The Zwei Programming Language: Big step semantics

LAMP

EPFL

Name	a, b		
Integer	n		
Unary operator	*unop*	$::= - \mid !$	
Binary operator	*binop*	$::= + \mid - \mid * \mid / \mid \% \mid == \mid != \mid < \mid \leq \mid > \mid \geq \mid \&\&$	
Term	t, t'	$::= n$	integer literal
		$\mid a$	variable, current instance
		$\mid$ *unop* t	unary operation
		$\mid t$ *binop* t'	binary operation
		$\mid$ `readInt`	read integer
		$\mid$ `readChar`	read character
		$\mid$ `if` $(t)\ t'$ `else` t''	conditional
		$\mid \{\overline{S}\ t\}$	block
Statement	S	$::=$ `while` $(t)\ \{\overline{S}\}$	loop
		$\mid$ `var` $a : T = t$	local variable
		$\mid$ `set` $a = t$	variable assignment
		$\mid$ `do` t	instruction
		$\mid$ `printInt`(t)	print integer
		$\mid$ `printChar`(t)	print character
Values	v	$::= \mathbb{I}(n)$	integers
Frame	σ	$::= \epsilon$	empty frame
		$\mid a \mapsto v, \sigma$	
Stack	Σ	$::= \epsilon$	empty stack
		$\mid \sigma; \Sigma$	

Fig. 1. Imperative fragment of Zwei

$$
\begin{array}{rl}
[\![-]\!](n) \stackrel{\text{def}}{=} -n & \qquad [\![+]\!](n_1, n_2) \stackrel{\text{def}}{=} n_1 + n_2 \\
[\![!]\!](n) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases} & \qquad [\![-]\!](n_1, n_2) \stackrel{\text{def}}{=} n_1 - n_2 \\
[\![\&\&]\!](n_1, n_2) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n_1 * n_2 \neq 0 \\ 0 & \text{otherwise} \end{cases} & \qquad [\![*]\!](n_1, n_2) \stackrel{\text{def}}{=} n_1 * n_2 \\
 & \qquad [\![/]\!](n_1, n_2) \stackrel{\text{def}}{=} n_1 / n_2 \\
 & \qquad [\![\%]\!](n_1, n_2) \stackrel{\text{def}}{=} n_1 \; mod \; n_2
\end{array}
$$

Fig. 2. Arithmetic and logical operations on integers

$$
\begin{array}{rl}
[\![<]\!](n_1, n_2) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n_1 < n_2 \\ 0 & \text{otherwise} \end{cases} & \\
[\![\leq]\!](n_1, n_2) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n_1 \leq n_2 \\ 0 & \text{otherwise} \end{cases} & \qquad [\![==]\!](n_1, n_2) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n_1 = n_2 \\ 0 & \text{otherwise} \end{cases} \\
[\![>]\!](n_1, n_2) \stackrel{\text{def}}{=} [\![<]\!](n_2, n_1) & \qquad [\![!=]\!](n_1, n_2) \stackrel{\text{def}}{=} [\![!]\!]([\![==]\!](n_1, n_2)) \\
[\![\geq]\!](n_1, n_2) \stackrel{\text{def}}{=} [\![\leq]\!](n_2, n_1) &
\end{array}
$$

Fig. 3. Comparisons of integer values

$$
(\textsc{find-hd}) \; \frac{}{a \mapsto v, \sigma; \Sigma \; \vdash \; a \Rightarrow v} \qquad (\textsc{find-tl}) \; \frac{\sigma; \Sigma \; \vdash \; a \Rightarrow v}{b \mapsto v', \sigma; \Sigma \; \vdash \; a \Rightarrow v} \; b \neq a
$$

$$
(\textsc{find-nxt}) \; \frac{\Sigma \; \vdash \; a \Rightarrow v}{\epsilon; \Sigma \; \vdash \; a \Rightarrow v}
$$

Getting a value

$$
(\textsc{add-hd}) \; \frac{\sigma; \Sigma \; \vdash \; a \mapsto v \Rightarrow \sigma'; \Sigma'}{b \mapsto v', \sigma; \Sigma \; \vdash \; a \mapsto v \Rightarrow b \mapsto v', \sigma'; \Sigma'} \; b \neq a
$$

$$
(\textsc{add-tl}) \; \frac{}{\epsilon; \Sigma \; \vdash \; a \mapsto v \Rightarrow a \mapsto v, \epsilon; \Sigma}
$$

Adding a binding

$$
(\textsc{upd-hd}) \; \frac{}{a \mapsto v, \sigma; \Sigma \; \vdash \; a/v' \Rightarrow a \mapsto v', \sigma; \Sigma}
$$

$$
(\textsc{upd-tl}) \; \frac{\sigma; \Sigma \; \vdash \; a/v' \Rightarrow \sigma'; \Sigma'}{b \mapsto v, \sigma; \Sigma \; \vdash \; a/v' \Rightarrow b \mapsto v, \sigma'; \Sigma'} \; b \neq a
$$

$$
(\textsc{upd-nxt}) \; \frac{\Sigma \; \vdash \; a/v' \Rightarrow \Sigma'}{\epsilon; \Sigma \; \vdash \; a/v' \Rightarrow \epsilon; \Sigma'}
$$

Updating a binding

Fig. 4. Operations on stacks

$$(\text{EVAL-INT})\ \frac{}{\Sigma\ \vdash\ n \Downarrow \mathbb{I}(n), \langle \Sigma \rangle} \qquad (\text{EVAL-VAR})\ \frac{\Sigma\ \vdash\ a \Rightarrow v}{\Sigma\ \vdash\ a \Downarrow v, \langle \Sigma \rangle}$$

$$(\text{EVAL-UNOP})\ \frac{\Sigma\ \vdash\ t \Downarrow \mathbb{I}(n), \langle \Sigma' \rangle}{\Sigma\ \vdash\ \mathit{unop}\ t \Downarrow \mathbb{I}([\![\mathit{unop}]\!](n)), \langle \Sigma' \rangle}$$

$$(\text{EVAL-BINOP-INT})\ \frac{\Sigma\ \vdash\ t_1 \Downarrow \mathbb{I}(n_1), \langle \Sigma' \rangle \qquad \mathit{binop} \neq \&\& \qquad \Sigma'\ \vdash\ t_2 \Downarrow \mathbb{I}(n_2), \langle \Sigma'' \rangle}{\Sigma\ \vdash\ t_1\ \mathit{binop}\ t_2 \Downarrow \mathbb{I}([\![\mathit{binop}]\!](n_1, n_2)), \langle \Sigma'' \rangle}$$

$$(\text{EVAL-AND-FALSE})\ \frac{\Sigma\ \vdash\ t_1 \Downarrow \mathbb{I}(0), \langle \Sigma' \rangle}{\Sigma\ \vdash\ t_1\ \&\&\ t_2 \Downarrow \mathbb{I}(0), \langle \Sigma' \rangle}$$

$$(\text{EVAL-AND-TRUE})\ \frac{\Sigma\ \vdash\ t_1 \Downarrow \mathbb{I}(n_1), \langle \Sigma' \rangle \qquad n_1 \neq 0 \qquad \Sigma'\ \vdash\ t_2 \Downarrow \mathbb{I}(n_2), \langle \Sigma'' \rangle}{\Sigma\ \vdash\ t_1\ \&\&\ t_2 \Downarrow \mathbb{I}([\![\&\&]\!](n_1, n_2)), \langle \Sigma'' \rangle}$$

$$(\text{EVAL-READINT})\ \frac{\text{an integer } n \text{ is read on standard input}}{\Sigma\ \vdash\ \texttt{readInt} \Downarrow \mathbb{I}(n), \langle \Sigma \rangle}$$

$$(\text{EVAL-READCHAR-SOME})\ \frac{\text{a character of unicode } n \text{ is read on standard input}}{\Sigma\ \vdash\ \texttt{readChar} \Downarrow \mathbb{I}(n), \langle \Sigma \rangle}$$

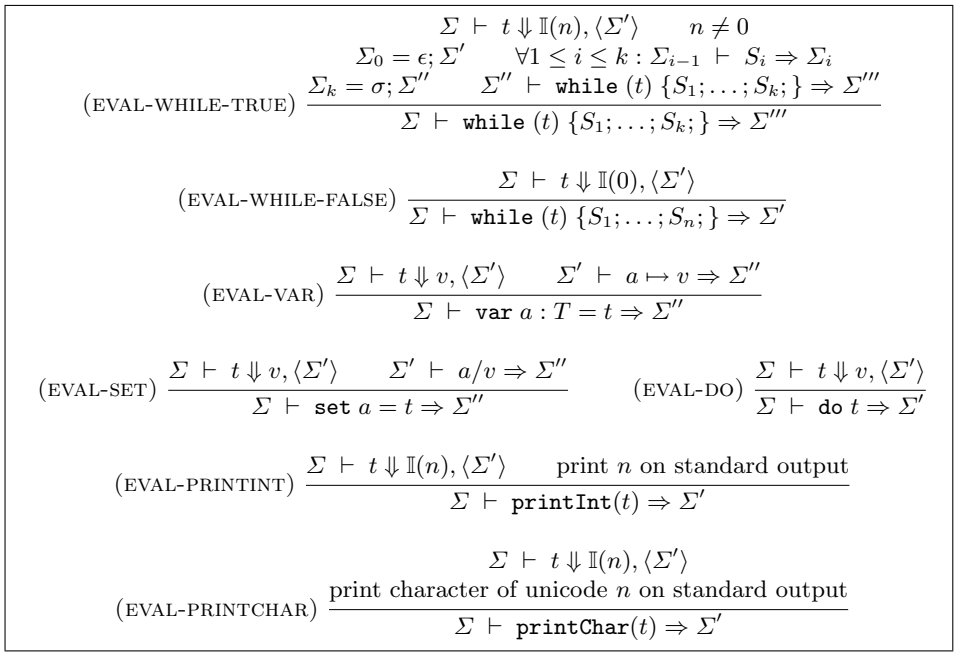
$$(\text{EVAL-READCHAR-NONE})\ \frac{\text{standard input is closed}}{\Sigma\ \vdash\ \texttt{readChar} \Downarrow \mathbb{I}(-1), \langle \Sigma \rangle}$$

$$(\text{EVAL-IF-THEN})\ \frac{\Sigma\ \vdash\ t \Downarrow \mathbb{I}(n), \langle \Sigma' \rangle \qquad n \neq 0 \qquad \Sigma'\ \vdash\ t_1 \Downarrow v_1, \langle \Sigma'' \rangle}{\Sigma\ \vdash\ \texttt{if}\ (t)\ t_1\ \texttt{else}\ t_2 \Downarrow v_1, \langle \Sigma'' \rangle}$$

$$(\text{EVAL-IF-ELSE})\ \frac{\Sigma\ \vdash\ t \Downarrow \mathbb{I}(0), \langle \Sigma' \rangle \qquad \Sigma'\ \vdash\ t_2 \Downarrow v_2, \langle \Sigma'' \rangle}{\Sigma\ \vdash\ \texttt{if}\ (t)\ t_1\ \texttt{else}\ t_2 \Downarrow v_2, \langle \Sigma'' \rangle}$$

$$(\text{EVAL-BLOCK})\ \frac{\Sigma_0 = \epsilon; \Sigma \qquad \forall 1 \leq i \leq k : \Sigma_{i-1}\ \vdash\ S_i \Rightarrow \Sigma_i \qquad \Sigma_k\ \vdash\ t \Downarrow v, \langle \sigma; \Sigma' \rangle}{\Sigma\ \vdash\ \{S_1; \ldots; S_k;\ t\} \Downarrow v, \langle \Sigma' \rangle}$$

Fig. 5. Evaluation of expressions



$$\text{(EVAL-WHILE-TRUE)}\ \frac{\Sigma \vdash t \Downarrow \mathbb{I}(n), \langle \Sigma' \rangle \qquad n \neq 0 \qquad \Sigma_0 = \epsilon; \Sigma' \qquad \forall 1 \leq i \leq k : \Sigma_{i-1} \vdash S_i \Rightarrow \Sigma_i \qquad \Sigma_k = \sigma; \Sigma'' \qquad \Sigma'' \vdash \texttt{while}\ (t)\ \{S_1; \ldots; S_k;\} \Rightarrow \Sigma'''}{\Sigma \vdash \texttt{while}\ (t)\ \{S_1; \ldots; S_k;\} \Rightarrow \Sigma'''}$$

$$\text{(EVAL-WHILE-FALSE)}\ \frac{\Sigma \vdash t \Downarrow \mathbb{I}(0), \langle \Sigma' \rangle}{\Sigma \vdash \texttt{while}\ (t)\ \{S_1; \ldots; S_n;\} \Rightarrow \Sigma'}$$

$$\text{(EVAL-VAR)}\ \frac{\Sigma \vdash t \Downarrow v, \langle \Sigma' \rangle \qquad \Sigma' \vdash a \mapsto v \Rightarrow \Sigma''}{\Sigma \vdash \texttt{var}\ a : T = t \Rightarrow \Sigma''}$$

$$\text{(EVAL-SET)}\ \frac{\Sigma \vdash t \Downarrow v, \langle \Sigma' \rangle \qquad \Sigma' \vdash a/v \Rightarrow \Sigma''}{\Sigma \vdash \texttt{set}\ a = t \Rightarrow \Sigma''} \qquad \text{(EVAL-DO)}\ \frac{\Sigma \vdash t \Downarrow v, \langle \Sigma' \rangle}{\Sigma \vdash \texttt{do}\ t \Rightarrow \Sigma'}$$

$$\text{(EVAL-PRINTINT)}\ \frac{\Sigma \vdash t \Downarrow \mathbb{I}(n), \langle \Sigma' \rangle \qquad \text{print } n \text{ on standard output}}{\Sigma \vdash \texttt{printInt}(t) \Rightarrow \Sigma'}$$

$$\text{(EVAL-PRINTCHAR)}\ \frac{\Sigma \vdash t \Downarrow \mathbb{I}(n), \langle \Sigma' \rangle \qquad \text{print character of unicode } n \text{ on standard output}}{\Sigma \vdash \texttt{printChar}(t) \Rightarrow \Sigma'}$$

Fig. 6. Evaluation of statements

Class declaration	D	$::= \texttt{class}\ a\ \texttt{extends}\ s\ \{\overline{d}\}$	
Super class	s	$::= a \mid \texttt{none}$	
Member declaration	d	$::= \texttt{val}\ a : T$	field declaration
		$\mid \texttt{def}\ a(\overline{a} : \overline{T}) : T = t$	method definition
Term	t, u	$::= \texttt{new}\ a(\overline{t})$	instance creation
		$\mid t.a$	field selection
		$\mid t.a(\overline{t})$	method call
		$\mid \texttt{null}$	null reference
		$\mid \ldots$	as before
Values	v	$::= \mathbb{O}(id, \sigma, \overline{m})$	object of identity $id \in \mathbb{N}$ (σ represents the fields)
		$\mid \bot$	null value
		$\mid \ldots$	as before
Methods	m	$::= \texttt{def}\ a(\overline{a} : \overline{T}) : T = t$	

Fig. 7. Adding Object layer to Zwei

$$
\begin{aligned}
[\![==]\!](\mathbb{O}(id_1,\sigma_1,\overline{m}_1),\mathbb{O}(id_2,\sigma_2,\overline{m}_2)) &\stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } id_1 = id_2 \\ 0 & \text{otherwise} \end{cases} \\
[\![==]\!](\bot,\mathbb{O}(id,\sigma,\overline{m})) &\stackrel{\text{def}}{=} 0 \\
[\![==]\!](\mathbb{O}(id,\sigma,\overline{m}),\bot) &\stackrel{\text{def}}{=} 0 \\
[\![==]\!](\bot,\bot) &\stackrel{\text{def}}{=} 1 \\
[\![==]\!](\mathbb{I}(n_1),\mathbb{I}(n_2)) &\stackrel{\text{def}}{=} [\![==]\!](n_1,n_2) \qquad \text{(see before)} \\
[\![!=]\!](v_1,v_2) &\stackrel{\text{def}}{=} [\![!]\!]([\![==]\!](v_1,v_2))
\end{aligned}
$$

Fig. 8. Comparison of values

$$
\text{(EVAL-NULL)}\ \frac{}{\Sigma \ \vdash\ \texttt{null} \Downarrow \bot, \langle \Sigma \rangle}
$$

$$
\text{(EVAL-SELECT)}\ \frac{\Sigma \ \vdash\ t \Downarrow \mathbb{O}(id,\sigma,\overline{m}), \langle \Sigma' \rangle \qquad \sigma; \epsilon \ \vdash\ a \Rightarrow v}{\Sigma \ \vdash\ t.a \Downarrow v, \langle \Sigma' \rangle}
$$

$$
\text{(EVAL-CALL)}\ \frac{\begin{array}{c}\Sigma \ \vdash\ t \Downarrow \mathbb{O}(id,\sigma,\overline{m}), \langle \Sigma_0 \rangle \qquad \texttt{def}\ a(a_1 : T_1, \ldots, a_n : T_n) : T = u \in \overline{m} \\ \forall 1 \le i \le n : \Sigma_{i-1} \ \vdash\ t_i \Downarrow v_i, \langle \Sigma_i \rangle \\ this \mapsto \mathbb{O}(id,\sigma,\overline{m}), a_1 \mapsto v_1, \ldots, a_n \mapsto v_n; \epsilon \ \vdash\ u \Downarrow v, \langle \Sigma' \rangle\end{array}}{\Sigma \ \vdash\ t.a(t_1,\ldots,t_n) \Downarrow v, \langle \Sigma_n \rangle}
$$

$$
\text{(EVAL-NEW)}\ \frac{\begin{array}{c}\Sigma_0 = \Sigma \qquad \forall 1 \le i \le n : \Sigma_{i-1} \ \vdash\ t_i \Downarrow v_i, \langle \Sigma_i \rangle \\ v = createObj(a)\langle v_1,\ldots,v_n \rangle\end{array}}{\Sigma \ \vdash\ \texttt{new}\ a(t_1,\ldots,t_n) \Downarrow v, \langle \Sigma_n \rangle}
$$

$$
\text{(EVAL-EQNEQ)}\ \frac{\begin{array}{c}\Sigma \ \vdash\ t_1 \Downarrow v_1, \langle \Sigma' \rangle \qquad \Sigma' \ \vdash\ t_2 \Downarrow v_2, \langle \Sigma'' \rangle \\ binop \in \{==, !=\} \qquad n = [\![binop]\!](v_1,v_2)\end{array}}{\Sigma \ \vdash\ t_1\ binop\ t_2 \Downarrow \mathbb{I}(n), \langle \Sigma'' \rangle}
$$

Fig. 9. Evaluation of object layer

$$\text{CREATE-ROOT}\ \frac{\texttt{class}\ a\ \texttt{extends none}\ \{\overline{d}\} \in \overline{D} \qquad id \text{ is a ``fresh'' integer} \qquad o = initObj(\mathbb{O}(id, \epsilon, \epsilon), \langle\overline{d}\rangle, \langle v_1, \ldots v_n\rangle)}{createObj(a)\langle v_1, \ldots, v_n\rangle \stackrel{\text{def}}{=} o}$$

$$\text{CREATE-CHILD}\ \frac{\begin{array}{c}\texttt{class}\ a\ \texttt{extends}\ b\ \{\overline{d}\} \in \overline{D} \\ \overline{d} \text{ contains exactly } k \text{ field declarations} \qquad k \leq n \\ o = initObj(createObj(b)\langle v_1, \ldots, v_{n-k}\rangle, \langle\overline{d}\rangle, \langle v_{n-k+1}, \ldots, v_n\rangle)\end{array}}{createObj(a)\langle v_1, \ldots, v_n\rangle \stackrel{\text{def}}{=} o}$$

Fig. 10. Creation of objects (with $\overline{D}$ as list of class declarations)

$$\text{INIT-DONE}\ \frac{}{initObj(\mathbb{O}(id, \sigma, \overline{m}), \langle\epsilon\rangle, \langle\epsilon\rangle) \stackrel{\text{def}}{=} \mathbb{O}(id, \sigma, \overline{m})}$$

$$\text{INIT-FIELD}\ \frac{\sigma; \epsilon \vdash a \mapsto v \Rightarrow \sigma'; \epsilon \qquad o = initObj(\mathbb{O}(id, \sigma', \overline{m}), \langle\overline{d}\rangle, \langle\overline{v}\rangle)}{initObj(\mathbb{O}(id, \sigma, \overline{m}), \langle\texttt{val}\ a : T; \overline{d}\rangle, \langle v, \overline{v}\rangle) \stackrel{\text{def}}{=} o}$$

$$\text{INIT-METHOD}\ \frac{\begin{array}{c}\texttt{def}\ a(\overline{a}' : \overline{T}') : T' = t' \notin \overline{m} \\ \overline{m}' = \texttt{def}\ a(\overline{a} : \overline{T}) : T = t, \overline{m} \qquad o = initObj(\mathbb{O}(id, \sigma, \overline{m}'), \langle\overline{d}\rangle, \langle\overline{v}\rangle)\end{array}}{initObj(\mathbb{O}(id, \sigma, \overline{m}), \langle\texttt{def}\ a(\overline{a} : \overline{T}) : T = t; \overline{d}\rangle, \langle\overline{v}\rangle) \stackrel{\text{def}}{=} o}$$

$$\text{INIT-OVERRIDE}\ \frac{\begin{array}{c}\overline{m} = \overline{m}_1, \texttt{def}\ a(\overline{a}' : \overline{T}') : T' = t', \overline{m}_2 \\ \overline{m}' = \overline{m}_1, \texttt{def}\ a(\overline{a} : \overline{T}) : T = t, \overline{m}_2 \\ o = initObj(\mathbb{O}(id, \sigma, \overline{m}'), \langle\overline{d}\rangle, \langle\overline{v}\rangle)\end{array}}{initObj(\mathbb{O}(id, \sigma, \overline{m}), \langle\texttt{def}\ a(\overline{a} : \overline{T}) : T = t; \overline{d}\rangle, \langle\overline{v}\rangle) \stackrel{\text{def}}{=} o}$$

Fig. 11. Initialisation of objects