The PREV'20 Programming Language

1 Lexical structure

Programs in the PREV'20 programming language are written in ASCII character set (no additional characters denoting post-alveolar consonants are allowed).

Programs in the PREV'20 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`) optionally preceded by a sign (`+` or `-`).
  - constants of type char:
    A character with ASCI 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 ASCI 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*:
  `boolean char del do else fun if integer new then typ var void where 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'20 programming language is defined by context free grammar with the start symbol *prg* and productions

(program)	*prg*	$\longrightarrow$	*decl* { *decl* }
(type declaration)	*decl*	$\longrightarrow$	`typ` identifier `=` *type*
(variable declaration)	*decl*	$\longrightarrow$	`var` identifier `:` *type*
(function declaration)	*decl*	$\longrightarrow$	`fun` identifier `(` [identifier `:` *type* { `,` identifier `:` *type* }] `)` `:` *type* `=` *expr*

(atomic type)	*type*	⟶	void | char | integer | boolean
(named type)	*type*	⟶	identifier
(array type)	*type*	⟶	[*expr*] *type*
(pointer type)	*type*	⟶	^ *type*
(record type)	*type*	⟶	{ identifier : *type* { , identifier : *type* } }
(enclosed type)	*type*	⟶	(*type*)
(constant expression)	*expr*	⟶	const
(variable access)	*expr*	⟶	identifier
(function call)	*expr*	⟶	identifier ([*expr* { , *expr* }])
(allocation expression)	*expr*	⟶	(new | del) *expr*
(compound expression)	*expr*	⟶	{ *stmt* ; { *stmt* ; } }
(typecast expression)	*expr*	⟶	(*expr* : *type*)
(enclosed expression)	*expr*	⟶	(*expr*)
(infix expression)	*expr*	⟶	*expr* (& | | | == | != | < | > | <= | >= | * | / | % | + | -) *expr*
(prefix expression)	*expr*	⟶	(! | + | - | ^) *expr*
(postfix expression)	*expr*	⟶	*expr* ([*expr*] | ^ | . identifier)
(where expression)	*expr*	⟶	*expr* where { *decl* { *decl* } }
(expression statement)	*stmt*	⟶	*expr*
(assignment statement)	*stmt*	⟶	*expr* = *expr*
(conditional statement)	*stmt*	⟶	if *expr* then *stmt* else *stmt*
(loop statement)	*stmt*	⟶	while *expr* do *stmt*

where const denotes any constant.

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

The precedence of the operators is as follows:

postfix operators	[] ^ .	THE HIGHEST PRECEDENCE
prefix operators	! + - ^ new del	
multiplicative operators	* / %	
additive operators	+ -	
relational operators	== != < > <= >=	
conjunctive operator	&	
disjunctive operator	|	
declaration binder	where	THE LOWEST PRECEDENCE

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

3.1 Name binding

Let function $[\![\cdot]\!]_{\mathrm{BIND}}$ binds a name to its declaration according to the rules of namespaces and scopes as 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 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. Where-expression

   $$\textit{expr}\ \texttt{where}\ \texttt{\{}\ \textit{decl}\ \{\ \textit{decl}\}\ \texttt{\}}$$

   creates a new scope: it starts with *expr* and ends with `}`.

2. Function declaration

   $$\texttt{fun}\ \text{identifier}\ \texttt{(}\ [\,\text{identifier}\ \texttt{:}\ \textit{type}\ \{\ \texttt{,}\ \text{identifier}\ \texttt{:}\ \textit{type}\ \}\,]\ \texttt{)}\ \texttt{:}\ \textit{type}\ \texttt{=}\ \textit{expr}$$

   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 expression 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

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

denotes the set of all data types of PREV'20. 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 PREV'20.

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

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

maps syntactic phrases of PREV'20 to types. Function $[\![\cdot]\!]_{\text{ISTYPE}}$ denotes the type described by a phrase, function $[\![\cdot]\!]_{\text{OFTYPE}}$ 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.

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

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

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

$$\frac{[\![\mathit{type}]\!]_{\text{ISTYPE}} = \tau \quad \tau \in \mathcal{T}_d}{[\![\texttt{\^{}}\ \mathit{type}]\!]_{\text{ISTYPE}} = \mathbf{ptr}(\tau)} \tag{T4}$$

$$\frac{[\![\mathit{type}]\!]_{\text{ISTYPE}} = \tau}{[\![\texttt{(}\mathit{type}\texttt{)}]\!]_{\text{ISTYPE}} = \tau} \tag{T5}$$

Value expressions.

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

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

$$\frac{[\![\mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{bool}}{[\![\texttt{!}\ \mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{bool}} \quad \frac{[\![\mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{int} \quad \mathit{op} \in \{\texttt{+}, \texttt{-}\}}{[\![\mathit{op}\ \mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{int}} \tag{V3}$$

$$\frac{[\![\mathit{expr}_1]\!]_{\text{OFTYPE}} = \mathbf{bool} \quad [\![\mathit{expr}_2]\!]_{\text{OFTYPE}} = \mathbf{bool} \quad \mathit{op} \in \{\texttt{\&}, \texttt{|}\}}{[\![\mathit{expr}_1\ \mathit{op}\ \mathit{expr}_2]\!]_{\text{OFTYPE}} = \mathbf{bool}} \tag{V4}$$

$$\frac{[\![\mathit{expr}_1]\!]_{\text{OFTYPE}} = \mathbf{int} \quad [\![\mathit{expr}_2]\!]_{\text{OFTYPE}} = \mathbf{int} \quad \mathit{op} \in \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{\%}\}}{[\![\mathit{expr}_1\ \mathit{op}\ \mathit{expr}_2]\!]_{\text{OFTYPE}} = \mathbf{int}} \tag{V5}$$

$$\frac{\begin{array}{c}[\![\mathit{expr}_1]\!]_{\text{OFTYPE}} = \tau \quad [\![\mathit{expr}_2]\!]_{\text{OFTYPE}} = \tau \\ \tau \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \quad \mathit{op} \in \{\texttt{==}, \texttt{!=}\}\end{array}}{[\![\mathit{expr}_1\ \mathit{op}\ \mathit{expr}_2]\!]_{\text{OFTYPE}} = \mathbf{bool}} \tag{V6}$$

$$\frac{\begin{array}{c}[\![\mathit{expr}_1]\!]_{\text{OFTYPE}} = \tau \quad [\![\mathit{expr}_2]\!]_{\text{OFTYPE}} = \tau \\ \tau \in \{\mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \quad \mathit{op} \in \{\texttt{<=}, \texttt{>=}, \texttt{<}, \texttt{>}\}\end{array}}{[\![\mathit{expr}_1\ \mathit{op}\ \mathit{expr}_2]\!]_{\text{OFTYPE}} = \mathbf{bool}} \tag{V7}$$

$$\frac{[\![\mathit{expr}]\!]_{\text{OFTYPE}} = \tau}{[\![\texttt{\^{}}\ \mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{ptr}(\tau)} \quad \frac{[\![\mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{ptr}(\tau)}{[\![\mathit{expr}\ \texttt{\^{}}]\!]_{\text{OFTYPE}} = \tau} \tag{V8}$$

$$\frac{[\![\mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{int}}{[\![\texttt{new}\ \mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{ptr}(\mathbf{void})} \quad \frac{[\![\mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{ptr}(\tau)}{[\![\texttt{del}\ \mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{void}} \tag{V9}$$

$$\frac{[\![\mathit{expr}_1]\!]_{\text{OFTYPE}} = \mathbf{arr}(n \times \tau) \quad [\![\mathit{expr}_2]\!]_{\text{OFTYPE}} = \mathbf{int}}{[\![\mathit{expr}_1 \texttt{[} \mathit{expr}_2 \texttt{]}]\!]_{\text{OFTYPE}} = \tau} \tag{V10}$$

$$\frac{[\![\mathit{expr}]\!]_{\text{OFTYPE}} = \mathbf{rec}_{id_1,\dots,id_n}(\tau_1, \dots, \tau_n) \quad \mathit{identifier} = id_i}{[\![\mathit{expr}\,\texttt{.}\,\mathit{identifier}]\!]_{\text{OFTYPE}} = \tau_i} \tag{V11}$$

$$\frac{\begin{array}{c}[\![\mathit{identifier}]\!]_{\text{OFTYPE}} = (\tau_1, \dots, \tau_n) \to \tau \\ \forall i \in \{1 \dots n\}\colon [\![\mathit{expr}_i]\!]_{\text{OFTYPE}} = \tau_i \wedge \tau_i \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \tau \in \{\mathbf{void}, \mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\}\end{array}}{[\![\mathit{identifier}\texttt{(}\mathit{expr}_1\texttt{,} \dots \texttt{,}\mathit{expr}_n\texttt{)}]\!]_{\text{OFTYPE}} = \tau} \tag{V12}$$

$$\frac{\forall i \in \{1 \dots n\}\colon [\![\mathit{stmt}_i]\!]_{\text{OFTYPE}} = \tau_i}{[\![\texttt{\{}\mathit{stmt}_1\texttt{;} \dots \mathit{stmt}_n\texttt{;}\texttt{\}}]\!]_{\text{OFTYPE}} = \tau_n} \tag{V13}$$

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \tau_1 \quad \llbracket \mathit{type} \rrbracket_{\mathrm{ISTYPE}} = \tau_2 \quad \tau_1, \tau_2 \in \{\mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\}}{\llbracket \texttt{(}\mathit{expr} \,\texttt{:}\, \mathit{type}\texttt{)} \rrbracket_{\mathrm{OFTYPE}} = \tau_2} \tag{v14}$$

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \tau}{\llbracket \texttt{(}\mathit{expr}\texttt{)} \rrbracket_{\mathrm{OFTYPE}} = \tau} \quad \frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \tau}{\llbracket \mathit{expr} \ \texttt{where} \ \mathit{decls} \rrbracket_{\mathrm{OFTYPE}} = \tau} \tag{v15}$$

Statements.

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

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{bool} \quad \llbracket \mathit{stmts}_1 \rrbracket_{\mathrm{OFTYPE}} = \tau_1 \quad \llbracket \mathit{stmts}_2 \rrbracket_{\mathrm{OFTYPE}} = \tau_2}{\llbracket \texttt{if} \ \mathit{expr} \ \texttt{then} \ \mathit{stmts}_1 \ \texttt{else} \ \mathit{stmts}_2 \ \rrbracket_{\mathrm{OFTYPE}} = \mathbf{void}} \tag{s2}$$

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{bool} \quad \llbracket \mathit{stmts} \rrbracket_{\mathrm{OFTYPE}} = \tau}{\llbracket \texttt{while} \ \mathit{expr} \ \texttt{do} \ \mathit{stmts} \ \rrbracket_{\mathrm{OFTYPE}} = \mathbf{void}} \tag{s3}$$

Declarations.

$$\frac{\llbracket \mathit{identifier} \rrbracket_{\mathrm{BIND}} = \texttt{typ} \ \mathit{identifier} \,\texttt{:}\, \mathit{type} \quad \llbracket \mathit{type} \rrbracket_{\mathrm{ISTYPE}} = \tau}{\llbracket \mathit{identifier} \rrbracket_{\mathrm{ISTYPE}} = \tau} \tag{d1}$$

$$\frac{\llbracket \mathit{identifier} \rrbracket_{\mathrm{BIND}} = \texttt{var} \ \mathit{identifier} \,\texttt{:}\, \mathit{type} \quad \llbracket \mathit{type} \rrbracket_{\mathrm{ISTYPE}} = \tau \quad \tau \in \mathcal{T}_d \setminus \{\mathbf{void}\}}{\llbracket \mathit{identifier} \rrbracket_{\mathrm{OFTYPE}} = \tau} \tag{d2}$$

$$\frac{\begin{array}{c} \llbracket \mathit{identifier} \rrbracket_{\mathrm{BIND}} = \texttt{fun} \ \mathit{identifier}\,\texttt{(}\mathit{identifer}_1 \,\texttt{:}\, \mathit{type}_1, \ldots, \mathit{identifer}_n \,\texttt{:}\, \mathit{type}_n\texttt{)} \,\texttt{:}\, \mathit{type} \,\texttt{=}\, \mathit{expr} \\ \forall i \in \{1 \ldots n\}\colon \llbracket \mathit{type}_i \rrbracket_{\mathrm{ISTYPE}} = \tau_i \land \tau_i \in \{\mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \\ \llbracket \mathit{type} \rrbracket_{\mathrm{ISTYPE}} = \tau \quad \llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \tau \quad \tau \in \{\mathbf{void}, \mathbf{bool}, \mathbf{char}, \mathbf{int}\} \cup \{\mathbf{ptr}(\tau) \mid \tau \in \mathcal{T}_d\} \end{array}}{\llbracket \mathit{identifier} \rrbracket_{\mathrm{OFTYPE}} = (\tau_1, \ldots, \tau_n) \to \tau} \tag{d3}$$

3.3 Lvalues

The semantic function

$$\llbracket \cdot \rrbracket_{\mathrm{ISADDR}}\colon \mathcal{P} \to \{\mathbf{true}, \mathbf{false}\}$$

denotes which phrases represent lvalues.

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

$$\frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{OFTYPE}} = \mathbf{ptr}(\tau)}{\llbracket \mathit{expr}\texttt{\^{}} \rrbracket_{\mathrm{ISADDR}} = \mathbf{true}} \quad \frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{ISADDR}} = \mathbf{true}}{\llbracket \mathit{expr}\,\texttt{[}\mathit{expr}'\texttt{]} \rrbracket_{\mathrm{ISADDR}} = \mathbf{true}} \quad \frac{\llbracket \mathit{expr} \rrbracket_{\mathrm{ISADDR}} = \mathbf{true}}{\llbracket \mathit{expr}\,\texttt{.}\,\mathit{identifier} \rrbracket_{\mathrm{ISADDR}} = \mathbf{true}}$$

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

3.4 Operational semantics

Operational semantics is described by semantic functions

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

where P denotes the set of phrases of PREV'20, 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.

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

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

$$\frac{[\![\mathit{expr}_1]\!]^{\text{M}}_{\text{ADDR}} = \langle n_1, \text{M}'\rangle \quad [\![\mathit{expr}_2]\!]^{\text{M}'}_{\text{EXPR}} = \langle n_2, \text{M}''\rangle \quad [\![\mathit{expr}_1]\!]_{\text{OFTYPE}} = \mathbf{arr}(n \times \tau)}{[\![\mathit{expr_1}\,\texttt{[}\mathit{expr_2}\texttt{]}]\!]^{\text{M}}_{\text{ADDR}} = \langle n_1 + n_2 * \text{sizeof}(\tau), \text{M}''\rangle} \tag{A3}$$

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

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

Expressions.

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

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

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

$$\frac{[\![\mathit{expr}]\!]^{\text{M}}_{\text{EXPR}} = \langle n, \text{M}'\rangle \quad \text{op} \in \{\texttt{!}, \texttt{+}, \texttt{-}\}}{[\![\text{op } \mathit{expr}]\!]^{\text{M}}_{\text{EXPR}} = \langle \text{op } n, \text{M}'\rangle} \tag{EX4}$$

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

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

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle \quad [\![\mathrm{new}(n)]\!]^{\mathrm{M}'}_{\mathrm{EXPR}} = \langle a, \mathrm{M}''\rangle}{[\![\texttt{new}\ \mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle a, \mathrm{M}''\rangle} \tag{EX7}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle a, \mathrm{M}'\rangle \quad [\![\mathrm{del}(a)]\!]^{\mathrm{M}'}_{\mathrm{EXPR}} = \langle \mathrm{undef}, \mathrm{M}''\rangle}{[\![\texttt{del}\ \mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathrm{undef}, \mathrm{M}''\rangle} \tag{EX8}$$

$$\frac{\mathrm{addr}(\mathit{identifier}) = a}{[\![\mathit{identifier}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle M[a], \mathrm{M}\rangle} \tag{EX9}$$

$$\frac{[\![\mathit{expr_1}\,\texttt{[}\mathit{expr_2}\texttt{]}]\!]^{\mathrm{M}}_{\mathrm{ADDR}} = \langle a, M'\rangle}{[\![\mathit{expr_1}\,\texttt{[}\mathit{expr_2}\texttt{]}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle M'[a], \mathrm{M}'\rangle} \tag{EX10}$$

$$\frac{[\![\mathit{expr}\,.\,\mathit{identifier}]\!]^{\mathrm{M}}_{\mathrm{ADDR}} = \langle a, M'\rangle}{[\![\mathit{expr}\,.\,\mathit{identifier}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle M'[a], \mathrm{M}'\rangle} \tag{EX11}$$

$$\frac{[\![\mathit{expr}_1]\!]^{\mathrm{M}_0}_{\mathrm{EXPR}} = \langle n_1, \mathrm{M}_1\rangle \ \ldots\ [\![\mathit{expr}_m]\!]^{\mathrm{M}_{m-1}}_{\mathrm{EXPR}} = \langle n_m, \mathrm{M}_m\rangle}{[\![\mathit{identifier}\texttt{(}\mathit{expr}_1,\ldots,\mathit{expr}_m\texttt{)}]\!]^{\mathrm{M}_0}_{\mathrm{EXPR}} = \langle \mathit{identifier}(n_1,\ldots,n_m), \mathrm{M}_m\rangle} \tag{EX12}$$

$$\frac{[\![\mathit{stmt}_1]\!]^{\mathrm{M}_0}_{\mathrm{STMT}} = \langle n_1, \mathrm{M}_1\rangle \ \ldots\ [\![\mathit{stmt}_m]\!]^{\mathrm{M}_{m-1}}_{\mathrm{STMT}} = \langle n_m, \mathrm{M}_m\rangle}{[\![\texttt{\{}\mathit{stmt}_1,\ldots,\mathit{stmt}_m\texttt{\}}]\!]^{\mathrm{M}_0}_{\mathrm{EXPR}} = \langle n_m, \mathrm{M}_m\rangle} \tag{EX13}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle}{[\![\texttt{(}\mathit{expr}\texttt{)}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle} \tag{EX14}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle \quad [\![\mathit{type}]\!]_{\mathrm{ISTYPE}} \neq \textbf{char}}{[\![\texttt{(}\mathit{expr}\,\texttt{:}\,\mathit{type}\texttt{)}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle} \tag{EX15}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle \quad [\![\mathit{type}]\!]_{\mathrm{ISTYPE}} = \textbf{char}}{[\![\texttt{(}\mathit{expr}\,\texttt{:}\,\mathit{type}\texttt{)}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n \bmod 256, \mathrm{M}'\rangle} \tag{EX16}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle}{[\![\mathit{expr}\ \texttt{where \{}\mathit{decls}\texttt{\}}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle n, \mathrm{M}'\rangle} \tag{EX17}$$

Statements.

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

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

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \textbf{true}, \mathrm{M}'\rangle \quad [\![\mathit{stmt}_1]\!]^{\mathrm{M}'}_{\mathrm{STMT}} = \langle \mathrm{undef}, \mathrm{M}''\rangle}{[\![\texttt{if}\ \mathit{expr}\ \texttt{then}\ \mathit{stmt}_1\ \texttt{else}\ \mathit{stmt}_2]\!]_{\mathrm{STMT}} = \langle \mathrm{undef}, \mathrm{M}''\rangle} \tag{ST3}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathbf{false}, \mathrm{M}' \rangle \quad [\![\mathit{stmt}_2]\!]^{\mathrm{M}'}_{\mathrm{EXPR}} = \langle \mathrm{undef}, \mathrm{M}'' \rangle}{[\![\texttt{if}\ \mathit{expr}\ \texttt{then}\ \mathit{stmt}_1\ \texttt{else}\ \mathit{stmt}_2]\!]_{\mathrm{STMT}} = \langle \mathrm{undef}, \mathrm{M}'' \rangle} \tag{ST4}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathbf{true}, \mathrm{M}' \rangle \quad [\![\mathit{stmt}]\!]^{\mathrm{M}'}_{\mathrm{STMT}} = \langle \mathrm{undef}, \mathrm{M}'' \rangle}{[\![\texttt{while}\ \mathit{expr}\ \texttt{do}\ \mathit{stmt}]\!]^{\mathrm{M}}_{\mathrm{STMT}} = [\![\texttt{while}\ \mathit{expr}\ \texttt{do}\ \mathit{stmt}]\!]^{\mathrm{M}''}_{\mathrm{STMT}}} \tag{ST5}$$

$$\frac{[\![\mathit{expr}]\!]^{\mathrm{M}}_{\mathrm{EXPR}} = \langle \mathbf{false}, \mathrm{M}' \rangle}{[\![\texttt{while}\ \mathit{expr}\ \texttt{do}\ \mathit{stmt}]\!]^{\mathrm{M}}_{\mathrm{STMT}} = \langle \mathrm{undef}, \mathrm{M}' \rangle} \tag{ST6}$$