

The PREV'23 Programming Language

1 Lexical structure

Programs in the PREV'23 programming language are written in ASCII character set (no additional characters denoting post-alveolar consonants are allowed).

Programs in the PREV'23 programming language consist of the following lexical elements:

- *Constants:*
 - constant of type void: **none**
 - constants of type boolean: **true false**
 - constants of type integer:
 - A nonempty finite string of digits (0...9), not 0 padded but optionally preceded by a sign (+ or -).
 - constants of type char:
 - A character with ASCII code in range {32...126} enclosed in single quotes ('); a single quote in the constant (except at the beginning and at the end) must be preceded with backslash (\).
 - string constants:
 - A (possibly empty) string of character with ASCII codes in range {32...126} enclosed in double quotes ("); each double quote in the constant (except at the beginning and at the end) must be preceded with a backslash (\).
 - constants of pointer types: **nil**
- *Symbols:*
 - () { } [] . , : ; & | ! == != < > <= >= * / % + - ^ =
- *Keywords:*
 - bool char del do else fun if in int let new then typ var void while**
- *Identifiers:*
 - A nonempty finite string of letters (A...Z and a...z), digits (0...9), and underscores (_) that (a) starts with either a letter or an underscore and (b) is not a keyword or a constant.
- *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 tab is 8 spaces wide.

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

2 Syntax structure

The concrete syntax of the PREV'23 programming language is defined by context free grammar with the start symbol *declarations* and the following productions:

declarations

$$\begin{aligned} \longrightarrow & \left(\textit{type-declarations} \mid \textit{function-declarations} \mid \textit{variable-declarations} \right) \\ & \left\{ \left(\textit{type-declarations} \mid \textit{function-declarations} \mid \textit{variable-declarations} \right) \right\} \end{aligned}$$

type-declarations

→ **typ** identifier = *type* { , identifier = *type* } ;

function-declarations

→ **fun** identifier ([identifier : *type* { , identifier : *type* }]) : *type* [= *statement*]
 { , identifier ([identifier : *type* { , identifier : *type* }]) : *type* [= *statement*] } ;

variable-declarations

→ **var** identifier : *type* { , identifier : *type* } ;

type

→ **void** | **char** | **int** | **bool** | identifier
 → [*expression*] *type*
 → ^ *type*
 → { identifier : *type* { , identifier : *type* } }
 → (*type*)

expression

→ constant
 → identifier [([*expression* { , *expression* }])]
 → *expression* ([*expression*] | ^ | . identifier)
 → unary-operator *expression*
 → *expression* binary-operator *expression*
 → (*expression* [: *type*])
 → **new** (*type*) | **del** (*expression*)

statement

→ *expression* [= *expression*]
 → **if** *expression* **then** *statement* [**else** *statement*]
 → **while** *expression* **do** *statement*
 → **let** *declarations* **in** *statement*
 → { *statement* { ; *statement* } }

Unary operators are !, +, - and ^.

Binary operators are |, &, ==, !=, <, >, <=, >=, *, /, %, + and -.

The precedence of the operators is as follows:

<i>postfix operators</i>	[] ^ .	THE HIGHEST PRECEDENCE
<i>prefix operators</i>	! + - ^	
<i>multiplicative operators</i>	* / %	
<i>additive operators</i>	+ -	
<i>relational operators</i>	== != < > <= >=	
<i>conjunctive operator</i>	&	THE LOWEST PRECEDENCE
<i>disjunctive operator</i>		

Relational operators are non-associative, all other binary operators are left associative.

In the grammar above, braces typeset as { } enclose sentential forms that can be repeated zero or more times, brackets typeset as [] enclose sentential forms that can be present or not while braces typeset as { } and brackets typeset as [] denote symbols that are a part of the program text.

3 Semantic structure

Let function $\llbracket \cdot \rrbracket_{\text{BIND}}$ bind a name to its declaration according to the rules of namespaces and scopes described below. Hence, the value of function $\llbracket \cdot \rrbracket_{\text{BIND}}$ depends on the context of its argument.

3.1 Name binding

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 record components belong to record-specific namespaces, i.e., each record defines its own namespace containing names of its components.

Scopes. A new scope is created in two ways:

1. Each global **typ**, **fun** and **var** declaration group creates a new scope extending to the end of file. All names declared in the same declaration group belong to the same scope.
2. Each **typ**, **fun** and **var** declaration group within a **let declarations in statement** creates a new scope extending to the end of *statement*. All names declared in the same declaration group belong to the same scope.
3. Function declaration

$$\text{identifier} \left(\left[\text{identifier} : \text{type} \{, \text{identifier} : \text{type} \} \right] \right) : \text{type} [= \text{statement}]$$

creates a new scope. The name of a function, the types of parameters and the type of a result belong to the outer scope while the names of parameters and the *statement* (if present) denoting the function body belong to the scope created by the function declaration.

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

3.2 Type system

The set

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

denotes the set of all data types of PREV'23. 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 PREV'23.

Structural equality of types: Types τ_1 and τ_2 are equal 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 PREV'23 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.

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

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

$$\frac{\forall i \in \{1 \dots n\}: \llbracket \text{type}_i \rrbracket_{\text{ISTYPE}} = \tau_i \wedge \tau_i \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{\llbracket \{id_1 : \text{type}_1, \dots, id_n : \text{type}_n\} \rrbracket_{\text{ISTYPE}} = \mathbf{rec}_{id_1, \dots, id_n}(\tau_1, \dots, \tau_n)} \quad (\text{T3})$$

$$\frac{\llbracket \text{type} \rrbracket_{\text{ISTYPE}} = \tau \quad \tau \in \mathcal{T}_d}{\llbracket \lceil \text{type} \rrbracket_{\text{ISTYPE}} = \mathbf{ptr}(\tau)} \quad (\text{T4})$$

$$\frac{\llbracket \text{type} \rrbracket_{\text{ISTYPE}} = \tau}{\llbracket (\text{type}) \rrbracket_{\text{ISTYPE}} = \tau} \quad (\text{T5})$$

Value expressions.

$$\overline{\llbracket \text{none} \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad \overline{\llbracket \text{nil} \rrbracket_{\text{OFTYPE}} = \mathbf{ptr}(\mathbf{void})} \quad \overline{\llbracket \text{string} \rrbracket_{\text{OFTYPE}} = \mathbf{ptr}(\mathbf{char})} \quad (\text{V1})$$

$$\overline{\llbracket \text{bool} \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad \overline{\llbracket \text{char} \rrbracket_{\text{OFTYPE}} = \mathbf{char}} \quad \overline{\llbracket \text{int} \rrbracket_{\text{OFTYPE}} = \mathbf{int}} \quad (\text{V2})$$

$$\frac{\llbracket \text{expr} \rrbracket_{\text{OFTYPE}} = \mathbf{bool}}{\llbracket ! \text{expr} \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad \frac{\llbracket \text{expr} \rrbracket_{\text{OFTYPE}} = \mathbf{int} \quad op \in \{+, -\}}{\llbracket op \text{ expr} \rrbracket_{\text{OFTYPE}} = \mathbf{int}} \quad (\text{V3})$$

$$\frac{\llbracket \text{expr}_1 \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad \llbracket \text{expr}_2 \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad op \in \{\&, |\}}{\llbracket \text{expr}_1 \text{ op } \text{expr}_2 \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad (\text{V4})$$

$$\frac{\llbracket \text{expr}_1 \rrbracket_{\text{OFTYPE}} = \mathbf{int} \quad \llbracket \text{expr}_2 \rrbracket_{\text{OFTYPE}} = \mathbf{int} \quad op \in \{+, -, *, /, \%\}}{\llbracket \text{expr}_1 \text{ op } \text{expr}_2 \rrbracket_{\text{OFTYPE}} = \mathbf{int}} \quad (\text{V5})$$

$$\frac{\begin{array}{l} \llbracket \text{expr}_1 \rrbracket_{\text{OFTYPE}} = \tau \quad \llbracket \text{expr}_2 \rrbracket_{\text{OFTYPE}} = \tau \\ \tau \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \quad op \in \{==, !=\} \end{array}}{\llbracket \text{expr}_1 \text{ op } \text{expr}_2 \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad (\text{V6})$$

$$\frac{\begin{array}{l} \llbracket \text{expr}_1 \rrbracket_{\text{OFTYPE}} = \tau \quad \llbracket \text{expr}_2 \rrbracket_{\text{OFTYPE}} = \tau \\ \tau \in \{\mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \quad op \in \{<=, >=, <, >\} \end{array}}{\llbracket \text{expr}_1 \text{ op } \text{expr}_2 \rrbracket_{\text{OFTYPE}} = \mathbf{bool}} \quad (\text{V7})$$

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

$$\frac{\llbracket type \rrbracket_{\text{ISTYPE}} = \tau \quad \tau \in \mathcal{T}_d}{\llbracket \mathbf{new}(type) \rrbracket_{\text{OFTYPE}} = \mathbf{ptr}(\tau)} \quad \frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{ptr}(\tau)}{\llbracket \mathbf{del}(expr) \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{v9})$$

$$\frac{\llbracket expr_1 \rrbracket_{\text{OFTYPE}} = \mathbf{arr}(n \times \tau) \quad \llbracket expr_2 \rrbracket_{\text{OFTYPE}} = \mathbf{int}}{\llbracket expr_1 [expr_2] \rrbracket_{\text{OFTYPE}} = \tau} \quad (\text{v10})$$

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

$$\frac{\begin{array}{l} \llbracket identifier \rrbracket_{\text{OFTYPE}} = (\tau_1, \dots, \tau_n) \rightarrow \tau \\ \forall i \in \{1 \dots n\}: \llbracket expr_i \rrbracket_{\text{OFTYPE}} = \tau_i \wedge \tau_i \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \tau \in \{\mathbf{void}, \mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \end{array}}{\llbracket identifier(expr_1, \dots, expr_n) \rrbracket_{\text{OFTYPE}} = \tau} \quad (\text{v12})$$

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

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \tau}{\llbracket (expr) \rrbracket_{\text{OFTYPE}} = \tau} \quad (\text{v14})$$

Statements.

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

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad \llbracket stmts \rrbracket_{\text{OFTYPE}} = \tau}{\llbracket \mathbf{if} \ expr \ \mathbf{then} \ stmts \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s2})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad \llbracket stmts_1 \rrbracket_{\text{OFTYPE}} = \tau_1 \quad \llbracket stmts_2 \rrbracket_{\text{OFTYPE}} = \tau_2}{\llbracket \mathbf{if} \ expr \ \mathbf{then} \ stmts_1 \ \mathbf{else} \ stmts_2 \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s3})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \mathbf{bool} \quad \llbracket stmts \rrbracket_{\text{OFTYPE}} = \tau}{\llbracket \mathbf{while} \ expr \ \mathbf{do} \ stmts \rrbracket_{\text{OFTYPE}} = \mathbf{void}} \quad (\text{s4})$$

$$\frac{\llbracket expr \rrbracket_{\text{OFTYPE}} = \tau}{\llbracket (expr) \rrbracket_{\text{OFTYPE}} = \tau} \quad \frac{\llbracket stmt \rrbracket_{\text{OFTYPE}} = \tau}{\llbracket \mathbf{let} \ decls \ \mathbf{in} \ stmt \rrbracket_{\text{OFTYPE}} = \tau} \quad (\text{s5})$$

$$\frac{\forall i \in \{1 \dots n\}: \llbracket stmt_i \rrbracket_{\text{OFTYPE}} = \tau_i}{\llbracket \{stmt_1; \dots; stmt_n\} \rrbracket_{\text{OFTYPE}} = \tau_n} \quad (\text{s6})$$

Declarations.

$$\frac{\llbracket identifier \rrbracket_{\text{BIND}} = \mathbf{typ} \ identifier : type \quad \llbracket type \rrbracket_{\text{ISTYPE}} = \tau}{\llbracket identifier \rrbracket_{\text{ISTYPE}} = \tau} \quad (\text{d1})$$

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

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

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

3.3 Lvalues

The semantic function

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

denotes which phrases represent lvalues.

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

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

In all other cases the value of $\llbracket \cdot \rrbracket_{\text{ISADDR}}$ equals **false**.

3.4 Linkage

A variable or a function has external linkage if

- it is not declared inside a function and
- its declaration does not redeclare a name already declared at any outer scope.

3.5 Operational semantics

Operational semantics is described by semantic functions

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

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.

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

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

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

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

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

Expressions.

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

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

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

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle \quad \text{op} \in \{!, +, -\}}{\llbracket \text{op } expr \rrbracket_{\text{EXPR}}^M = \langle \text{op } n, M' \rangle} \quad (\text{EX4})$$

$$\frac{\llbracket expr_1 \rrbracket_{\text{EXPR}}^M = \langle n_1, M' \rangle \quad \llbracket expr_2 \rrbracket_{\text{EXPR}}^{M'} = \langle n_2, M'' \rangle \quad \text{op} \in \{!, \&, ==, !=, <, >, <=, >=, +, -, *, /, \%\}}{\llbracket expr_1 \text{ op } expr_2 \rrbracket_{\text{EXPR}}^M = \langle n_1 \text{ op } n_2, M'' \rangle} \quad (\text{EX5})$$

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

$$\frac{\llbracket \text{new}(\text{sizeof}(type)) \rrbracket_{\text{EXPR}}^M = \langle a, M' \rangle}{\llbracket \text{new } (expr) \rrbracket_{\text{EXPR}}^M = \langle a, M' \rangle} \quad (\text{EX7})$$

$$\frac{\llbracket expr \rrbracket_{\text{EXPR}}^M = \langle a, M' \rangle \quad \llbracket \text{del}(a) \rrbracket_{\text{EXPR}}^{M'} = \langle \text{undef}, M'' \rangle}{\llbracket \text{del } expr \rrbracket_{\text{EXPR}}^M = \langle \text{undef}, M'' \rangle} \quad (\text{EX8})$$

$$\frac{\text{addr}(\text{identifier}) = a}{\llbracket \text{identifier} \rrbracket_{\text{EXPR}}^M = \langle M[a], M \rangle} \quad (\text{EX9})$$

$$\frac{\llbracket \text{expr}_1 [\text{expr}_2] \rrbracket_{\text{ADDR}}^M = \langle a, M' \rangle}{\llbracket \text{expr}_1 [\text{expr}_2] \rrbracket_{\text{EXPR}}^M = \langle M'[a], M' \rangle} \quad (\text{EX10})$$

$$\frac{\llbracket \text{expr} . \text{identifier} \rrbracket_{\text{ADDR}}^M = \langle a, M' \rangle}{\llbracket \text{expr} . \text{identifier} \rrbracket_{\text{EXPR}}^M = \langle M'[a], M' \rangle} \quad (\text{EX11})$$

$$\frac{\llbracket \text{expr}_1 \rrbracket_{\text{EXPR}}^{M_0} = \langle n_1, M_1 \rangle \dots \llbracket \text{expr}_m \rrbracket_{\text{EXPR}}^{M_{m-1}} = \langle n_m, M_m \rangle}{\llbracket \text{identifier}(\text{expr}_1, \dots, \text{expr}_m) \rrbracket_{\text{EXPR}}^{M_0} = \langle \text{identifier}(n_1, \dots, n_m), M_m \rangle} \quad (\text{EX12})$$

$$\frac{\llbracket \text{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle}{\llbracket (\text{expr}) \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle} \quad (\text{EX13})$$

$$\frac{\llbracket \text{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle \quad \llbracket \text{type} \rrbracket_{\text{ISTYPE}} \neq \mathbf{char}}{\llbracket (\text{expr} : \text{type}) \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle} \quad (\text{EX14})$$

$$\frac{\llbracket \text{expr} \rrbracket_{\text{EXPR}}^M = \langle n, M' \rangle \quad \llbracket \text{type} \rrbracket_{\text{ISTYPE}} = \mathbf{char}}{\llbracket (\text{expr} : \text{type}) \rrbracket_{\text{EXPR}}^M = \langle n \bmod 256, M' \rangle} \quad (\text{EX15})$$

Statements.

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

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

$$\frac{\llbracket \text{expr} \rrbracket_{\text{EXPR}}^M = \langle \mathbf{true}, M' \rangle \quad \llbracket \text{stmt}_1 \rrbracket_{\text{STMT}}^{M'} = \langle \text{undef}, M'' \rangle}{\llbracket \text{if } \text{expr} \text{ then } \text{stmt}_1 \text{ else } \text{stmt}_2 \rrbracket_{\text{STMT}} = \langle \text{undef}, M'' \rangle} \quad (\text{ST3})$$

$$\frac{\llbracket \text{expr} \rrbracket_{\text{EXPR}}^M = \langle \mathbf{false}, M' \rangle \quad \llbracket \text{stmt}_2 \rrbracket_{\text{EXPR}}^{M'} = \langle \text{undef}, M'' \rangle}{\llbracket \text{if } \text{expr} \text{ then } \text{stmt}_1 \text{ else } \text{stmt}_2 \rrbracket_{\text{STMT}} = \langle \text{undef}, M'' \rangle} \quad (\text{ST4})$$

$$\frac{\llbracket \text{expr} \rrbracket_{\text{EXPR}}^M = \langle \mathbf{true}, M' \rangle \quad \llbracket \text{stmt} \rrbracket_{\text{STMT}}^{M'} = \langle \text{undef}, M'' \rangle}{\llbracket \text{while } \text{expr} \text{ do } \text{stmt} \rrbracket_{\text{STMT}}^M = \llbracket \text{while } \text{expr} \text{ do } \text{stmt} \rrbracket_{\text{STMT}}^{M''}} \quad (\text{ST5})$$

$$\frac{\llbracket \text{expr} \rrbracket_{\text{EXPR}}^M = \langle \mathbf{false}, M' \rangle}{\llbracket \text{while } \text{expr} \text{ do } \text{stmt} \rrbracket_{\text{STMT}}^M = \langle \text{undef}, M' \rangle} \quad (\text{ST6})$$

$$\frac{\llbracket \text{stmt} \rrbracket_{\text{STMT}}^M = \langle n, M' \rangle}{\llbracket \text{let } \text{decls} \text{ in } \text{stmt} \rrbracket_{\text{STMT}}^M = \langle n, M' \rangle} \quad (\text{ST7})$$

$$\frac{\llbracket \text{stmt}_1 \rrbracket_{\text{STMT}}^{M_0} = \langle n_1, M_1 \rangle \dots \llbracket \text{stmt}_m \rrbracket_{\text{STMT}}^{M_{m-1}} = \langle n_m, M_m \rangle}{\llbracket \{ \text{stmt}_1; \dots; \text{stmt}_m \} \rrbracket_{\text{STMT}}^{M_0} = \langle n_m, M_m \rangle} \quad (\text{ST8})$$