

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

→ (*type-definition* | *variable-definition* | *function-definition*)⁺

type-definition

→ identifier = *type*

variable-definition

→ identifier : *type*

function-definition

→ identifier ((*parameters*)[?]) : *type* (= *statement* ({ *definitions* })[?])[?]

parameters

→ ([^])[?] identifier : *type* (, ([^])[?] identifier : *type*)^{*}

statement

→ *expression* ;

→ *expression* = *expression* ;

→ if *expression* then *statement* (else *statement*)[?]

→ while *expression* : *statement*

→ return *expression* ;

→ { (*statement*)⁺ }

type

→ void | bool | char | int

→ [intconst] *type*

→ [^] *type*

→ (*components*)

→ { *components* }

→ identifier

components

→ identifier : *type* (, identifier : *type*)^{*}

expression

→ voidconst | boolconst | charconst | intconst | strconst | ptrconst

→ identifier (((*expression* (, *expression*)^{*})[?])[?])[?]

→ prefix-operator *expression*

→ *expression* binary-operator *expression*

→ < *type* > *expression*

→ *expression* [*expression*]

→ *expression* . identifier

→ *expression* [^]

→ sizeof (*type*)

→ (*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:

<i>postfix operators</i>	<code>[.] ^ .</code>	THE HIGHEST PRECEDENCE	
<i>prefix operators</i>	<code>not + - ^ <.></code>		
<i>multiplicative operators</i>	<code>* / %</code>		
<i>additive operators</i>	<code>+ -</code>		
<i>relational operators</i>	<code>== != < > <= >=</code>		
<i>conjunctive operator</i>	<code>and</code>		
<i>disjunctive operator</i>	<code>or</code>		
			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 $[[\cdot]]_{\text{BIND}}$ bind a name to its declaration according to the rules of namespaces and scopes described below. Hence, the value of function $[[\cdot]]_{\text{BIND}}$ 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

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

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

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

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

$$\llbracket \cdot \rrbracket_{\text{ISTYPE}}: \mathcal{P} \rightarrow \mathcal{T} \quad \text{and} \quad \llbracket \cdot \rrbracket_{\text{OFTYPE}}: \mathcal{P} \rightarrow \mathcal{T}$$

map syntactic phrases of LANG'24 to types. Function $\llbracket \cdot \rrbracket_{\text{ISTYPE}}$ denotes the type described by a phrase, function $\llbracket \cdot \rrbracket_{\text{OFTYPE}}$ 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.

$$\overline{\llbracket \mathbf{void} \rrbracket_{\text{ISTYPE}} = \mathbf{void}} \quad \overline{\llbracket \mathbf{bool} \rrbracket_{\text{ISTYPE}} = \mathbf{bool}} \quad \overline{\llbracket \mathbf{char} \rrbracket_{\text{ISTYPE}} = \mathbf{char}} \quad \overline{\llbracket \mathbf{int} \rrbracket_{\text{ISTYPE}} = \mathbf{int}} \quad (\text{T1})$$

$$\frac{\llbracket type \rrbracket_{\text{ISTYPE}} = \tau \quad \text{val}(int) = n \quad 0 < n \leq 2^{63} - 1 \quad \tau \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{\llbracket [int] type \rrbracket_{\text{ISTYPE}} = \mathbf{arr}(n \times \tau)} \quad (\text{T2})$$

$$\frac{n > 0 \quad \forall i \in \{1 \dots n\}: \llbracket type_i \rrbracket_{\text{ISTYPE}} = \tau_i \quad \tau_i \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{\llbracket (id_1: type_1, \dots, id_n: type_n) \rrbracket_{\text{ISTYPE}} = \mathbf{struct}_{id_1, \dots, id_n}(\tau_1, \dots, \tau_n)} \quad (\text{T3})$$

$$\frac{n > 0 \quad \forall i \in \{1 \dots n\}: \llbracket type_i \rrbracket_{\text{ISTYPE}} = \tau_i \quad \tau_i \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{\llbracket \{id_1: type_1, \dots, id_n: type_n\} \rrbracket_{\text{ISTYPE}} = \mathbf{union}_{id_1, \dots, id_n}(\tau_1, \dots, \tau_n)} \quad (\text{T4})$$

$$\frac{\llbracket type \rrbracket_{\text{ISTYPE}} = \tau \quad \tau \in \mathcal{T}_d}{\llbracket \uparrow type \rrbracket_{\text{ISTYPE}} = \mathbf{ptr}(\tau)} \quad (\text{T5})$$

Value expressions.

$$\overline{\llbracket \mathbf{none} \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad \overline{\llbracket \mathbf{nil} \rrbracket_{\text{OFTYPE}} = \mathbf{ptr}(\mathbf{void})} \quad \overline{\llbracket string \rrbracket_{\text{OFTYPE}} = \mathbf{ptr}(\mathbf{char})} \quad (\text{v1})$$

$$\overline{\llbracket bool \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad \overline{\llbracket char \rrbracket_{\text{OFTYPE}} = \mathbf{char}} \quad \overline{\llbracket int \rrbracket_{\text{OFTYPE}} = \mathbf{int}} \quad (\text{v2})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{bool}}{\llbracket \text{not } expr \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad \frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{int} \quad op \in \{+, -\}}{\llbracket op \ expr \rrbracket_{\text{OFTYPE}} = \mathbf{int}} \quad (\text{v3})$$

$$\frac{\llbracket expr_1 \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad \llbracket expr_2 \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad op \in \{\mathbf{and}, \mathbf{or}\}}{\llbracket expr_1 \ op \ expr_2 \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad (\text{v4})$$

$$\frac{\llbracket expr_1 \rrbracket_{\text{OFTYPE}} = \mathbf{int} \quad \llbracket expr_2 \rrbracket_{\text{OFTYPE}} = \mathbf{int} \quad op \in \{+, -, *, /, \% \}}{\llbracket expr_1 \ op \ expr_2 \rrbracket_{\text{OFTYPE}} = \mathbf{int}} \quad (\text{v5})$$

$$\frac{\llbracket expr_1 \rrbracket_{\text{OFTYPE}} = \tau \quad \llbracket expr_2 \rrbracket_{\text{OFTYPE}} = \tau \quad \tau \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \quad op \in \{=, !=\}}{\llbracket expr_1 \ op \ expr_2 \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad (\text{v6})$$

$$\frac{\llbracket expr_1 \rrbracket_{\text{OFTYPE}} = \tau \quad \llbracket expr_2 \rrbracket_{\text{OFTYPE}} = \tau \quad \tau \in \{\mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \quad op \in \{<=, >=, <, >\}}{\llbracket expr_1 \ op \ expr_2 \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad (\text{v7})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \tau \quad \llbracket expr \rrbracket_{\text{ISLVAL}} = \mathbf{true}}{\llbracket \hat{\ } expr \rrbracket_{\text{OFTYPE}} = \mathbf{ptr}(\tau)} \quad \frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{ptr}(\tau)}{\llbracket expr \ \hat{\ } \rrbracket_{\text{OFTYPE}} = \tau} \quad (\text{v8})$$

$$\frac{\llbracket expr_1 \rrbracket_{\text{OFTYPE}} = \mathbf{arr}(n \times \tau) \quad \llbracket expr_2 \rrbracket_{\text{OFTYPE}} = \mathbf{int} \quad \llbracket expr_1 \rrbracket_{\text{ISLVAL}} = \mathbf{true}}{\llbracket expr_1 \ [\ expr_2 \] \rrbracket_{\text{OFTYPE}} = \tau} \quad (\text{v9})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{struct}_{id_1, \dots, id_n}(\tau_1, \dots, \tau_n) \quad identifier = id_i}{\llbracket expr . identifier \rrbracket_{\text{OFTYPE}} = \tau_i} \quad (\text{v10})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{union}_{id_1, \dots, id_n}(\tau_1, \dots, \tau_n) \quad identifier = id_i}{\llbracket expr . identifier \rrbracket_{\text{OFTYPE}} = \tau_i} \quad (\text{v11})$$

$$\frac{\begin{array}{l} \llbracket identifier \rrbracket_{\text{OFTYPE}} = (\tau_1, \dots, \tau_n) \rightarrow \tau \\ \forall i \in \{1 \dots n\}: \llbracket expr_i \rrbracket_{\text{OFTYPE}} = \tau_i \wedge \tau_i \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \tau \in \{\mathbf{void}, \mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \forall i \in \{1 \dots n\}: (\text{the } i\text{-th parameter is a call-by-reference}) \implies \llbracket expr_i \rrbracket_{\text{ISLVAL}} = \mathbf{true} \end{array}}{\llbracket identifier(expr_1, \dots, expr_n) \rrbracket_{\text{OFTYPE}} = \tau} \quad (\text{v12})$$

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

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \tau}{\llbracket (expr) \rrbracket_{\text{OFTYPE}} = \tau} \quad \frac{\llbracket type \rrbracket_{\text{ISTYPE}} = \tau}{\llbracket \text{sizeof}(type) \rrbracket_{\text{OFTYPE}} = \mathbf{int}} \quad (\text{v14})$$

Statements.

$$\frac{\begin{array}{l} \llbracket expr_1 \rrbracket_{\text{OFTYPE}} = \tau \quad \llbracket expr_2 \rrbracket_{\text{OFTYPE}} = \tau \\ \tau \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \llbracket expr_1 \rrbracket_{\text{ISLVAL}} = \mathbf{true} \end{array}}{\llbracket expr_1 = expr_2 ; \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s1})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{void}}{\llbracket expr ; \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s2})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad \llbracket stmts \rrbracket_{\text{OFTYPE}} = \tau}{\llbracket \text{if } expr \text{ then } stmts \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s3})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad \llbracket stmts_1 \rrbracket_{\text{OFTYPE}} = \tau_1 \quad \llbracket stmts_2 \rrbracket_{\text{OFTYPE}} = \tau_2}{\llbracket \text{if } expr \text{ then } stmts_1 \text{ else } stmts_2 \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s4})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad \llbracket stmts \rrbracket_{\text{OFTYPE}} = \tau}{\llbracket \text{while } expr : stmts \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s5})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \tau \quad (\text{the result type of the innermost function is } \tau)}{\llbracket \text{return } expr ; \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s6})$$

$$\frac{n > 0 \quad \forall i \in \{1 \dots n\}: \llbracket stmt_i \rrbracket_{\text{OFTYPE}} = \tau_i}{\llbracket \{stmt_1 \dots stmt_n\} \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s7})$$

Declarations.

$$\frac{\llbracket identifier \rrbracket_{\text{BIND}} = identifier = type \quad \llbracket type \rrbracket_{\text{ISTYPE}} = \tau}{\llbracket identifier \rrbracket_{\text{ISTYPE}} = \tau} \quad (\text{D1})$$

$$\frac{\llbracket identifier \rrbracket_{\text{BIND}} = identifier : type \quad \llbracket type \rrbracket_{\text{ISTYPE}} = \tau \quad \tau \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{\llbracket identifier \rrbracket_{\text{OFTYPE}} = \tau} \quad (\text{D2})$$

$$\frac{\begin{array}{l} \llbracket identifier \rrbracket_{\text{BIND}} = identifier (identifier_1 : type_1, \dots, identifier_n : type_n) : type \\ \forall i \in \{1 \dots n\}: \llbracket type_i \rrbracket_{\text{ISTYPE}} = \tau_i \wedge \tau_i \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \llbracket type \rrbracket_{\text{ISTYPE}} = \tau \quad \tau \in \{\mathbf{void}, \mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \end{array}}{\llbracket identifier \rrbracket_{\text{OFTYPE}} = (\tau_1, \dots, \tau_n) \rightarrow \tau} \quad (\text{D3})$$

$$\frac{\begin{array}{l} \llbracket identifier \rrbracket_{\text{BIND}} = identifier (identifier_1 : type_1, \dots, identifier_n : type_n) : type = stmt \\ \forall i \in \{1 \dots n\}: \llbracket type_i \rrbracket_{\text{ISTYPE}} = \tau_i \wedge \tau_i \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \llbracket type \rrbracket_{\text{ISTYPE}} = \tau \quad \llbracket stmt \rrbracket_{\text{OFTYPE}} = \mathbf{void} \quad \tau \in \{\mathbf{void}, \mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \end{array}}{\llbracket identifier \rrbracket_{\text{OFTYPE}} = (\tau_1, \dots, \tau_n) \rightarrow \tau} \quad (\text{D4})$$

3.3 Lvalues

The semantic function

$$\llbracket \cdot \rrbracket_{\text{ISLVAL}} : \mathcal{P} \rightarrow \{\mathbf{true}, \mathbf{false}\}$$

denotes which phrases represent lvalues.

$$\frac{\llbracket identifier \rrbracket_{\text{BIND}} = \text{variable declaration}}{\llbracket identifier \rrbracket_{\text{ISLVAL}} = \mathbf{true}} \quad \frac{\llbracket identifier \rrbracket_{\text{BIND}} = \text{parameter declaration}}{\llbracket identifier \rrbracket_{\text{ISLVAL}} = \mathbf{true}}$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{ptr}(\tau)}{\llbracket expr \wedge \rrbracket_{\text{ISLVAL}} = \mathbf{true}} \quad \frac{\llbracket expr \rrbracket_{\text{ISLVAL}} = \mathbf{true}}{\llbracket expr [expr'] \rrbracket_{\text{ISLVAL}} = \mathbf{true}} \quad \frac{\llbracket expr \rrbracket_{\text{ISLVAL}} = \mathbf{true}}{\llbracket expr . identifier \rrbracket_{\text{ISLVAL}} = \mathbf{true}}$$

$$\frac{\llbracket expr \rrbracket_{\text{ISLVAL}} = \mathbf{true}}{\llbracket \langle type \rangle expr \rrbracket_{\text{ISLVAL}} = \mathbf{true}}$$

In all other cases the value of $\llbracket \cdot \rrbracket_{\text{ISLVAL}}$ 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

$$\begin{aligned} \llbracket \cdot \rrbracket_{\text{ADDR}} &: \mathcal{P} \times \mathcal{M} \rightarrow \mathcal{I} \times \mathcal{M} \\ \llbracket \cdot \rrbracket_{\text{EXPR}} &: \mathcal{P} \times \mathcal{M} \rightarrow \mathcal{I} \times \mathcal{M} \\ \llbracket \cdot \rrbracket_{\text{STMT}} &: \mathcal{P} \times \mathcal{M} \rightarrow \mathcal{M} \end{aligned}$$

where \mathcal{P} denotes the set of phrases of PREV'23, \mathcal{I} denotes the set of 64-bit integers, and \mathcal{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).

Auxiliary 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. Auxiliary function `sizeof` returns the size of a type. Auxiliary function `val` returns the value of an integer constant or an ASCII code of a char constant.

Addresses.

$$\frac{}{\llbracket string \rrbracket_{\text{ADDR}}^{\mathcal{M}} = \langle \text{addr}(string), \mathcal{M} \rangle} \quad (\text{A1})$$

$$\frac{\text{addr}(identifier) = a}{\llbracket identifier \rrbracket_{\text{ADDR}}^{\mathcal{M}} = \langle a, \mathcal{M} \rangle} \quad (\text{A2})$$

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

$$\frac{\llbracket expr \rrbracket_{\text{ADDR}}^{\mathcal{M}} = \langle n_1, \mathcal{M}' \rangle}{\llbracket expr . identifier \rrbracket_{\text{ADDR}}^{\mathcal{M}} = \langle n_1 + \text{addr}(identifier), \mathcal{M}' \rangle} \quad (\text{A4})$$

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^{\mathcal{M}} = \langle n, \mathcal{M}' \rangle}{\llbracket expr \wedge \rrbracket_{\text{ADDR}}^{\mathcal{M}} = \langle n, \mathcal{M}' \rangle} \quad (\text{A5})$$

Expressions.

$$\frac{}{\llbracket \mathbf{none} \rrbracket_{\text{EXPR}}^{\mathcal{M}} = \langle \text{undef}, \mathcal{M} \rangle} \quad \frac{}{\llbracket \mathbf{nil} \rrbracket_{\text{EXPR}}^{\mathcal{M}} = \langle 0, \mathcal{M} \rangle} \quad (\text{EX1})$$

$$\frac{}{\llbracket \mathbf{true} \rrbracket_{\text{EXPR}}^M = \langle 1, M \rangle \quad \llbracket \mathbf{false} \rrbracket_{\text{EXPR}}^M = \langle 0, M \rangle} \quad (\text{EX2})$$

$$\frac{}{\llbracket \mathbf{char} \rrbracket_{\text{EXPR}}^M = \langle \text{val}(\mathbf{char}), M \rangle \quad \llbracket \mathbf{int} \rrbracket_{\text{EXPR}}^M = \langle \text{val}(\mathbf{int}), M \rangle} \quad (\text{EX3})$$

$$\frac{\llbracket \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle \quad \text{op} \in \{\mathbf{not}, +, -\}}{\llbracket \text{op } \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle \text{op } n, M' \rangle} \quad (\text{EX4})$$

$$\frac{\llbracket \mathbf{expr}_1 \rrbracket_{\text{EXPR}}^M = \langle n_1, M' \rangle \quad \llbracket \mathbf{expr}_2 \rrbracket_{\text{EXPR}}^{M'} = \langle n_2, M'' \rangle \quad \text{op} \in \{\mathbf{or}, \mathbf{and}, \mathbf{==}, \mathbf{!=}, \mathbf{<}, \mathbf{>}, \mathbf{<=}, \mathbf{>=}, +, -, *, /, \% \}}{\llbracket \mathbf{expr}_1 \text{ op } \mathbf{expr}_2 \rrbracket_{\text{EXPR}}^M = \langle n_1 \text{ op } n_2, M'' \rangle} \quad (\text{EX5})$$

$$\frac{\llbracket \mathbf{expr} \rrbracket_{\text{ADDR}}^M = \langle n, M' \rangle \quad \llbracket \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle}{\llbracket \sim \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle \quad \llbracket \mathbf{expr} \sim \rrbracket_{\text{EXPR}}^M = \langle M'[n], M' \rangle} \quad (\text{EX6})$$

$$\frac{\text{addr}(\mathbf{identifier}) = a}{\llbracket \mathbf{identifier} \rrbracket_{\text{EXPR}}^M = \langle M[a], M \rangle} \quad (\text{EX7})$$

$$\frac{\llbracket \mathbf{expr}_1 \llbracket \mathbf{expr}_2 \rrbracket \rrbracket_{\text{ADDR}}^M = \langle a, M' \rangle}{\llbracket \mathbf{expr}_1 \llbracket \mathbf{expr}_2 \rrbracket \rrbracket_{\text{EXPR}}^M = \langle M'[a], M' \rangle} \quad (\text{EX8})$$

$$\frac{\llbracket \mathbf{expr} . \mathbf{identifier} \rrbracket_{\text{ADDR}}^M = \langle a, M' \rangle}{\llbracket \mathbf{expr} . \mathbf{identifier} \rrbracket_{\text{EXPR}}^M = \langle M'[a], M' \rangle} \quad (\text{EX9})$$

$$\frac{\llbracket \mathbf{expr}_1 \rrbracket_{\text{EXPR}}^{M_0} = \langle n_1, M_1 \rangle \dots \llbracket \mathbf{expr}_m \rrbracket_{\text{EXPR}}^{M_{m-1}} = \langle n_m, M_m \rangle}{\llbracket \mathbf{identifier}(\mathbf{expr}_1, \dots, \mathbf{expr}_m) \rrbracket_{\text{EXPR}}^{M_0} = \langle \mathbf{identifier}(n_1, \dots, n_m), M_m \rangle} \quad (\text{EX10})$$

$$\frac{\llbracket \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle}{\llbracket \langle \mathbf{expr} \rangle \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle} \quad (\text{EX11})$$

$$\frac{\llbracket \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle \quad \llbracket \mathbf{type} \rrbracket_{\text{ISTYPE}} \neq \mathbf{char}}{\llbracket \langle \mathbf{type} \rangle \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle} \quad (\text{EX12})$$

$$\frac{\llbracket \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle \quad \llbracket \mathbf{type} \rrbracket_{\text{ISTYPE}} = \mathbf{char}}{\llbracket \langle \mathbf{type} \rangle \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle n \bmod 256, M' \rangle} \quad (\text{EX13})$$

Statements.

$$\frac{\llbracket \mathbf{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle}{\llbracket \mathbf{expr} \rrbracket_{\text{STMT}}^M = M'} \quad (\text{ST1})$$

$$\frac{\llbracket \mathbf{expr}_1 \rrbracket_{\text{ADDR}}^M = \langle n_1, M' \rangle \quad \llbracket \mathbf{expr}_2 \rrbracket_{\text{EXPR}}^{M'} = \langle n_2, M'' \rangle \quad \forall a: M'''[a] = \begin{cases} n_2 & a = n_1 \\ M''[a] & \text{otherwise} \end{cases}}{\llbracket \mathbf{expr}_1 = \mathbf{expr}_2 \rrbracket_{\text{STMT}}^M = M'''} \quad (\text{ST2})$$

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^M = \langle \mathbf{true}, M' \rangle \quad \llbracket stmt_1 \rrbracket_{\text{STMT}}^{M'} = M''}{\llbracket \mathbf{if} \ expr \ \mathbf{then} \ stmt_1 \rrbracket_{\text{STMT}} = M''} \quad (\text{ST3})$$

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^M = \langle \mathbf{false}, M' \rangle}{\llbracket \mathbf{if} \ expr \ \mathbf{then} \ stmt_1 \rrbracket_{\text{STMT}} = M'} \quad (\text{ST4})$$

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^M = \langle \mathbf{true}, M' \rangle \quad \llbracket stmt_1 \rrbracket_{\text{STMT}}^{M'} = M''}{\llbracket \mathbf{if} \ expr \ \mathbf{then} \ stmt_1 \ \mathbf{else} \ stmt_2 \rrbracket_{\text{STMT}} = M''} \quad (\text{ST5})$$

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^M = \langle \mathbf{false}, M' \rangle \quad \llbracket stmt_2 \rrbracket_{\text{STMT}}^{M'} = M''}{\llbracket \mathbf{if} \ expr \ \mathbf{then} \ stmt_1 \ \mathbf{else} \ stmt_2 \rrbracket_{\text{STMT}} = M''} \quad (\text{ST6})$$

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^M = \langle \mathbf{true}, M' \rangle \quad \llbracket stmt \rrbracket_{\text{STMT}}^{M'} = M''}{\llbracket \mathbf{while} \ expr \ : \ stmt \rrbracket_{\text{STMT}}^M = \llbracket \mathbf{while} \ expr \ : \ stmt \rrbracket_{\text{STMT}}^{M''}} \quad (\text{ST7})$$

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^M = \langle \mathbf{false}, M' \rangle}{\llbracket \mathbf{while} \ expr \ : \ stmt \rrbracket_{\text{STMT}}^M = M'} \quad (\text{ST8})$$

$$\frac{\llbracket stmt_1 \rrbracket_{\text{STMT}}^{M_0} = M_1 \ \dots \ \llbracket stmt_m \rrbracket_{\text{STMT}}^{M_{m-1}} = M_m}{\llbracket \{ stmt_1 \ \dots \ stmt_m \} \rrbracket_{\text{STMT}}^{M_0} = M_m} \quad (\text{ST9})$$

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle}{\llbracket \mathbf{return} \ expr \rrbracket_{\text{STMT}}^M = M' \ \wedge \ \mathbf{function \ terminates \ returning} \ n} \quad (\text{ST10})$$