$$
\begin{array}{lll}
\langle \textit{basetype} \rangle & ::= & \text{IntT} \\
 & | & \text{BoolT} \\[1ex]
\langle \textit{type} \rangle & ::= & \langle \textit{basetype} \rangle \\
 & | & \text{PointerT } \langle \textit{type} \rangle \\
 & | & \text{TupleT } \langle \textit{basetype} \rangle * \\[1ex]
\langle \textit{bool} \rangle & ::= & \top \\
 & | & \bot \\[1ex]
\langle \textit{order} \rangle & ::= & \text{Parallel} \\
 & | & \text{Sequential} \\[1ex]
\langle \textit{function} \rangle & ::= & \text{Function } \langle \textit{type} \rangle\ \langle \textit{expr} \rangle \\[1ex]
\langle \textit{assumption} \rangle & ::= & \text{InLet } \langle \textit{expr} \rangle \\
 & | & \text{InLoop } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{InFunc } \langle \textit{function} \rangle \\
 & | & \text{InSwitch } \langle \mathbb{N} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{InIf } \langle \textit{bool} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{AfterWrite } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle
\end{array}
$$

$$
\begin{array}{lll}
\langle \textit{expr} \rangle & ::= & \text{Arg } \langle \textit{type} \rangle \\
 & | & \text{Int } \langle \mathbb{N} \rangle \\
 & | & \text{Bool } \langle \textit{bool} \rangle \\
 & | & \text{Empty} \\
 & | & \text{Add } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{Sub } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{Mul } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{LessThan } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{And } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{Or } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{Write } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{PtrAdd } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{Not } \langle \textit{expr} \rangle \\
 & | & \text{Print } \langle \textit{expr} \rangle \\
 & | & \text{Read } \langle \textit{expr} \rangle \\
 & | & \text{Get } \langle \textit{expr} \rangle\ \langle \mathbb{N} \rangle \\
 & | & \text{Alloc } \langle \textit{expr} \rangle\ \langle \textit{type} \rangle \\
 & | & \text{Call } \langle \textit{function} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{Single } \langle \textit{expr} \rangle \\
 & | & \text{Concat } \langle \textit{order} \rangle\ \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{Switch } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle * \\
 & | & \text{If } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{Let } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{DoWhile } \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
 & | & \text{Assume } \langle \textit{assumption} \rangle\ \langle \textit{expr} \rangle
\end{array}
$$

Figure 1: expr abstract syntax.

Big-step operational semantics

$\langle e, \alpha, \sigma \rangle \Downarrow \langle v, \sigma' \rangle$ means: with argument α and state σ, e evaluates to v and the resulting state is σ'.

A state is pair (M, L), containing memory and a print log.

$$\frac{}{\langle \text{Num } n, \alpha, \sigma \rangle \Downarrow \langle n, \sigma \rangle} \quad \text{(E-NUM)}$$

$$\frac{}{\langle \text{Bool } b, \alpha, \sigma \rangle \Downarrow \langle b, \sigma \rangle} \quad \text{(E-BOOL)}$$

$$\frac{\langle e_1, \alpha, \sigma \rangle \Downarrow \langle v_1, \sigma' \rangle \qquad \langle e_2, \alpha, \sigma' \rangle \Downarrow \langle v_2, \sigma'' \rangle}{\langle \text{Add } e_1\ e_2, \alpha, \sigma \rangle \Downarrow \langle v_1 + v_2, \sigma'' \rangle} \quad \text{(E-ADD)}$$

$$\frac{\langle e_c, \alpha, \sigma \rangle \Downarrow \langle \top, \sigma' \rangle \qquad \langle e_t, \alpha, \sigma' \rangle \Downarrow \langle v_t, \sigma'' \rangle}{\langle \text{If } e_c\ e_t\ e_e, \alpha, \sigma \rangle \Downarrow \langle v_t, \sigma'' \rangle} \quad \text{(E-IFTRUE)}$$

$$\frac{\langle e_c, \alpha, \sigma \rangle \Downarrow \langle \bot, \sigma' \rangle \qquad \langle e_e, \alpha, \sigma' \rangle \Downarrow \langle v_e, \sigma'' \rangle}{\langle \text{If } e_c\ e_t\ e_e, \alpha, \sigma \rangle \Downarrow \langle v_e, \sigma'' \rangle} \quad \text{(E-IFFALSE)}$$

$$\frac{\langle e_{pred}, \alpha, \sigma \rangle \Downarrow \langle i, \sigma' \rangle \qquad \langle e_i, \alpha, \sigma' \rangle \Downarrow \langle v, \sigma'' \rangle}{\langle \text{Switch } e_{pred}\ (e_1, \ldots, e_n), \alpha, \sigma \rangle \Downarrow \langle v, \sigma'' \rangle} \quad \text{(E-SWITCH)}$$

$$\frac{\langle e_{in}, \alpha, \sigma \rangle \Downarrow \langle v_{in}, \sigma' \rangle \qquad \langle e_{out}, v_{in}, \sigma' \rangle \Downarrow \langle v_{out}, \sigma'' \rangle}{\langle \text{Let } e_{in}\ e_{out}, \alpha, \sigma \rangle \Downarrow \langle v_{out}, \sigma'' \rangle} \quad \text{(E-LET)}$$

$$\frac{\langle e, \alpha, \sigma \rangle \Downarrow \langle v, (M, L) \rangle}{\langle \text{Print } e, \alpha, \sigma \rangle \Downarrow \langle [], (M, L \mathbin{++} v) \rangle} \quad \text{(E-PRINT)}$$

$$\frac{\langle e, \alpha, \sigma \rangle \Downarrow \langle v, (M, L) \rangle}{\langle \text{Read } e, \alpha, \sigma \rangle \Downarrow \langle M[v], (M, L) \rangle} \quad \text{(E-READ)}$$

$$\frac{\langle e, \alpha, \sigma \rangle \Downarrow \langle n, (M, L) \rangle \qquad (M', p) = \text{malloc}(M, n * \text{sizeof}(\tau))}{\langle \text{Alloc } e\ \tau, \alpha, \sigma \rangle \Downarrow \langle p, (M', L) \rangle} \quad \text{(E-ALLOC)}$$

$$\frac{\langle e_p, \alpha, \sigma \rangle \Downarrow \langle v_p, \sigma' \rangle \qquad \langle e_d, \alpha, \sigma' \rangle \Downarrow \langle v_d, (M, L) \rangle}{\langle \text{Write } e_p\ e_d, \alpha, \sigma \rangle \Downarrow \langle [], (M[v_p \to v_d], L) \rangle} \quad \text{(E-WRITE)}$$

$$\frac{\langle e, \alpha, \sigma \rangle \Downarrow \langle v, \sigma' \rangle}{\langle \text{Single } e, \alpha, \sigma \rangle \Downarrow \langle [v], \sigma' \rangle} \quad \text{(E-SINGLE)}$$

$$\frac{\langle e_1, \alpha, \sigma \rangle \Downarrow \langle v_1, \sigma' \rangle \qquad \langle e_2, \alpha, \sigma' \rangle \Downarrow \langle v_2, \sigma'' \rangle}{\langle \text{Concat Sequential } e_1\ e_2, \alpha, \sigma \rangle \Downarrow \langle v_1 \mathbin{++} v_2, \sigma'' \rangle} \quad \text{(E-CONCATSEQ)}$$

$$\frac{\langle e_{in}, \alpha, \sigma \rangle \Downarrow \langle \alpha', \sigma' \rangle \qquad \langle e_{pred\_out}, \alpha', \sigma' \rangle \Downarrow \langle [\bot, v], \sigma'' \rangle}{\langle \text{DoWhile } e_{in}\ e_{out}, \alpha, \sigma \rangle \Downarrow \langle v, \sigma'' \rangle} \quad \text{(E-DOWHILEFALSE)}$$

$$\frac{\langle e_{in}, \alpha, \sigma \rangle \Downarrow \langle \alpha', \sigma' \rangle \qquad \langle e_{pred\_out}, \alpha', \sigma' \rangle \Downarrow \langle [\top, \alpha''], \sigma'' \rangle \qquad \langle \text{DoWhile } e_{in}\ e_{pred\_out}, \alpha'', \sigma'' \rangle \Downarrow \langle v, \sigma''' \rangle}{\langle \text{DoWhile } e_{in}\ e_{pred\_out}, \alpha, \sigma \rangle \Downarrow \langle v, \sigma''' \rangle}$$

(E-DOWHILETRUE)

$$\frac{\langle e, \alpha, \sigma \rangle \Downarrow \langle v, \sigma' \rangle \qquad \exists\ \alpha_2, \sigma_2, \sigma_3.\ \langle e_{in}, \alpha_2, \sigma_2 \rangle \Downarrow \langle \alpha, \sigma_3 \rangle}{\langle \text{Assume (InLet } e_{in})\ e, \alpha, \sigma \rangle \Downarrow \langle v, \sigma' \rangle} \quad \text{(E-ASSUMEINLET)}$$

$$\frac{\langle e, \alpha, \sigma \rangle \Downarrow \langle v, \sigma' \rangle \qquad \exists\ \alpha_2, v_2, \sigma_2.\ \langle \text{Write } e_p\ e_d, \alpha_2, \sigma_2 \rangle \Downarrow \langle v_2, \sigma_3 \rangle}{\langle \text{Assume (AfterWrite } e_p\ e_d)\ e, \alpha, \sigma \rangle \Downarrow \langle v, \sigma' \rangle} \quad \text{(E-ASSUMEAFTERWRITE)}$$

Denotational semantics

$$\frac{}{[\![\text{Num } n]\!]\,(\alpha, \sigma) \triangleq (n, \sigma)} \quad \text{(E-Num)}$$

$$\frac{}{[\![\text{Bool } b]\!]\,(\alpha, \sigma) \triangleq (b, \sigma)} \quad \text{(E-Bool)}$$

$$\frac{[\![e_1]\!]\,(\alpha, \sigma) = (v_1, \sigma') \qquad [\![e_2]\!]\,(\alpha, \sigma') = (v_2, \sigma'')}{[\![\text{Add } e_1\ e_2]\!]\,(\alpha, \sigma) \triangleq (v_1 + v_2, \sigma'')} \quad \text{(E-Add)}$$

$$\frac{[\![e_c]\!]\,(\alpha, \sigma) = (\top, \sigma') \qquad [\![e_t]\!]\,(\alpha, \sigma') = (v_t, \sigma'')}{[\![\text{If } e_c\ e_t\ e_e]\!]\,(\alpha, \sigma) \triangleq (v_t, \sigma'')} \quad \text{(E-IfTrue)}$$

$$\frac{[\![e_c]\!]\,(\alpha, \sigma) = (\bot, \sigma') \qquad [\![e_e]\!]\,(\alpha, \sigma') = (v_e, \sigma'')}{[\![\text{If } e_c\ e_t\ e_e]\!]\,(\alpha, \sigma) \triangleq (v_e, \sigma'')} \quad \text{(E-IfFalse)}$$

$$\frac{[\![e_{pred}]\!]\,(\alpha, \sigma) = (i, \sigma') \qquad [\![e_i]\!]\,(\alpha, \sigma') = (v, \sigma'')}{[\![\text{Switch } e_{pred}\ (e_1, \ldots, e_n)]\!]\,(\alpha, \sigma) \triangleq (v, \sigma'')} \quad \text{(E-Switch)}$$

$$\frac{[\![e_{in}]\!]\,(\alpha, \sigma) = (v_{in}, \sigma') \qquad [\![e_{out}]\!]\,(v_{in}, \sigma') = (v_{out}, \sigma'')}{[\![\text{Let } e_{in}\ e_{out}]\!]\,(\alpha, \sigma) \triangleq (v_{out}, \sigma'')} \quad \text{(E-Let)}$$

$$\frac{[\![e]\!]\,(\alpha, \sigma) = (v, (M, L))}{[\![\text{Print } e]\!]\,(\alpha, \sigma) \triangleq ([], (M, L \mathbin{+\!+} v))} \quad \text{(E-Print)}$$

$$\frac{[\![e]\!]\,(\alpha, \sigma) = (v, (M, L))}{[\![\text{Read } e]\!]\,(\alpha, \sigma) \triangleq (M[v], (M, L))} \quad \text{(E-Read)}$$

$$\frac{[\![e]\!]\,(\alpha, \sigma) = (n, (M, L)) \qquad (M', p) = \text{malloc}(M, n * \text{sizeof}(\tau))}{[\![\text{Alloc } e\ \tau]\!]\,(\alpha, \sigma) \triangleq (p, (M', L))} \quad \text{(E-Alloc)}$$

$$\frac{[\![e_p]\!]\,(\alpha, \sigma) = (v_p, \sigma') \qquad [\![e_d]\!]\,(\alpha, \sigma') = (v_d, (M, L))}{[\![\text{Write } e_p\ e_d]\!]\,(\alpha, \sigma) \triangleq ([], (M[v_p \to v_d], L))} \quad \text{(E-Write)}$$

$$\frac{[\![e]\!]\,(\alpha, \sigma) = (v, \sigma')}{[\![\text{Single } e]\!]\,(\alpha, \sigma) \triangleq ([v], \sigma')} \quad \text{(E-Single)}$$

$$\frac{[\![e_1]\!]\,(\alpha, \sigma) = (v_1, \sigma') \qquad [\![e_2]\!]\,(\alpha, \sigma') = (v_2, \sigma'')}{[\![\text{Concat Sequential } e_1\ e_2]\!]\,(\alpha, \sigma) \triangleq (v_1 \mathbin{+\!+} v_2, \sigma'')} \quad \text{(E-ConcatSeq)}$$

$$\frac{[\![e_{in}]\!]\,(\alpha, \sigma) = (\alpha', \sigma') \qquad [\![e_{pred\_out}]\!]\,(\alpha', \sigma') = ([\bot, v], \sigma'')}{[\![\text{DoWhile } e_{in}\ e_{out}]\!]\,(\alpha, \sigma) \triangleq (v, \sigma'')} \quad \text{(E-DoWhileFalse)}$$

$$\frac{[\![e_{in}]\!]\,(\alpha, \sigma) = (\alpha', \sigma') \qquad [\![e_{pred\_out}]\!]\,(\alpha', \sigma') = ([\top, \alpha''], \sigma'') \qquad [\![\text{DoWhile } e_{in}\ e_{pred\_out}]\!]\,(\alpha'', \sigma'') = (v, \sigma''')}{[\![\text{DoWhile } e_{in}\ e_{pred\_out}]\!]\,(\alpha, \sigma) \triangleq (v, \sigma''')}$$

(E-DoWhileTrue)

$$\frac{[\![e]\!]\,(\alpha, \sigma) = (v, \sigma') \qquad \exists\, \alpha_2, \sigma_2, \sigma_3.\ [\![e_{in}]\!]\,(\alpha_2, \sigma_2) = (\alpha, \sigma_3)}{[\![\text{Assume (InLet } e_{in}) \ e]\!]\,(\alpha, \sigma) \triangleq (v, \sigma')} \quad \text{(E-AssumeInLet)}$$

$$\frac{[\![e]\!]\,(\alpha, \sigma) = (v, \sigma') \qquad \exists\, \alpha_2, v_2, \sigma_2.\ [\![\text{Write } e_p\ e_d]\!]\,(\alpha_2, \sigma_2) = (v_2, \sigma_3)}{[\![\text{Assume (AfterWrite } e_p\ e_d) \ e]\!]\,(\alpha, \sigma) \triangleq (v, \sigma')} \quad \text{(E-AssumeAfterWrite)}$$