$\langle \textit{basetype} \rangle$ ::= IntT
  | BoolT

$\langle \textit{type} \rangle$ ::= $\langle \textit{basetype} \rangle$
  | PointerT $\langle \textit{type} \rangle$
  | TupleT $\langle \textit{basetype} \rangle$*

$\langle \textit{bool} \rangle$ ::= True
  | False

$\langle \textit{order} \rangle$ ::= Parallel
  | Sequential

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

$\langle \textit{assumption} \rangle$ ::= InLet $\langle \textit{expr} \rangle$
  | InLoop $\langle \textit{expr} \rangle$ $\langle \textit{expr} \rangle$
  | InFunc $\langle \textit{function} \rangle$
  | InSwitch $\langle \mathbb{N} \rangle$ $\langle \textit{expr} \rangle$
  | InIf $\langle \textit{bool} \rangle$ $\langle \textit{expr} \rangle$

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

Figure 1: expr abstract syntax.

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\langle e_{in}, \alpha, \sigma \rangle \Downarrow \langle \alpha', \sigma' \rangle \qquad \langle e_{pred\_out}, \alpha', \sigma' \rangle \Downarrow \langle [\bot, v], \sigma'' \rangle}{\langle \text{DoWhile } e_{in} \; e_{pred\_out}, \alpha, \sigma \rangle \; \Downarrow \; \langle v, \sigma'' \rangle} \quad (\text{E-D}\textsc{o}\text{W}\textsc{hile}\text{F}\textsc{alse})$$

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

(E-DoWhileTrue)