougaNumber} \\ \mathcal{T}_{\mathcal{J}} [\![\texttt{nothing}]\!] &= \mathsf{Void} \\ \mathcal{T}_{\mathcal{J}} [\![D]\!] &= D \end{aligned}$$