Type Checking

- Traversing of abstract tree from the root

- Checks:

1. Equality of function's name with names specified before and after the body
2. Definition of constants in **const** section: compatibility of instance with type
3. Visibility of referenced identifiers
4. Compatibility in number and type of actual parameters with formal parameters
5. Compatibility of return expressions with function codomain
6. Return statement always executed for any function call
7. Compatibility of operators with operands
8. Compatibility of LHS of assignment with RHS of assignment (expression)
9. Compatibility of expressions with statements in which they are involved

- Type inference: computation of result schema of each operation

$\boldsymbol{logic\text{-}expr \rightarrow expr_1\ expr_2}$

- Qualifier: `AND`, `OR`

- Constraint: type($expr_1$) = **`bool`** **and** type($expr_2$) = **`bool`**

- Type inference: type($logic\text{-}expr$) = **`bool`**

$rel\text{-}expr \rightarrow\ expr_1\ expr_2$

- Qualifier: `EQUAL`, `NEQ`

- Constraint: type($expr_1$) = type($expr_2$)

- Type inference: type($rel\text{-}expr$) = **`bool`**

$\textit{rel-expr} \rightarrow \; \textit{expr}_1 \; \textit{expr}_2$

- Qualifier: `'>'`, `GEQ`, `'<'`, `LEQ`

- Constraint: type($expr_1$) = type($expr_2$) **`and`** type($expr_1$) $\in$ { **`int`**, **`real`**, **`string`** }

- Type inference: type($\textit{rel-expr}$) = **`bool`**

$rel\text{-}expr \rightarrow \; expr_1 \; expr_2$

- Qualifier: `IN`

- Constraint: type($expr_2$) = **`vector`** `[…]` **`of`** type($expr_1$)

- Type inference: type($rel\text{-}expr$) = **`bool`**

$\textit{math-expr} \rightarrow \textit{expr}_1\ \textit{expr}_2$

- Constraint: (type($expr_1$) = **`int`** **and** type($expr_2$) = **`int`**) **or**
  (type($expr_1$) = **`real`** **and** type($expr_2$) = **`real`**)

- Type inference: type($\textit{math-expr}$) = type($expr_1$)

neg-expr → *expr*

- Qualifier: `'-'`

- Constraint: math(*expr*)

- Type inference: type(*neg-expr*) = type(*expr*)

neg-expr → *expr*

- Qualifier: `NOT`
- Constraint: type(*expr*) = **`bool`**
- Type inference: type(*neg-expr*) = **`bool`**

wr-expr → *specifier-opt expr*

- Constraint: null(*specifier-opt*) or type(*specifier-opt*) = **`string`**

- Type inference: type(*wr-expr*) = type(*expr*)

rd-expr $\rightarrow$ *specifier-opt domain*

- Constraint: null(*specifier-opt*) or type(*specifier-opt*) = **`string`**

- Type inference: type(*rd-expr*) = type(*domain*)

left-hand-side → **id**

- Constraint: visible(name(**id**)) **and** `class`(name(**id**)) ∈ { `VAR`, `PAR` })

- Type inference = type(*left-hand-side*) = `schema`(name(**id**))

fielding → *left-hand-side* **id**

- Constraint: type(*left-hand-side*) = **`struct`** **`and`**
  (name(**id**): domain) ∈ attr(type(*left-hand-side*))

- Type inference = type(*fielding*) = domain

indexing → *left-hand-side expr*

- Constraint: type(*left-hand-side*) = **`vector [...] of`** domain,
  type(*expr*) = **`int`**

- Type inference = type(*indexing*) = domain

$\textit{instance-expr} \rightarrow \textit{expr}_1 \ \textit{expr}_2 \ \ldots \ \textit{expr}_n$

- Qualifier: `STRUCT`

- Type inference:

type(*instance-expr*) = **`struct`**(**nil**: type($expr_1$), …, **nil**: type($expr_n$))

$\boldsymbol{instance\text{-}expr \rightarrow expr_1 \; expr_2 \; \ldots \; expr_n}$

- Qualifier: `VECTOR`
- Constraint: $\forall i \in [1..\, n], \forall j \in [1..\, n]$ (type(expr$_i$) $\approx$ type(expr$_j$))
- Type inference: type(*instance-expr*) = **`vector`** `[` n `]` **`of`** type($expr_1$)

$\textbf{\textit{func-call}} \rightarrow \textbf{id} \ \textbf{\textit{expr}}_1 \ \textbf{\textit{expr}}_2 \ \ldots \ \textbf{\textit{expr}}_n$

- Constraint:

  visible(name(**id**)) **and** class(name(**id**)) = `FUNC` **and**
  n = number of formal parameters of name(**id**) **and**
  $\forall i \in [1..\ n]$ (type(expr$_i$) compatible with i-th formal parameter of name(**id**))

- Type inference: type(*func-call*) = `schema`(name(**id**))

$\textit{cond-expr} \rightarrow \textit{expr}_1\ \textit{expr}_2\ \textit{elsif-expr-list-opt}\ \textit{expr}_3$

- Constraint: type($expr_1$) = **`bool`** **and** type($expr_2$) $\approx$ type($expr_3$) **and** (null($\textit{elsif-expr-list-opt}$)) **or** type($\textit{elsif-expr-list-opt}$) $\approx$ type($expr_2$))

- Type inference: type($\textit{cond-expr}$) = type($expr_2$)

$\textit{elsif-expr-list-opt} \rightarrow \textit{expr}_{11}\ \textit{expr}_{12}\ \ \textit{expr}_{21}\ \textit{expr}_{22}\ \ \ldots\ \ \textit{expr}_{n1}\ \textit{expr}_{n2}$

- Constraint: $\forall\, i \in [1..n]$ (type($expr_{i1}$) = `bool`) **and**
  $\forall\, i \in [1..n], \forall\, j \in [1..n]$ (type($expr_{i2}$) $\approx$ type($expr_{j2}$))

- Type inference: type(*elsif-expr-list-opt*) =

  if n = 0 **then**
      nil
  else
      type($expr_{12}$)
  endif

built-in-call → *expr*

- Qualifier: `TOINT`

- Constraint: type(*expr*) = **`real`**

- Type inference = type(*built-in-call*) = **`int`**

built-in-call → *expr*

- Qualifier: `TOREAL`

- Constraint: type(*expr*) = **`int`**

- Type inference = type(*built-in-call*) = **`real`**

assign-stat $\rightarrow$ *left-hand-side expr*

- Constraint: type(*left-hand-side*) ≈ type(*expr*)

if-stat $\rightarrow$ *expr* *stat-list*$_1$ *elsif-stat-list-opt* [*stat-list*$_2$]

- Constraint: type(*expr*) = **`bool`**

$\boldsymbol{elsif\text{-}stat\text{-}list\text{-}opt \rightarrow\ expr_{11}\ stat\text{-}list_{12}\ \ldots\ expr_{n1}\ stat\text{-}list_{n2}}$

- Constraint: $\forall\, i \in [1..n]\ (\text{type}(expr_{i1}) = \texttt{bool})$

while-stat → *expr stat-list*

- Constraint: type(*expr*) = `bool`

$\textbf{\textit{for-stat}} \rightarrow \textbf{id} \; \textbf{\textit{expr}}_1 \; \textbf{\textit{expr}}_2 \; \textbf{\textit{stat-list}}$

- Constraint: visible(name(**id**)) **and**
  class(name(**id**)) $\in$ { `VAR`, `PAR` } **and**
  `schema`(name(**id**)) `.type` = `INT` **and**
  type($expr_1$) = **`int`** **and**
  type($expr_2$) = **`int`**

foreach-stat → **id** *expr* *stat-list*

- Constraint: visible(name(**id**)) **and**
  class(name(**id**)) ∈ { `VAR`, `PAR` } **and**
  `schema`(*expr*) = **`vector`** `[`...`]` **`of`** (`schema`(name(**id**)) `.type`)

read-stat $\rightarrow$ *specifier-opt* **id**

- Constraint: visible(name(**id**)), class(name(**id**)) $\in$ { `VAR`, `PAR` } **and** (null(*specifier-opt*) **or** type(*specifier-opt*) = **`string`**)

write-stat → *specifier-opt expr*

- Constraint: null(*specifier-opt*) **or** type(*specifier-opt*) = **`string`**