c	numeric value
a	array
x	term variable
f	function name
κ	program counter
n	index variable
k	index variable
b_h	boolean value
c_h	numeric value
f_h	function name
rval	"return value" variable
rnset	"rval-not-set" variable

l	::=		literals
	|	TRUE	bitmask true (`0b1111...`)
	|	FALSE	bitmask false (`0b0000...`)
v	::=		values
	|	l	numeric literal
	|	c	numeric value
	|	a	bytearray
$\ominus$	::=		unary operations
	|	`~`	bitwise not
$\oplus$	::=		binary operations
	|	`+`	
	|	`-`	
	|	`*`	
	|	`<<`	
	|	`>>`	
	|	`&`	bitwise and
	|	`|`	bitwise or
	|	$\texttt{==}_\texttt{s}$	equals (sign extended)
	|	$\texttt{!=}_\texttt{s}$	
	|	$\texttt{>}_\texttt{s}$	
	|	$\texttt{<}_\texttt{s}$	
	|	$\texttt{>=}_\texttt{s}$	
	|	$\texttt{<=}_\texttt{s}$	
fval	::=		function spec
	|	$(x_1, .., x_n) : s$ `@`e	
fndef	::=		function definition
	|	**fdef** f *fval*	
program	::=		program
	|	$fndef_1; ..; fndef_n;$ **expose** *fndef*	list of fdefs
Λ	::=		function store
	|	$\emptyset_\Lambda$	empty function store
	|	$\Lambda[f \mapsto fval]$	define function
Γ	::=		global memory
	|	$\emptyset_\Gamma$	
	|	$\Gamma[a \mapsto \texttt{[]}]$	new array
	|	$\Gamma(a)[v_1 \mapsto v_2]$	array update
μ	::=		local memory
	|	$\emptyset_\mu$	empty memory

	$\mu[x \mapsto v]$	add/update variable
|	$\mu_1 \vert \mu_2$	push stack frame

e	::=		expressions
	|	l	numeric literal
	|	c	numeric value
	|	a	bytearray
	|	x	variable
	|	$a[e]$	array access
	|	$\ominus e$	unary operation
	|	$e_1 \oplus e_2$	binary operation
	|	$f(e_1, .., e_n)$	function application
	|	s `@`e	function body

s	::=		statements
	|	**skip**	skip
	|	$s_1; s_2$	sequence
	|	$\{x_1/v_1, .., x_k/v_k\}s$	variable substitution
	|	**def** $x := e$	variable declaration
	|	**def** $x := a$	array declaration
	|	$x := e$	variable assignment
	|	$a[e_1] := e_2$	array assignment
	|	**for** x **from** v_1 **to** $v_2 : s$	for loop

v_h	::=		values
	|	b_h	boolean value
	|	c_h	numeric value

$\ominus_h$	::=		unary operations
	|	`!`	logical not
	|	`~`	bitwise not

$\oplus_h$	::=		binary operations
	|	`+`	
	|	`-`	
	|	`*`	
	|	`<<`	
	|	`>>`	
	|	`&`	
	|	`|`	
	|	`&&`	
	|	`||`	
	|	`==`	
	|	`!=`	
	|	`>`	
	|	`<`	
	|	`>=`	

	|	`<=`

e_h	::=		expressions
	|	b_h	boolean value
	|	c_h	numeric value
	|	x	variable
	|	$a[e_h]$	array access
	|	$\ominus_h e_h$	unary operation
	|	$e_{h1} \oplus e_{h2}$	binary operation
	|	$f_h(e_{h1}, .., e_{hn})$	function application

s_h	::=		statements
	|	**hskip**	skip
	|	$s_{h1}; s_{h2}$	sequence
	|	**hdef** $x := e_h$	variable declaration
	|	**hdef** $x := a$	array declaration
	|	$x := e_h$	variable assignment
	|	$a[e_{h1}] := e_{h2}$	array assignment
	|	**hfor** x **from** v_{h1} **to** $v_{h2} : s_h$	for loop
	|	**hif** e_h **then** s_{h1} **else** s_{h2}	conditional branch
	|	**hreturn** e_h	return

hfval	::=		function spec
	|	$(x_1, .., x_n) : s_h$	

hfndef	::=		function definition
	|	**fdef** f_h *hfval*	

hprogram	::=		program
	|	$hfndef_1; .. ; hfndef_n;$ **expose** *hfndef*	list of fdefs

ctx	::=		branch context
	|	l	numeric literal
	|	x	variable
	|	$\ominus ctx$	unary operation
	|	$ctx_1 \oplus ctx_2$	binary operation

$\boxed{\{\Lambda, \Gamma, \mu, \kappa\}\ e \longrightarrow \{\Lambda', \Gamma', \mu', \kappa'\}\ e'}$ e reduces to e'

$$\frac{\begin{array}{c}\mu = \mu'[x \mapsto v] \\ \kappa' = \kappa + 1\end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\ x \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ v}\ \textsc{Exr\_var}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e'}{\{\Lambda, \Gamma, \mu, \kappa\}\ a[e] \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ a[e']}\ \textsc{Exr\_arr\_get\_expr}$$

$$\frac{\begin{array}{c}v' = \Gamma(a)[v] \\ \kappa' = \kappa + 1\end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\ a[v] \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ v'}\ \textsc{Exr\_arr\_get\_val}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e'}{\{\Lambda, \Gamma, \mu, \kappa\}\ \ominus e \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ \ominus e'}\ \textsc{Exr\_unop\_expr}$$

$$\frac{\begin{array}{c} v' \equiv [\![\ominus v]\!] \\ \kappa' = \kappa + 1 \end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\ \ominus v \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ v'} \quad \textsc{Exr\_unop\_val}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e_1 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e_1'}{\{\Lambda, \Gamma, \mu, \kappa\}\ e_1 \oplus e_2 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e_1' \oplus e_2} \quad \textsc{Exr\_binop\_l}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e_2 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e_2'}{\{\Lambda, \Gamma, \mu, \kappa\}\ v \oplus e_2 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ v \oplus e_2'} \quad \textsc{Exr\_binop\_r}$$

$$\frac{\begin{array}{c} v_3 \equiv [\![v_1 \oplus v_2]\!] \\ \kappa' = \kappa + 1 \end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\ v_1 \oplus v_2 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ v_3} \quad \textsc{Exr\_binop\_val}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e_1 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e_1'}{\{\Lambda, \Gamma, \mu, \kappa\}\, f(v_1, \ldots, v_k, e_1, e_2, \ldots, e_n) \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\, f(v_1, \ldots, v_k, e_1', e_2, \ldots, e_n)} \quad \textsc{Exr\_fn\_expr}$$

$$\frac{\begin{array}{c} \Lambda = \Lambda'[f \mapsto (x_1, \ldots, x_k) : s\ @e] \\ \mu' = \mu|\emptyset_\mu \\ \kappa' = \kappa + 1 \end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\, f(v_1, \ldots, v_k) \longrightarrow \{\Lambda, \Gamma, \mu', \kappa'\}\, \{x_1/v_1, \ldots, x_k/v_k\} s\ @e} \quad \textsc{Exr\_fn\_call}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e'}{\{\Lambda, \Gamma, \mu, \kappa\}\, \mathbf{skip}\ @e \longrightarrow \{\Lambda, \Gamma, \mu, \kappa\}\ e'} \quad \textsc{Exr\_skip\_expr}$$

$$\frac{\mu = \mu_1|\mu_2}{\{\Lambda, \Gamma, \mu, \kappa\}\, \mathbf{skip}\ @v \longrightarrow \{\Lambda, \Gamma, \mu_1, \kappa\}\ v} \quad \textsc{Exr\_skip\_val}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ s_1\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu', \kappa'\}\ s_1'\ @e_0}{\{\Lambda, \Gamma, \mu, \kappa\}\ s_1; s_2\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu', \kappa'\}\ s_1'; s_2\ @e_0} \quad \textsc{Exr\_seq}$$

$$\frac{}{\{\Lambda, \Gamma, \mu, \kappa\}\, \mathbf{skip}; s\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa\}\ s\ @e_0} \quad \textsc{Exr\_seq\_skip}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e'}{\{\Lambda, \Gamma, \mu, \kappa\}\, \mathbf{def}\ x := e\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\, \mathbf{def}\ x := e'\ @e_0} \quad \textsc{Exr\_def\_expr}$$

$$\frac{\begin{array}{c} \mu' = \mu[x \mapsto v] \\ \kappa' = \kappa + 1 \end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\, \mathbf{def}\ x := v\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu', \kappa'\}\, \mathbf{skip}\ @e_0} \quad \textsc{Exr\_def\_val}$$

$$\frac{\begin{array}{c} \Gamma' = \Gamma[a \mapsto []] \\ \mu' = \mu[x \mapsto a] \\ \kappa' = \kappa + 1 \end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\, \mathbf{def}\ x := a\ @e_0 \longrightarrow \{\Lambda, \Gamma', \mu', \kappa'\}\, \mathbf{skip}\ @e_0} \quad \textsc{Exr\_def\_arr}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e'}{\{\Lambda, \Gamma, \mu, \kappa\}\ x := e\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ x := e'\ @e_0} \quad \textsc{Exr\_assign\_expr}$$

$$\frac{\begin{array}{c} \mu' = \mu[x \mapsto v] \\ \kappa' = \kappa + 1 \end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\ x := v\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu', \kappa'\}\, \mathbf{skip}\ @e_0} \quad \textsc{Exr\_assign\_val}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e_1 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e_1'}{\{\Lambda, \Gamma, \mu, \kappa\}\ a[e_1] := e_2\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ a[e_1'] := e_2\ @e_0} \quad \textsc{Exr\_arr\_assign\_expr\_l}$$

$$\frac{\{\Lambda, \Gamma, \mu, \kappa\}\ e_2 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ e_2'}{\{\Lambda, \Gamma, \mu, \kappa\}\ a[v_1] := e_2\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ a[v_1] := e_2'\ @e_0} \quad \textsc{Exr\_arr\_assign\_expr\_r}$$

$$\frac{\begin{array}{c}\Gamma' = \Gamma(a)[v_1 \mapsto v_2] \\ \kappa' = \kappa + 1\end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\ a[v_1] := v_2\ @e_0 \longrightarrow \{\Lambda, \Gamma', \mu, \kappa'\}\ \textbf{skip}\ @e_0}\ \textsc{Exr\_arr\_assign\_val}$$

$$\frac{\begin{array}{c}x \notin \mu \\ v_1 < v_2 \\ v_1' = v_1 + 1 \\ \kappa' = \kappa + 1\end{array}}{\{\Lambda, \Gamma, \mu, \kappa\}\ \textbf{for}\ x\ \textbf{from}\ v_1\ \textbf{to}\ v_2 : s\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu, \kappa'\}\ \{x/v_1\}s; \textbf{for}\ x\ \textbf{from}\ v_1'\ \textbf{to}\ v_2 : s\ @e_0}\ \textsc{Exr\_for}$$

$$\frac{v_1 = v_2}{\{\Lambda, \Gamma, \mu, \kappa\}\ \textbf{for}\ x\ \textbf{from}\ v_2\ \textbf{to}\ v_2 : s\ @e_0 \longrightarrow \{\Lambda, \Gamma, \mu', \kappa\}\ \textbf{skip}\ @e_0}\ \textsc{Exr\_for\_base}$$

$\boxed{e_h \longrightarrow_t e}$ e_h is transformed to e

$$\frac{v \equiv [\![v_h]\!]_t}{v_h \longrightarrow_t v}\ \textsc{Ext\_val}$$

$$\frac{e_h \longrightarrow_t e}{a[e_h] \longrightarrow_t a[e]}\ \textsc{Ext\_arr\_get}$$

$$\frac{\begin{array}{c}\textbf{fdef}\ f_h\ hfval \longrightarrow_t \textbf{fdef}\ f\ fval \\ e_{h1} \longrightarrow_t e_1 \quad .. \quad e_{hk} \longrightarrow_t e_k\end{array}}{f_h(e_{h1}, .., e_{hk}) \longrightarrow_t f(e_1, .., e_k)}\ \textsc{Ext\_fn\_call}$$

$\boxed{s_h \xrightarrow{ctx}_t s}$ s_h is transformed to s

$$\frac{}{\textbf{hskip} \xrightarrow{ctx}_t \textbf{skip}}\ \textsc{Stt\_skip}$$

$$\frac{\begin{array}{c}s_{h1} \xrightarrow{ctx}_t s_1 \\ s_{h2} \xrightarrow{ctx}_t s_2\end{array}}{s_{h1}; s_{h2} \xrightarrow{ctx}_t s_1; s_2}\ \textsc{Stt\_seq}$$

$$\frac{e_h \longrightarrow_t e}{\textbf{hdef}\ x := e_h \xrightarrow{ctx}_t \textbf{def}\ x := e}\ \textsc{Stt\_var\_dec}$$

$$\frac{}{\textbf{hdef}\ x := a \xrightarrow{ctx}_t \textbf{def}\ x := a}\ \textsc{Stt\_arr\_dec}$$

$$\frac{\begin{array}{c}e_h \longrightarrow_t e \\ e' = ctx\ \&\ rnset \\ e'' = e\ \&\ e' \\ e''' = x\ \&\ (\sim e')\end{array}}{x := e_h \xrightarrow{ctx}_t x := (e''\ |\ e''')}\ \textsc{Stt\_var\_assign}$$

$$\frac{\begin{array}{c}e_{h1} \longrightarrow_t e_1 \\ e_{h2} \longrightarrow_t e_2 \\ e' = ctx\ \&\ rnset \\ e'' = e_2\ \&\ e' \\ e''' = a[e_1]\ \&\ (\sim e')\end{array}}{a[e_{h1}] := e_{h2} \xrightarrow{ctx}_t a[e_1] := (e''\ |\ e''')}\ \textsc{Stt\_arr\_assign}$$

$$\frac{\begin{array}{c}v_{h1} \longrightarrow_t v_1 \\ v_{h2} \longrightarrow_t v_2 \\ s_h \xrightarrow{ctx}_t s\end{array}}{\textbf{hfor}\ x\ \textbf{from}\ v_{h1}\ \textbf{to}\ v_{h2} : s_h \xrightarrow{ctx}_t \textbf{for}\ x\ \textbf{from}\ v_1\ \textbf{to}\ v_2 : s}\ \textsc{Stt\_for}$$

$$\frac{\begin{array}{l} e_h \longrightarrow_t e \\ ctx_1 = e \,\&\, ctx \\ ctx_2 = (\sim e) \,\&\, ctx \\ s_{h1} \xrightarrow{ctx_1}_t s_1 \\ s_{h2} \xrightarrow{ctx_2}_t s_2 \end{array}}{\textbf{hif}\ e_h\ \textbf{then}\ s_{h1}\ \textbf{else}\ s_{h2} \xrightarrow{ctx}_t s_1; s_2} \quad \textsc{Stt\_if}$$

$$\frac{\begin{array}{l} e_h \longrightarrow_t e \\ e' = ctx \,\&\, rnset \\ e'' = e \,\&\, e' \end{array}}{\textbf{hreturn}\ e_h \xrightarrow{ctx}_t rval := (e'' \mid rval); rnset := (rnset \,\&\, (\sim ctx))} \quad \textsc{Stt\_ret}$$

$\boxed{hfndef \longrightarrow_t fndef}$ *hfndef* is transformed to *fndef*

$$\frac{s_h \xrightarrow{\textsc{True}}_t s}{\textbf{fdef}\ f_h\,(x_1,\,..\,,x_k) : s_h \longrightarrow_t \textbf{fdef}\ f\,(x_1,\,..\,,x_k) : s\ \texttt{@}rval} \quad \textsc{Fdeft\_fdef}$$

$\boxed{\ominus_h \xrightarrow{\ominus} \ominus}$ $\ominus_h$ is transformed to $\ominus$

$$\frac{}{\texttt{!} \xrightarrow{\ominus} \sim} \quad \textsc{Unopt\_lnot}$$

$$\frac{}{\ominus_h \xrightarrow{\ominus} \ominus} \quad \textsc{Unopt\_unop}$$

$\boxed{\oplus_h \xrightarrow{\oplus} \oplus}$ $\oplus_h$ is transformed to $\oplus$

$$\frac{}{\texttt{\&\&} \xrightarrow{\oplus} \texttt{\&}} \quad \textsc{Binopt\_land}$$

$$\frac{}{\texttt{||} \xrightarrow{\oplus} \texttt{|}} \quad \textsc{Binopt\_lor}$$

$$\frac{}{\texttt{==} \xrightarrow{\oplus} \texttt{==}_\texttt{s}} \quad \textsc{Binopt\_eq}$$

$$\frac{}{\texttt{!=} \xrightarrow{\oplus} \texttt{!=}_\texttt{s}} \quad \textsc{Binopt\_neq}$$

$$\frac{}{\texttt{>} \xrightarrow{\oplus} \texttt{>}_\texttt{s}} \quad \textsc{Binopt\_gt}$$

$$\frac{}{\texttt{<} \xrightarrow{\oplus} \texttt{<}_\texttt{s}} \quad \textsc{Binopt\_lt}$$

$$\frac{}{\texttt{>=} \xrightarrow{\oplus} \texttt{>=}_\texttt{s}} \quad \textsc{Binopt\_gte}$$

$$\frac{}{\texttt{<=} \xrightarrow{\oplus} \texttt{<=}_\texttt{s}} \quad \textsc{Binopt\_lte}$$

$$\frac{}{\oplus_h \xrightarrow{\oplus} \oplus} \quad \textsc{Binopt\_binop}$$

```
Definition rules:          48 good    0 bad
Definition rule clauses: 113 good    0 bad
```