$$
\begin{aligned}
\langle \textit{basetype} \rangle ::=\ & \text{IntT} \\
\mid\ & \text{BoolT}
\end{aligned}
$$

$$
\begin{aligned}
\langle \textit{type} \rangle ::=\ & \langle \textit{basetype} \rangle \\
\mid\ & \text{PointerT}\ \langle \textit{type} \rangle \\
\mid\ & \text{TupleT}\ \langle \textit{basetype} \rangle^{*}
\end{aligned}
$$

$$
\begin{aligned}
\langle \textit{bool} \rangle ::=\ & \top \\
\mid\ & \bot
\end{aligned}
$$

$$
\begin{aligned}
\langle \textit{order} \rangle ::=\ & \text{Parallel} \\
\mid\ & \text{Sequential}
\end{aligned}
$$

$$
\langle \textit{function} \rangle ::= \text{Function}\ \langle \textit{type} \rangle\ \langle \textit{expr} \rangle
$$

$$
\begin{aligned}
\langle \textit{assumption} \rangle ::=\ & \text{InLet}\ \langle \textit{expr} \rangle \\
\mid\ & \text{InLoop}\ \langle \textit{expr} \rangle\ \langle \textit{expr} \rangle \\
\mid\ & \text{InFunc}\ \langle \textit{function} \rangle \\
\mid\ & \text{InSwitch}\ \langle \mathbb{N} \rangle\ \langle \textit{expr} \rangle \\
\mid\ & \text{InIf}\ \langle \textit{bool} \rangle\ \langle \textit{expr} \rangle
\end{aligned}
$$

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

Figure 1: expr abstract syntax.

$\langle e, \alpha, \sigma \rangle \Downarrow \langle v, \sigma' \rangle$ 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.

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

$$\frac{\langle c, \alpha, \sigma \rangle \Downarrow \langle \top, \sigma' \rangle \qquad \langle t, \alpha, \sigma' \rangle \Downarrow \langle v, \sigma'' \rangle}{\langle \text{If } c \; t \; e, \sigma \rangle \;\Downarrow\; \langle v, \sigma'' \rangle} \quad \text{(E-IFTRUE)}$$

$$\frac{\langle c, \alpha, \sigma \rangle \Downarrow \langle \bot, \sigma' \rangle \qquad \langle e, \alpha, \sigma' \rangle \Downarrow \langle v, \sigma'' \rangle}{\langle \text{If } c \; t \; e, \sigma \rangle \;\Downarrow\; \langle v, \sigma'' \rangle} \quad \text{(E-IFFALSE)}$$

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

$$\frac{\langle e, \alpha, \sigma \rangle \Downarrow \langle v, \sigma' \rangle}{\langle \text{Assume } e \; a, \alpha, \sigma \rangle \;\Downarrow\; \langle v, \sigma' \rangle} \quad \text{(E-ASSUME)}$$

$$\frac{\langle i, \alpha, \sigma \rangle \Downarrow \langle \alpha', \sigma' \rangle \qquad \langle o, \alpha', \sigma' \rangle \Downarrow \langle v, \sigma'' \rangle}{\langle \text{Let } i \; o, \alpha, \sigma \rangle \;\Downarrow\; \langle v, \sigma'' \rangle} \quad \text{(E-LET)}$$

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

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

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

$$\frac{\langle p, \alpha, \sigma \rangle \Downarrow \langle v_p, \sigma' \rangle \qquad \langle e, \alpha, \sigma' \rangle \Downarrow \langle v_e, (M, L) \rangle}{\langle \text{Store } p \; e, \alpha, \sigma \rangle \;\Downarrow\; \langle [], (M[v_p \to v_e], L) \rangle} \quad \text{(E-STORE)}$$

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

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

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

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

(E-DOWHILETRUE)