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

logic-expr → *expr*$_1$ *expr*$_2$

- Qualifier: `AND`, `OR`
- Constraint: type(*expr*$_1$) = **`bool`** **and** type(*expr*$_2$) = **`bool`**
- Type inference: type(*logic-expr*) = **`bool`**

rel-expr → *expr*$_1$ *expr*$_2$

- Qualifier: `EQUAL`, `NEQ`
- Constraint: type(*expr*$_1$) = type(*expr*$_2$)
- Type inference: type(*rel-expr*) = **`bool`**

rel-expr → *expr*$_1$ *expr*$_2$

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

- Constraint: type(*expr*$_1$) = type(*expr*$_2$) **and**
  type(*expr*$_1$) ∈ { **`char`**, **`int`**, **`real`**, **`string`** }

- Type inference: type(*rel-expr*) = **`bool`**

rel-expr → *expr*$_1$ *expr*$_2$

- Qualifier: `IN`
- Constraint: type(*expr*$_2$) = **`vector [...] of`** type(*expr*$_1$)
- Type inference: type(*rel-expr*) = **`bool`**

math-expr → *expr*$_1$ *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(*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 → 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`, `CONST` })

- 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

instance-expr → *expr*$_1$ *expr*$_2$ … *expr*$_n$

- Qualifier: `STRUCT`

- Type inference:

type(*instance-expr*) = **`struct`**(**nil**: type(*expr*$_1$), …, **nil**: type(*expr*$_n$))

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

- 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($expr_1$)

func-call → **id** $expr_1$ $expr_2$ … $expr_n$

- 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**))

cond-expr → *expr*$_1$ *expr*$_2$ *elsif-expr-list-opt* *expr*$_3$

- Constraint: type(*expr*$_1$) = **`bool`** **and** type(*expr*$_2$) ≈ type(*expr*$_3$) **and**
  (null(*elsif-expr-list-opt*)) **or** type(*elsif-expr-list-opt*) ≈ type(*expr*$_2$))

- Type inference: type(*cond-expr*) = type(*expr*$_2$)

***elsif-expr-list-opt* → *expr*$_{11}$ *expr*$_{12}$ *expr*$_{21}$ *expr*$_{22}$ … *expr*$_{n1}$ *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(expr12)
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 → *left-hand-side expr*

- Constraint: type(*left-hand-side*) ≈ type(*expr*) **and**
  left-hand-side refers either to `VAR` or `PAR`

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

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

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

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

while-stat → expr stat-list

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

for-stat → **id** *expr*$_1$ *expr*$_2$ *stat-list*

- Constraint: visible(name(**id**)) **and**
  class(name(**id**)) ∈ { `VAR`, `PAR` } **and**
  `schema`(name(**id**))`.type` = `INT` **and**
  type(*expr*$_1$) = **`int`** **and**
  type(*expr*$_2$) = **`int`** **and**
  name(**id**) is not assigned in *stat-list*

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 → *specifier-opt* **id**

- Constraint: visible(name(**id**)), class(name(**id**)) ∈ { `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`**