The LANG'24 Language Specification

1 Lexical structure

Programs in the LANG'24 programming language are written in ASCII character set, e.g., no additional characters denoting post-alveolar consonants are allowed.

Programs in the LANG'24 programming language consist of the following lexical elements:

- *Literals*:
  - *numerical literals*:
    A nonempty finite string of decimal digits (`0`...`9`) optionally preceded by a sign (`+` or `-`).
  - *character literals*:
    A character enclosed in single quotes (`'`). A character in a string literal can be specified by (a) any printable ASCII character, i.e., with ASCII code in range $\{32...126\}$ but the single quote and the backslash must be preceded by the backslash (\\), (b) a control sequence `\n` denoting the end of line or (c) an ASCII code represented as `\XX` where `X` stands for any uppercase hexadecimal digit (`0`...`9` or `A`...`F`).
  - *string literals*:
    A possibly empty string of characters enclosed in double quotes (`"`). A character in a string literal can be specified by (a) any printable ASCII character, i.e., with ASCII code in range $\{32...126\}$ but the double quote and the backslash must be preceded by the backslash (\\), (b) a control sequence `\n` denoting the end of line or (c) an ASCII code represented as `\XX` where `X` stands for any uppercase hexadecimal digit (`0`...`9` or `A`...`F`).
- *Symbols*:
  `( ) { } [ ] . , : ; == != < > <= >= * / % + - ^ =`
- *Identifiers*:
  A nonempty finite string of letters (`A`...`Z` and `a`...`z`), decimal digits (`0`...`9`), and underscores (`_`) that (a) starts with either a letter or an underscore and (b) is not a keyword or a constant.
- *Built-in data types:*
- *Keywords*:
  - literals: `true false nil none`
  - built-in data types: `bool char int void`
  - operators: `and not or sizeof`
  - statements: `return if then else while`
- *Comments*:
  A string of characters starting with a hash (`#`) and extending to the end of line.
- *White space*:
  Space, horizontal tab (HT), line feed (LF) and carriage return (CR). Line feed alone denotes the end of line within a source file. Horizontal tabs are considered to be 8 spaces wide.

Lexical elements are recognized from left to right using the longest match approach.

2 Syntax structure

The concrete syntax of the LANG'24 programming language is defined by a context free grammar with the start symbol *definitions* and the following productions:

definitions
$\longrightarrow$ $\big($ *type-definition* | *variable-definition* | *function-definition* $\big)^{+}$

type-definition
$\longrightarrow$ identifier `=` *type*

variable-definition
$\longrightarrow$ identifier `:` *type*

function-definition
$\longrightarrow$ identifier `(` $\big($ *parameters* $\big)^{?}$ `)` `:` *type* $\big($ `=` *statement* $\big($ `{` *definitions* `}` $\big)^{?}$ $\big)^{?}$

parameters
$\longrightarrow$ $\big($ `^` $\big)^{?}$ identifier `:` *type* $\big($ `,` $\big($ `^` $\big)^{?}$ identifier `:` *type* $\big)^{*}$

statement
$\longrightarrow$ *expression* `;`
$\longrightarrow$ *expression* `=` *expression* `;`
$\longrightarrow$ `if` *expression* `then` *statement* $\big($ `else` *ststement* $\big)^{?}$
$\longrightarrow$ `while` *expression* `:` *statement*
$\longrightarrow$ `return` *expression* `;`
$\longrightarrow$ `{` $\big($ *statement* $\big)^{+}$ `}`

type
$\longrightarrow$ `void` | `bool` | `char` | `int`
$\longrightarrow$ `[` intconst `]` *type*
$\longrightarrow$ `^` *type*
$\longrightarrow$ `(` *components* `)`
$\longrightarrow$ `{` *components* `}`
$\longrightarrow$ identifier

components
$\longrightarrow$ identifier `:` *type* $\big($ `,` identifier `:` *type* $\big)^{*}$

expression
$\longrightarrow$ voidconst | boolconst | charconst | intconst | strconst | ptrconst
$\longrightarrow$ identifier $\big($ `(` $\big($ *expression* $\big($ `,` *expression* $\big)^{*}$ $\big)^{?}$ `)` $\big)^{?}$
$\longrightarrow$ prefix-operator *expression*
$\longrightarrow$ *expression* binary-operator *expression*
$\longrightarrow$ `<` *type* `>` *expression*
$\longrightarrow$ *expression* `[` *expression* `]`
$\longrightarrow$ *expression* `.` identifier
$\longrightarrow$ *expression* `^`
$\longrightarrow$ `sizeof` `(` *type* `)`
$\longrightarrow$ `(` *expression* `)`

Symbols voidconst, boolconst, charconst, intconst, strconst, and ptrconst denote void constant `none`, boolean constants `true` and `false`, character literals, integer literals, string literals, and pointer constant `nil`, respectively.

The precedence of the operators is as follows:

postfix operators	`[·] ^ .`	THE HIGHEST PRECEDENCE
prefix operators	`not + - ^ <·>`	
multiplicative operators	`* / %`	
additive operators	`+ -`	
relational operators	`== != < > <= >=`	
conjunctive operator	`and`	
disjunctive operator	`or`	THE LOWEST PRECEDENCE

Binary operators are left associative.

The `else` part of the conditional statement binds to the nearest preceding `if` part.

3 Semantic structure

3.1 Name binding

Let function $[\![\cdot]\!]_{\mathrm{BIND}}$ bind a name to its declaration according to the rules of namespaces and scopes described below. Hence, the value of function $[\![\cdot]\!]_{\mathrm{BIND}}$ depends on the context of its argument.

Namespaces. There are two kinds of a namespaces:

1. Names of types, functions, variables and parameters belong to one single global namespace.
2. Names of components belong to structure- or union-specific namespaces, i.e., each structure or union defines its own namespace containing names of its components.

Scopes. Two new scopes are created in every function definition

identifier `(` *parameters* `)` `:` *type* `=` *statement* `{` *definitions* `}`

as follows:

1. The name, the parameter types and the result type belong to the scope in which the function is defined.
2. The parameter names belong to the scope nested within the scope in which the function is defined.
3. Statements and definitions belong to the scope nested within the scope in which parameter names are defined.

If there are no parameters, statements or definitions, the scopes are created nevertheless.

All names declared within a given scope are visible in the entire scope unless hidden by a definition in the nested inner scope. A name can be declared within the same scope at most once.

3.2 Type system

The set

$$
\begin{aligned}
\mathcal{T}_d = \{&\mathbf{void}, \mathbf{char}, \mathbf{int}, \mathbf{bool}\} && \text{(atomic types)}\\
\cup\, \{&\mathbf{arr}(n \times \tau) \mid n > 0 \wedge \tau \in \mathcal{T}_d\} && \text{(arrays)}\\
\cup\, \{&\mathbf{struct}_{id_1,\ldots,id_n}(\tau_1, \ldots, \tau_n) \mid n > 0 \wedge \tau_1, \ldots, \tau_n \in \mathcal{T}_d\} && \text{(structs)}\\
\cup\, \{&\mathbf{union}_{id_1,\ldots,id_n}(\tau_1, \ldots, \tau_n) \mid n > 0 \wedge \tau_1, \ldots, \tau_n \in \mathcal{T}_d\} && \text{(unions)}\\
\cup\, \{&\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} && \text{(pointers)}
\end{aligned}
$$

denotes the set of all data types of LANG'24. The set

$$
\begin{aligned}
\mathcal{T} = {}& \mathcal{T}_d && \text{(data types)}\\
&\cup \{(\tau_1, \ldots, \tau_n) \to \tau \mid n \geq 0 \wedge \tau_1, \ldots, \tau_n, \tau \in \mathcal{T}_d\} && \text{(functions)}
\end{aligned}
$$

denotes the set of all types of LANG'24.

Structural equivalence of types: Types τ_1 and τ_2 are equivalent if (a) $\tau_1 = \tau_2$ or (b) if they are type synonyms (introduced by chains of type declarations) of types τ_1' and τ_2' where $\tau_1' = \tau_2'$.

Semantic functions

$$[\![\cdot]\!]_{\mathrm{ISTYPE}} \colon \mathcal{P} \to \mathcal{T} \quad \text{and} \quad [\![\cdot]\!]_{\mathrm{OFTYPE}} \colon \mathcal{P} \to \mathcal{T}$$

map syntactic phrases of LANG'24 to types. Function $[\![\cdot]\!]_{\mathrm{ISTYPE}}$ denotes the type described by a phrase, function $[\![\cdot]\!]_{\mathrm{OFTYPE}}$ denotes the type of a value described by a phrase.

The following assumptions are made in the rules below:

- Function val maps lexemes to data of the specified type.
- $\tau \in \mathcal{T}_d$ unless specified otherwise.

Type expressions.

$$\frac{}{[\![\texttt{void}]\!]_{\mathrm{ISTYPE}} = \mathbf{void}} \quad \frac{}{[\![\texttt{bool}]\!]_{\mathrm{ISTYPE}} = \mathbf{bool}} \quad \frac{}{[\![\texttt{char}]\!]_{\mathrm{ISTYPE}} = \mathbf{char}} \quad \frac{}{[\![\texttt{int}]\!]_{\mathrm{ISTYPE}} = \mathbf{int}} \tag{T1}$$

$$\frac{[\![\mathit{type}]\!]_{\mathrm{ISTYPE}} = \tau \quad \mathrm{val}(\mathit{int}) = n \\ 0 < n \leq 2^{63} - 1 \quad \tau \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{[\![\texttt{[}\mathit{int}\texttt{]}\,\mathit{type}]\!]_{\mathrm{ISTYPE}} = \mathbf{arr}(n \times \tau)} \tag{T2}$$

$$\frac{n > 0 \quad \forall i \in \{1 \ldots n\} \colon [\![\mathit{type}_i]\!]_{\mathrm{ISTYPE}} = \tau_i \quad \tau_i \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{[\![\texttt{(}\mathit{id}_1 \texttt{:} \mathit{type}_1, \ldots, \mathit{id}_n \texttt{:} \mathit{type}_n\texttt{)}]\!]_{\mathrm{ISTYPE}} = \mathbf{struct}_{id_1,\ldots,id_n}(\tau_1, \ldots, \tau_n)} \tag{T3}$$

$$\frac{n > 0 \quad \forall i \in \{1 \ldots n\} \colon [\![\mathit{type}_i]\!]_{\mathrm{ISTYPE}} = \tau_i \quad \tau_i \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{[\![\texttt{\{}\mathit{id}_1 \texttt{:} \mathit{type}_1, \ldots, \mathit{id}_n \texttt{:} \mathit{type}_n\texttt{\}}]\!]_{\mathrm{ISTYPE}} = \mathbf{union}_{id_1,\ldots,id_n}(\tau_1, \ldots, \tau_n)} \tag{T4}$$

$$\frac{[\![\mathit{type}]\!]_{\mathrm{ISTYPE}} = \tau \quad \tau \in \mathcal{T}_d}{[\![\texttt{\^{}}\ \mathit{type}]\!]_{\mathrm{ISTYPE}} = \mathbf{ptr}(\tau)} \tag{T5}$$

Value expressions.

$$\frac{}{[\![\texttt{none}]\!]_{\mathrm{OFTYPE}} = \mathbf{void}} \quad \frac{}{[\![\texttt{nil}]\!]_{\mathrm{OFTYPE}} = \mathbf{ptr}(\mathbf{void})} \quad \frac{}{[\![\mathit{string}]\!]_{\mathrm{OFTYPE}} = \mathbf{ptr}(\mathbf{char})} \tag{V1}$$

$$\frac{}{[\![\mathit{bool}]\!]_{\mathrm{OFTYPE}} = \mathbf{bool}} \quad \frac{}{[\![\mathit{char}]\!]_{\mathrm{OFTYPE}} = \mathbf{char}} \quad \frac{}{[\![\mathit{int}]\!]_{\mathrm{OFTYPE}} = \mathbf{int}} \tag{V2}$$

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{bool}}{\llbracket \texttt{not}\ \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{bool}} \qquad \frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{int} \quad \mathit{op} \in \{\texttt{+}, \texttt{-}\}}{\llbracket \mathit{op}\ \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{int}} \tag{v3}$$

$$\frac{\llbracket \mathit{expr}_1 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{bool} \quad \llbracket \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{bool} \quad \mathit{op} \in \{\texttt{and}, \texttt{or}\}}{\llbracket \mathit{expr}_1\ \mathit{op}\ \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{bool}} \tag{v4}$$

$$\frac{\llbracket \mathit{expr}_1 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{int} \quad \llbracket \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{int} \quad \mathit{op} \in \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{\%}\}}{\llbracket \mathit{expr}_1\ \mathit{op}\ \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{int}} \tag{v5}$$

$$\frac{\begin{array}{c}\llbracket \mathit{expr}_1 \rrbracket_{\mathrm{OFTYPE}} = \tau \quad \llbracket \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \tau \\ \tau \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \quad \mathit{op} \in \{\texttt{==}, \texttt{!=}\}\end{array}}{\llbracket \mathit{expr}_1\ \mathit{op}\ \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{bool}} \tag{v6}$$

$$\frac{\begin{array}{c}\llbracket \mathit{expr}_1 \rrbracket_{\mathrm{OFTYPE}} = \tau \quad \llbracket \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \tau \\ \tau \in \{\mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \quad \mathit{op} \in \{\texttt{<=}, \texttt{>=}, \texttt{<}, \texttt{>}\}\end{array}}{\llbracket \mathit{expr}_1\ \mathit{op}\ \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{bool}} \tag{v7}$$

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \tau \quad \llbracket \mathit{expr} \rrbracket_{\mathrm{ISLVAL}} = \texttt{true}}{\llbracket \texttt{\^{}}\ \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{ptr}(\tau)} \qquad \frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{ptr}(\tau)}{\llbracket \mathit{expr}\ \texttt{\^{}} \rrbracket_{\mathrm{OFTYPE}} = \tau} \tag{v8}$$

$$\frac{\llbracket \mathit{expr}_1 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{arr}(n \times \tau) \quad \llbracket \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \mathbf{int} \quad \llbracket \mathit{expr}_1 \rrbracket_{\mathrm{ISLVAL}} = \texttt{true}}{\llbracket \mathit{expr}_1\,\texttt{[}\mathit{expr}_2\texttt{]} \rrbracket_{\mathrm{OFTYPE}} = \tau} \tag{v9}$$

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{struct}_{id_1,\ldots,id_n}(\tau_1, \ldots, \tau_n) \quad \mathit{identifier} = id_i}{\llbracket \mathit{expr}\,\texttt{.}\,\mathit{identifier} \rrbracket_{\mathrm{OFTYPE}} = \tau_i} \tag{v10}$$

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{union}_{id_1,\ldots,id_n}(\tau_1, \ldots, \tau_n) \quad \mathit{identifier} = id_i}{\llbracket \mathit{expr}\,\texttt{.}\,\mathit{identifier} \rrbracket_{\mathrm{OFTYPE}} = \tau_i} \tag{v11}$$

$$\frac{\begin{array}{c}\llbracket \mathit{identifier} \rrbracket_{\mathrm{OFTYPE}} = (\tau_1, \ldots, \tau_n) \to \tau \\ \forall i \in \{1 \ldots n\}\colon \llbracket \mathit{expr}_i \rrbracket_{\mathrm{OFTYPE}} = \tau_i \wedge \tau_i \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \tau \in \{\mathbf{void}, \mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \forall i \in \{1 \ldots n\}\colon \text{(the } i\text{-th parameter is a call-by-reference)} \implies \llbracket \mathit{expr}_i \rrbracket_{\mathrm{ISLVAL}} = \texttt{true}\end{array}}{\llbracket \mathit{identifier}\,\texttt{(}\mathit{expr}_1, \ldots, \mathit{expr}_n\texttt{)} \rrbracket_{\mathrm{OFTYPE}} = \tau} \tag{v12}$$

$$\frac{\llbracket \mathit{type} \rrbracket_{\mathrm{ISTYPE}} = \tau_1 \quad \llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \tau_2 \quad \tau_1, \tau_2 \in \{\mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\}}{\llbracket \texttt{<}\mathit{type}\texttt{>}\mathit{expr}\texttt{)} \rrbracket_{\mathrm{OFTYPE}} = \tau_1} \tag{v13}$$

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \tau}{\llbracket \texttt{(}\mathit{expr}\texttt{)} \rrbracket_{\mathrm{OFTYPE}} = \tau} \qquad \frac{\llbracket \mathit{type} \rrbracket_{\mathrm{ISTYPE}} = \tau}{\llbracket \texttt{sizeof(}\mathit{type}\texttt{)} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{int}} \tag{v14}$$

Statements.

$$\frac{\begin{array}{c}\llbracket \mathit{expr}_1 \rrbracket_{\mathrm{OFTYPE}} = \tau \quad \llbracket \mathit{expr}_2 \rrbracket_{\mathrm{OFTYPE}} = \tau \\ \tau \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \llbracket \mathit{expr}_1 \rrbracket_{\mathrm{ISLVAL}} = \texttt{true}\end{array}}{\llbracket \mathit{expr}_1 \,\texttt{=}\, \mathit{expr}_2\ \texttt{;} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{void}} \tag{s1}$$

$$\frac{[\![expr]\!]_{\text{OFTYPE}} = \mathbf{void}}{[\![expr\ \texttt{;}]\!]_{\text{OFTYPE}} = \mathbf{void}} \tag{s2}$$

$$\frac{[\![expr]\!]_{\text{OFTYPE}} = \mathbf{bool} \quad [\![stmts]\!]_{\text{OFTYPE}} = \tau}{[\![\texttt{if}\ expr\ \texttt{then}\ stmts]\!]_{\text{OFTYPE}} = \mathbf{void}} \tag{s3}$$

$$\frac{[\![expr]\!]_{\text{OFTYPE}} = \mathbf{bool} \quad [\![stmts_1]\!]_{\text{OFTYPE}} = \tau_1 \quad [\![stmts_2]\!]_{\text{OFTYPE}} = \tau_2}{[\![\texttt{if}\ expr\ \texttt{then}\ stmts_1\ \texttt{else}\ stmts_2]\!]_{\text{OFTYPE}} = \mathbf{void}} \tag{s4}$$

$$\frac{[\![expr]\!]_{\text{OFTYPE}} = \mathbf{bool} \quad [\![stmts]\!]_{\text{OFTYPE}} = \tau}{[\![\texttt{while}\ expr\ \texttt{:}\ stmts]\!]_{\text{OFTYPE}} = \mathbf{void}} \tag{s5}$$

$$\frac{[\![expr]\!]_{\text{OFTYPE}} = \tau \quad \text{(the result type of the innermost function is } \tau\text{)}}{[\![\texttt{return}\ expr\ \texttt{;}]\!]_{\text{OFTYPE}} = \mathbf{void}} \tag{s6}$$

$$\frac{n > 0 \quad \forall i \in \{1 \dots n\} \colon [\![stmt_i]\!]_{\text{OFTYPE}} = \tau_i}{[\![\texttt{\{}stmt_1 \dots stmt_n\texttt{\}}]\!]_{\text{OFTYPE}} = \mathbf{void}} \tag{s7}$$

Declarations.

$$\frac{[\![identifier]\!]_{\text{BIND}} = identifier\texttt{=}type \quad [\![type]\!]_{\text{ISTYPE}} = \tau}{[\![identifier]\!]_{\text{ISTYPE}} = \tau} \tag{d1}$$

$$\frac{[\![identifier]\!]_{\text{BIND}} = identifier\texttt{:}type \quad [\![type]\!]_{\text{ISTYPE}} = \tau \quad \tau \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{[\![identifier]\!]_{\text{OFTYPE}} = \tau} \tag{d2}$$

$$\frac{\begin{array}{c}[\![identifier]\!]_{\text{BIND}} = identifier\texttt{(}identifer_1\texttt{:}type_1, \dots, identifer_n\texttt{:}type_n\texttt{)}\texttt{:}type \\ \forall i \in \{1 \dots n\} \colon [\![type_i]\!]_{\text{ISTYPE}} = \tau_i \wedge \tau_i \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ [\![type]\!]_{\text{ISTYPE}} = \tau \quad \tau \in \{\mathbf{void}, \mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\}\end{array}}{[\![identifier]\!]_{\text{OFTYPE}} = (\tau_1, \dots, \tau_n) \to \tau} \tag{d3}$$

$$\frac{\begin{array}{c}[\![identifier]\!]_{\text{BIND}} = identifier\texttt{(}identifer_1\texttt{:}type_1, \dots, identifer_n\texttt{:}type_n\texttt{)}\texttt{:}type\,\texttt{=}\,stmt \\ \forall i \in \{1 \dots n\} \colon [\![type_i]\!]_{\text{ISTYPE}} = \tau_i \wedge \tau_i \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ [\![type]\!]_{\text{ISTYPE}} = \tau \quad [\![stmt]\!]_{\text{OFTYPE}} = \mathbf{void} \quad \tau \in \{\mathbf{void}, \mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\}\end{array}}{[\![identifier]\!]_{\text{OFTYPE}} = (\tau_1, \dots, \tau_n) \to \tau} \tag{d4}$$

3.3 Lvalues

The semantic function

$$[\![\cdot]\!]_{\text{ISLVAL}} \colon \mathcal{P} \to \{\mathbf{true}, \mathbf{false}\}$$

denotes which phrases represent lvalues.

$$\frac{[\![identifier]\!]_{\text{BIND}} = \text{variable declaration}}{[\![identifier]\!]_{\text{ISLVAL}} = \mathbf{true}} \qquad \frac{[\![identifier]\!]_{\text{BIND}} = \text{parameter declaration}}{[\![identifier]\!]_{\text{ISLVAL}} = \mathbf{true}}$$

$$\frac{[\![expr]\!]_{\text{OFTYPE}} = \mathbf{ptr}(\tau)}{[\![expr\texttt{\^{}}]\!]_{\text{ISLVAL}} = \mathbf{true}} \quad \frac{[\![expr]\!]_{\text{ISLVAL}} = \mathbf{true}}{[\![expr\,\texttt{[}expr'\texttt{]}]\!]_{\text{ISLVAL}} = \mathbf{true}} \quad \frac{[\![expr]\!]_{\text{ISLVAL}} = \mathbf{true}}{[\![expr\,.\,identifier]\!]_{\text{ISLVAL}} = \mathbf{true}}$$

$$\frac{[\![expr]\!]_{\text{ISLVAL}} = \texttt{true}}{[\![\texttt{<}type\texttt{>}expr]\!]_{\text{ISLVAL}} = \mathbf{true}}$$

In all other cases the value of $[\![\cdot]\!]_{\text{ISLVAL}}$ equals **false**.

3.4 Linkage

A variable or a function has external linkage if it is not declared inside a function.

3.5 Operational semantics

Operational semantics is described by semantic functions

$$\begin{aligned} [\![\cdot]\!]_{\text{ADDR}} &: \mathcal{P} \times \mathcal{M} \to \mathcal{I} \times \mathcal{M} \\ [\![\cdot]\!]_{\text{EXPR}} &: \mathcal{P} \times \mathcal{M} \to \mathcal{I} \times \mathcal{M} \\ [\![\cdot]\!]_{\text{STMT}} &: \mathcal{P} \times \mathcal{M} \to \mathcal{M} \end{aligned}$$

where P denotes the set of phrases of PREV'23, I denotes the set of 64-bit integers, and M denotes possible states of the memory. Unary operators and binary operators perform 64-bit signed operations (except for type **char** where operations are performed on the lower 8 bits only).

Auxilary function addr returns either an absolute address for a static variable or a string constant or an offset for a local variable, parameter or record component. Auxilary function sizeof returns the size of a type. Auxilary function val returns the value of an integer constant or an ASCII code of a char constant.

Addresses.

$$\frac{}{[\![string]\!]^{\text{M}}_{\text{ADDR}} = \langle \text{addr}(string), \text{M} \rangle} \tag{A1}$$

$$\frac{\text{addr}(identifier) = a}{[\![identifier]\!]^{\text{M}}_{\text{ADDR}} = \langle a, \text{M} \rangle} \tag{A2}$$

$$\frac{[\![expr_1]\!]^{\text{M}}_{\text{ADDR}} = \langle n_1, \text{M}' \rangle \quad [\![expr_2]\!]^{\text{M}'}_{\text{EXPR}} = \langle n_2, \text{M}'' \rangle \quad [\![expr_1]\!]_{\text{OFTYPE}} = \mathbf{arr}(n \times \tau)}{[\![expr_1\,\texttt{[}expr_2\texttt{]}]\!]^{\text{M}}_{\text{ADDR}} = \langle n_1 + n_2 * \text{sizeof}(\tau), \text{M}'' \rangle} \tag{A3}$$

$$\frac{[\![expr]\!]^{\text{M}}_{\text{ADDR}} = \langle n_1, \text{M}' \rangle}{[\![expr\,.\,identifier]\!]^{\text{M}}_{\text{ADDR}} = \langle n_1 + \text{addr}(identifier), \text{M}' \rangle} \tag{A4}$$

$$\frac{[\![expr]\!]^{\text{M}}_{\text{EXPR}} = \langle n, \text{M}' \rangle}{[\![expr\texttt{\^{}}]\!]^{\text{M}}_{\text{ADDR}} = \langle n, \text{M}' \rangle} \tag{A5}$$

Expressions.

$$\frac{}{[\![\texttt{none}]\!]^{\text{M}}_{\text{EXPR}} = \langle \text{undef}, \text{M} \rangle} \quad \frac{}{[\![\texttt{nil}]\!]^{\text{M}}_{\text{EXPR}} = \langle 0, \text{M} \rangle} \tag{EX1}$$

$$\frac{}{[\![\texttt{true}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle 1, \mathrm{M}\rangle} \quad \frac{}{[\![\texttt{false}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle 0, \mathrm{M}\rangle} \tag{EX2}$$

$$\frac{}{[\![\mathit{char}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathrm{val}(\mathit{char}), \mathrm{M}\rangle} \quad \frac{}{[\![\mathit{int}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathrm{val}(\mathit{int}), \mathrm{M}\rangle} \tag{EX3}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle \quad \mathrm{op} \in \{\texttt{not}, \texttt{+}, \texttt{-}\}}{[\![\mathrm{op}\ \mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathrm{op}\ n, \mathrm{M}'\rangle} \tag{EX4}$$

$$\frac{[\![\mathit{expr}_1]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n_1, \mathrm{M}'\rangle \quad [\![\mathit{expr}_2]\!]^{\mathrm{M}'}_{\mathrm{EXPR}} = \langle n_2, \mathrm{M}''\rangle \quad \mathrm{op} \in \{\texttt{or}, \texttt{and}, \texttt{==}, \texttt{!=}, \texttt{<}, \texttt{>}, \texttt{<=}, \texttt{>=}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{\%}\}}{[\![\mathit{expr}_1\ \mathrm{op}\ \mathit{expr}_2]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n_1\ \mathrm{op}\ n_2, \mathrm{M}''\rangle} \tag{EX5}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{ADDR}} = \langle n, \mathrm{M}'\rangle}{[\![\texttt{\^{}}\,\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle} \quad \frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle}{[\![\mathit{expr}\texttt{\^{}}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathrm{M}'[n], \mathrm{M}'\rangle} \tag{EX6}$$

$$\frac{\mathrm{addr}(\mathit{identifier}) = a}{[\![\mathit{identifier}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle M[a], \mathrm{M}\rangle} \tag{EX7}$$

$$\frac{[\![\mathit{expr_1}\,\texttt{[}\mathit{expr_2}\texttt{]}]\!]^{\mathrm{M}}_{\mathrm{ADDR}} = \langle a, M'\rangle}{[\![\mathit{expr_1}\,\texttt{[}\mathit{expr_2}\texttt{]}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle M'[a], \mathrm{M}'\rangle} \tag{EX8}$$

$$\frac{[\![\mathit{expr}\,\texttt{.}\,\mathit{identifier}]\!]^{\mathrm{M}}_{\mathrm{ADDR}} = \langle a, M'\rangle}{[\![\mathit{expr}\,\texttt{.}\,\mathit{identifier}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle M'[a], \mathrm{M}'\rangle} \tag{EX9}$$

$$\frac{[\![\mathit{expr}_1]\!]^{\mathrm{M}_0}_{\mathrm{EXPR}} = \langle n_1, \mathrm{M}_1\rangle\ \ldots\ [\![\mathit{expr}_m]\!]^{\mathrm{M}_{m-1}}_{\mathrm{EXPR}} = \langle n_m, \mathrm{M}_m\rangle}{[\![\mathit{identifier}\,\texttt{(}\mathit{expr}_1, \ldots, \mathit{expr}_m\texttt{)}]\!]^{\mathrm{M}_0}_{\mathrm{EXPR}} = \langle \mathit{identifier}(n_1, \ldots, n_m), \mathrm{M}_m\rangle} \tag{EX10}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle}{[\![\texttt{(}\mathit{expr}\texttt{)}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle} \tag{EX11}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle \quad [\![\mathit{type}]\!]_{\mathrm{ISTYPE}} \neq \mathbf{char}}{[\![\texttt{<}\mathit{type}\texttt{>}\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle} \tag{EX12}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle \quad [\![\mathit{type}]\!]_{\mathrm{ISTYPE}} = \mathbf{char}}{[\![\texttt{(}\mathit{expr}\,\texttt{:}\,\mathit{type}\texttt{)}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n \bmod 256, \mathrm{M}'\rangle} \tag{EX13}$$

Statements.

$$\frac{[\![\mathit{expr}]\!]^{M}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle}{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{STMT}} = \mathrm{M}'} \tag{ST1}$$

$$\frac{\begin{array}{c}[\![\mathit{expr}_1]\!]^{\mathrm{M}}_{\mathrm{ADDR}} = \langle n_1, \mathrm{M}'\rangle \quad [\![\mathit{expr}_2]\!]^{\mathrm{M}'}_{\mathrm{EXPR}} = \langle n_2, \mathrm{M}''\rangle \\ \forall a\colon \mathrm{M}'''[a] = \begin{cases} n_2 & a = n_1 \\ M''[a] & \text{otherwise} \end{cases}\end{array}}{[\![\mathit{expr}_1\texttt{=}\mathit{expr}_2]\!]^{\mathrm{M}}_{\mathrm{STMT}} = \mathrm{M}'''} \tag{ST2}$$

$$\frac{[\![expr]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathbf{true}, \mathrm{M}' \rangle \quad [\![stmt_1]\!]^{\mathrm{M}'}_{\mathrm{STMT}} = \mathrm{M}''}{[\![\texttt{if } expr \texttt{ then } stmt_1]\!]_{\mathrm{STMT}} = \mathrm{M}''} \tag{ST3}$$

$$\frac{[\![expr]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathbf{false}, \mathrm{M}' \rangle}{[\![\texttt{if } expr \texttt{ then } stmt_1]\!]_{\mathrm{STMT}} = \mathrm{M}'} \tag{ST4}$$

$$\frac{[\![expr]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathbf{true}, \mathrm{M}' \rangle \quad [\![stmt_1]\!]^{\mathrm{M}'}_{\mathrm{STMT}} = \mathrm{M}''}{[\![\texttt{if } expr \texttt{ then } stmt_1 \texttt{ else } stmt_2]\!]_{\mathrm{STMT}} = \mathrm{M}''} \tag{ST5}$$

$$\frac{[\![expr]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathbf{false}, \mathrm{M}' \rangle \quad [\![stmt_2]\!]^{\mathrm{M}'}_{\mathrm{STMT}} = \mathrm{M}''}{[\![\texttt{if } expr \texttt{ then } stmt_1 \texttt{ else } stmt_2]\!]_{\mathrm{STMT}} = \mathrm{M}''} \tag{ST6}$$

$$\frac{[\![expr]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathbf{true}, \mathrm{M}' \rangle \quad [\![stmt]\!]^{\mathrm{M}'}_{\mathrm{STMT}} = \mathrm{M}''}{[\![\texttt{while } expr \texttt{ : } stmt]\!]^{\mathrm{M}}_{\mathrm{STMT}} = [\![\texttt{while } expr \texttt{ : } stmt]\!]^{\mathrm{M}''}_{\mathrm{STMT}}} \tag{ST7}$$

$$\frac{[\![expr]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathbf{false}, \mathrm{M}' \rangle}{[\![\texttt{while } expr \texttt{ : } stmt]\!]^{\mathrm{M}}_{\mathrm{STMT}} = \mathrm{M}'} \tag{ST8}$$

$$\frac{[\![stmt_1]\!]^{\mathrm{M}_0}_{\mathrm{STMT}} = \mathrm{M}_1 \ \ldots \ [\![stmt_m]\!]^{\mathrm{M}_{m-1}}_{\mathrm{STMT}} = \mathrm{M}_m}{[\![\texttt{\{} stmt_1 \ \ldots \ stmt_m \texttt{\}}]\!]^{\mathrm{M}_0}_{\mathrm{STMT}} = \mathrm{M}_m} \tag{ST9}$$

$$\frac{[\![expr]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}' \rangle}{[\![\texttt{return } expr]\!]^{\mathrm{M}}_{\mathrm{STMT}} = \mathrm{M}' \ \wedge \text{function terminates returning } n} \tag{ST10}$$