MAGNET – Mutable Arithmetic Expressions with Generators and Exceptions

1 INTRODUCTION

MAGNET is a toy language for the COSE212 course at Korea University. MAGNET stands for the **M**utable **A**rithmetic Expressions with **G**enerators and **E**xceptions, and it supports the following features:

- **undefined value** (undefined):
- **number values** (0, 1, -1, 2, -2, 3, -3, . . .)
- **boolean values** (true and false)
- **arithmetic operators**: negation (-), addition (+), subtraction (-), multiplication (*), division (/), and modulo (%)
- **logical operators**: conjunction (&&), disjunction (||), and negation (!)
- **comparison operators**: equality (== and !=) and relational (<, >, <=, and >=)
- **mutable variable definitions** (var) and **identifier lookup** (x)
- **variable assignment** (=) and **sequences** (;)
- **augmented assignment** (+=, -=, *=, /=, and %=) and **increment/decrement** (++ and --)
- **conditionals** (if-else), **while loops** (while), **break** (break), and **continue** (continue)
- **first-class functions** (=> or function)
- **function applications** and **return** (return)
- **try-catch** (try-catch) and **throw** (throw)
- **generators** (=>* or function*) and **yield** (yield)
- **iterator next** (_.next) and **iterator result accessors** (_.value and _.done)
- **for-of loops** (for-of)

This document is the specification of MAGNET. First, Section 2 describes the concrete syntax, and Section 3 describes the abstract syntax with the desugaring rules. Then, Section 4 describes the small-step operational (reduction) semantics of MAGNET.

2 CONCRETE SYNTAX

The concrete syntax of MAGNET is written in a variant of the extended Backus–Naur form (EBNF). The notation `<nt>` denotes a nonterminal, and `"t"` denotes a terminal. We use ? to denote an optional element and + (or *) to denote one or more (or zero or more) repetitions of the preceding element. We use butnot to denote a set difference to exclude some strings from a producible set of strings. We omit some obvious terminals using the ellipsis (. . .) notation.

```
// basic elements
<digit>    ::= "0" | "1" | "2" | ... | "9"
<number>   ::= "-"? <digit>+
<alphabet> ::= "A" | "B" | "C" | ... | "Z" | "a" | "b" | "c" | ... | "z"
<idstart>  ::= <alphabet> | "_"
<idcont>   ::= <alphabet> | "_" | <digit>
<keyword>  ::= "break" | "catch" | "continue" | "else" | "false"
             | "for" | "function" | "if" | "of" | "return" | "throw"
             | "true" | "try" | "undefined" | "var" | "while" | "yield"
<id>       ::= <idstart> <idcont>* butnot <keyword>
```

```
// expressions
<expr> ::= "undefined" | <number> | "true" | "false"
          // unary/binary operators and parentheses
          | <uop> <expr> | <expr> <bop> <expr> | "(" <expr> ")" | "{" <expr> "}"
          // mutable variable definitions
          "var" <id> "=" <expr> ";" <expr> | <id>
          // variable (augmented) assignment and sequence
          | <id> <aop> <expr> | <expr> ";" <expr>?
          // increment and decrement
          | "++" <id> | "--" <id> | <id> "++" | <id> "--"
          // conditionals and loops
          | "if" "(" <expr> ")" <expr> "else" <expr>
          | "while" "(" <expr> ")" <expr>
          // first-class functions
          | <id> "=>" <expr> | <params> "=>" <expr>
          | "function" <id> <params> "{" <expr> "}" <expr>
          // function applications and returns
          | <expr> "(" <expr> ")" | "return" <expr>
          // try-catch and throw
          | "try" <expr> "catch" "(" <id> ")" <expr> | "throw" <expr>
          // generators and yields
          | <id> "=>" "*" <expr> | <params> "=>" "*" <expr>
          | "function" "*" <id> <params> "{" <expr> "}" <expr> | "yield" <expr>
          // iterator next and iterator result accessors
          | <expr> "." "next" "(" <expr>? ")"
          | <expr> "." "value" | <expr> "." "done"
// operators
<aop> ::= "=" | "+=" | "-=" | "*=" | "/=" | "%="
<uop> ::= "-" | "!"
<bop> ::= "+" | "-" | "*" | "/" | "%" | "&&" | "||"
        | "==" | "!=" | "<" | "<=" | ">" | ">="
// parameters
<params> ::= "(" ")" | "(" <id> ")" | "(" <id> ["," <id>]* ")"
```

The precedence and associativity of operators are defined as follows:

Description	Operator	Precedence	Associativity
Postfix Unary	++, --, _.next, _.value, _.done	1	left
Prefix Unary	-, !, ++, --	2	right
Multiplicative	*, /, %	3	
Additive	+, -	4	
Relational	<, <=, >, >=	5	left
Equality	==, !=	6	
Logical Conjunction	&&	7	
Logical Disjunction	||	8	
Assignment	=, +=, -=, *=, /=, %=, var	9	right
Sequence	;	10	left

3 ABSTRACT SYNTAX

The abstract syntax of MAGNET is defined as follows:

Expressions	$\mathbb{E} \ni e ::=$ `undefined`	(EUndef)		`if (`e`)` e `else` e	(EIf)
	| n	(ENum)	|	`while (`e`)` e	(EWhile)
	| b	(EBool)	|	`break`	(EBreak)
	| e `+` e	(Add)	|	`continue`	(EContinue)
	| e `*` e	(Mul)	|	$\lambda(x, \ldots, x).e$	(EFun)
	| e `/` e	(EDiv)	|	$e(e, \ldots, e)$	(EApp)
	| e `%` e	(EMod)	|	`return` e	(EReturn)
	| e `==` e	(EEq)	|	`try` e `catch (`x`)` e	(ETry)
	| e `<` e	(ELt)	|	`throw` e	(EThrow)
	| `var` x`=`e`;` e	(EVar)	|	$\lambda*(x, \ldots, x).e$	(EGen)
	| x	(EId)	|	e`.next(`$e^?$`)`	(EIterNext)
	| x`=`e	(EAssign)	|	`yield` e	(EYield)
	| e`;` e	(ESeq)	|	e`.value`	(EValueField)
			|	e`.done`	(EDoneField)

where $\begin{cases} \text{Identifier} & x \in \mathbb{X} \quad (\texttt{String}) \\ \text{Number} & n \in \mathbb{Z} \quad (\texttt{BigInt}) \end{cases}$ Boolean $\mathbb{B} \ni b ::= \texttt{true} \mid \texttt{false}$ (Boolean)

The semantics of the remaining cases are defined with the following desugaring rules:

$\mathcal{D}[\![\texttt{-} e]\!] \quad = \mathcal{D}[\![e]\!] \texttt{ * } (-1)$

$\mathcal{D}[\![e_1 \texttt{ - } e_2]\!] = \mathcal{D}[\![e_1]\!] \texttt{ + } \mathcal{D}[\![\texttt{-} e_2]\!]$

$\mathcal{D}[\![e_1 \texttt{ \&\& } e_2]\!] = \texttt{if (}\mathcal{D}[\![e_1]\!]\texttt{) } \mathcal{D}[\![e_2]\!] \texttt{ else false}$

$\mathcal{D}[\![e_1 \texttt{ || } e_2]\!] = \texttt{if (}\mathcal{D}[\![e_1]\!]\texttt{) true else } \mathcal{D}[\![e_2]\!]$

$\mathcal{D}[\![\texttt{! } e]\!] \quad = \texttt{if (}\mathcal{D}[\![e]\!]\texttt{) false else true}$

$\mathcal{D}[\![e_1 \texttt{ != } e_2]\!] = \mathcal{D}[\![\texttt{! } (e_1 \texttt{ == } e_2)]\!]$

$\mathcal{D}[\![e_1 \texttt{ <= } e_2]\!] = \texttt{var } \underline{x_1} = \mathcal{D}[\![e_1]\!];$
$\qquad\qquad\qquad \texttt{var } \underline{x_2} = \mathcal{D}[\![e_2]\!];$
$\qquad\qquad\qquad \mathcal{D}[\![(\underline{x_1} \texttt{ == } \underline{x_2}) \texttt{ || } (\underline{x_1} \texttt{ < } \underline{x_2})]\!]$

$\mathcal{D}[\![e_1 \texttt{ > } e_2]\!] \quad = \mathcal{D}[\![\texttt{! } (e_1 \texttt{ <= } e_2)]\!]$

$\mathcal{D}[\![e_1 \texttt{ >= } e_2]\!] = \mathcal{D}[\![\texttt{! } (e_1 \texttt{ < } e_2)]\!]$

$\mathcal{D}[\![x \texttt{ += } e]\!] = x \texttt{ = } x \texttt{ + } \mathcal{D}[\![e]\!]$

$\mathcal{D}[\![x \texttt{ -= } e]\!] = x \texttt{ = } x \texttt{ - } \mathcal{D}[\![e]\!]$

$\mathcal{D}[\![x \texttt{ *= } e]\!] = x \texttt{ = } x \texttt{ * } \mathcal{D}[\![e]\!]$

$\mathcal{D}[\![x \texttt{ /= } e]\!] = x \texttt{ = } x \texttt{ / } \mathcal{D}[\![e]\!]$

$\mathcal{D}[\![x \texttt{ %= } e]\!] = x \texttt{ = } x \texttt{ % } \mathcal{D}[\![e]\!]$

$\mathcal{D}[\![\texttt{++ } x]\!] \quad = \mathcal{D}[\![x \texttt{ += } 1]\!]$

$\mathcal{D}[\![\texttt{-- } x]\!] \quad = \mathcal{D}[\![x \texttt{ -= } 1]\!]$

$\mathcal{D}[\![x \texttt{ ++}]\!] \quad = \texttt{var } \underline{x_1} = x; \ \mathcal{D}[\![x \texttt{ += } 1]\!]; \ \underline{x_1}$

$\mathcal{D}[\![x \texttt{ --}]\!] \quad = \texttt{var } \underline{x_1} = x; \ \mathcal{D}[\![x \texttt{ -= } 1]\!]; \ \underline{x_1}$

$\mathcal{D}[\![\texttt{function } x \texttt{ (}x_1, \ \ldots, \ x_n\texttt{)\{ } e_1 \texttt{ \} } e_2]\!] = \texttt{var } x = \texttt{(}x_1, \ \ldots, \ x_n\texttt{)} \texttt{ => } e_1; \ e_2$

$\mathcal{D}[\![\texttt{function* } x \texttt{ (}x_1, \ \ldots, \ x_n\texttt{)\{ } e_1 \texttt{ \} } e_2]\!] = \texttt{var } x = \texttt{(}x_1, \ \ldots, \ x_n\texttt{)} \texttt{ =>* } e_1; \ e_2$

$$\mathcal{D}[\![\texttt{for (}x \texttt{ of } e_1\texttt{) } e_2]\!] = \mathcal{D} \left\Vert \begin{array}{l} \texttt{var } \underline{x_1} = e_1; \\ \texttt{var } \underline{x_2} = \underline{x_1}\texttt{.next()}; \\ \texttt{while (! } \underline{x_2}\texttt{.done) \{} \\ \quad \texttt{var } x = \underline{x_2}\texttt{.value}; \\ \quad e_2; \ \underline{x_2} = \underline{x_1}\texttt{.next()} \\ \texttt{\}} \end{array} \right\Vert$$

where $\underline{x_k}$ denotes a fresh temporary variable. All the omitted cases recursively apply the desugaring rule to their sub-expressions. For example, $\mathcal{D}[\![e_1 \texttt{ + } e_2]\!] = \mathcal{D}[\![e_1]\!] \texttt{ + } \mathcal{D}[\![e_2]\!]$.

4 SEMANTICS

We use the following notations in the semantics:

$$\text{States} \qquad \langle \kappa \parallel s \parallel H \parallel M \rangle \ \in \ \mathbb{K} \times \mathbb{S} \times \mathbb{H} \times \mathbb{M} \quad \text{(State)}$$

$$\text{Continuations} \qquad \kappa \ \in \ \mathbb{K} \qquad\qquad \text{(Cont)}$$
$$\kappa \ ::= \square \ | \ i :: \kappa$$

Instructions	$i \ \in \ \mathbb{I}$	(Inst)		
	$i \ ::= (\sigma \vdash e)$	(IEval)	$\| \ \texttt{jmp-if}[\psi]$	(IJmpIf)
	$\| \ (+)$	(IAdd)	$\| \ \texttt{jmp}[c]$	(IJmp)
	$\| \ (*)$	(IMul)	$\| \ \texttt{call}[n]$	(ICall)
	$\| \ (/)$	(IDiv)	$\| \ \texttt{return}$	(IReturn)
	$\| \ (\%)$	(IMod)	$\| \ \texttt{next}$	(INext)
	$\| \ (==)$	(IEq)	$\| \ \texttt{yield}$	(IYield)
	$\| \ (<)$	(ILt)	$\| \ \texttt{value}$	(IValueField)
	$\| \ \texttt{def}[x,\dots,x][\sigma \vdash e]$	(IDef)	$\| \ \texttt{done}$	(IDoneField)
	$\| \ \texttt{write}[a]$	(IWrite)	$\| \ \texttt{pop}$	(IPop)

$$\text{Value Stacks} \quad s \ \in \ \mathbb{S} \qquad \text{(Stack)}$$
$$s \ ::= \blacksquare \ | \ v :: s$$

Values	$v \ \in \ \mathbb{V}$	(Value)		
	$v \ ::= \texttt{undefined}$	(UndefV)	$\| \ \langle \kappa \parallel s \parallel H \rangle$	(ContV)
	$\| \ n$	(NumV)	$\| \ \langle \lambda*(x,\dots,x).e, \sigma \rangle$	(GenV)
	$\| \ b$	(BoolV)	$\| \ \texttt{iter}[a]$	(IterV)
	$\| \ \langle \lambda(x,\dots,x).e, \sigma \rangle$	(CloV)	$\| \ \{\texttt{value} : v, \texttt{done} : b\}$	(ResultV)

$$\text{Control Handlers} \quad H \ \in \ \mathbb{H} = \mathbb{C} \xrightarrow{\text{fin}} \Psi \quad \text{(Handler)}$$

Control Operators	$c \ \in \ \mathbb{C}$	(Control)		
	$c \ ::= \texttt{return}$	(Return)	$\| \ \texttt{throw}$	(Throw)
	$\| \ \texttt{break}$	(Break)	$\| \ \texttt{finally}$	(Finally)
	$\| \ \texttt{continue}$	(Continue)	$\| \ \texttt{yield}$	(Yield)

$$\text{Continuation Values} \quad \psi, \langle \kappa \parallel s \parallel H \rangle \ \in \ \Psi = \mathbb{K} \times \mathbb{S} \times \mathbb{H} \quad \text{(KValue)}$$

$$\text{Memories} \qquad M \ \in \ \mathbb{M} = \mathbb{A} \xrightarrow{\text{fin}} \mathbb{V} \quad \text{(Mem)}$$

$$\text{Environments} \qquad \sigma \ \in \ \mathbb{X} \xrightarrow{\text{fin}} \mathbb{A} \qquad \text{(Env)}$$

$$\text{Addresses} \qquad a \ \in \ \mathbb{A} \qquad\qquad \text{(Addr)}$$

The small-step operational (reduction) semantics of MAGNET is defined in the following form of the reduction relation ($\rightarrow$):

$$\boxed{\langle \kappa \parallel s \parallel H \parallel M \rangle \rightarrow \langle \kappa \parallel s \parallel H \parallel M \rangle}$$

4.1 Reduction Relations for `IEval`

EUndef	$\langle(\sigma \vdash \mathsf{undefined}) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle\kappa \mathbin{\|} \mathsf{undefined} :: s \mathbin{\|} H \mathbin{\|} M\rangle$
ENum	$\langle(\sigma \vdash n) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle\kappa \mathbin{\|} n :: s \mathbin{\|} H \mathbin{\|} M\rangle$
EBool	$\langle(\sigma \vdash b) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle\kappa \mathbin{\|} b :: s \mathbin{\|} H \mathbin{\|} M\rangle$
EAdd	$\langle(\sigma \vdash e_1 + e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle(\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (+) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
EMul	$\langle(\sigma \vdash e_1 * e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle(\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (*) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
EDiv	$\langle(\sigma \vdash e_1 \mathbin{/} e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle(\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (/) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
EMod	$\langle(\sigma \vdash e_1 \mathbin{\%} e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle(\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (\%) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
EEq	$\langle(\sigma \vdash e_1 == e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle(\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (==) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
ELt	$\langle(\sigma \vdash e_1 < e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle(\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (<) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
EVar	$\langle(\sigma \vdash \mathsf{var}\ x{=}e_1;\ e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle(\sigma \vdash e_1) :: \mathsf{def}[x][\sigma \vdash e_2] :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
EId	$\langle(\sigma \vdash x) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle\kappa \mathbin{\|} M(\sigma(x)) :: s \mathbin{\|} H \mathbin{\|} M\rangle$
EAssign	$\langle(\sigma \vdash x{=}e) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle(\sigma \vdash e) :: \mathsf{write}(\sigma(x)) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
ESeq	$\langle(\sigma \vdash e_1;\ e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$	$\rightarrow \langle(\sigma \vdash e_1) :: \mathsf{pop} :: (\sigma \vdash e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$

4.1.1 Conditionals and While Loops.

EIf $\quad\langle(\sigma \vdash \mathsf{if}\ (e_1)\ e_2\ \mathsf{else}\ e_3) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
$\rightarrow \quad\langle(\sigma \vdash e_1) :: \mathsf{jmp\text{-}if}[\langle(\sigma \vdash e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H\rangle] :: (\sigma \vdash e_3) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$

EWhile $\quad\langle(\sigma \vdash \mathsf{while}\ (e_1)\ e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
$\rightarrow \quad\langle(\sigma \vdash e_1) :: \mathsf{jmp\text{-}if}[\psi_{\mathrm{body}}] :: \kappa \mathbin{\|} \mathsf{undefined} :: s \mathbin{\|} H \mathbin{\|} M\rangle$

$$\text{where} \begin{cases} \psi_{\mathrm{body}} &= \langle(\sigma \vdash e_2) :: \mathsf{jmp}[\mathsf{continue}] :: \square \mathbin{\|} s \mathbin{\|} H_{\mathrm{body}}\rangle \\ H_{\mathrm{body}} &= H[\mathsf{continue} \mapsto \psi_{\mathrm{continue}}, \mathsf{break} \mapsto \psi_{\mathrm{break}}] \\ \psi_{\mathrm{continue}} &= \langle\mathsf{pop} :: (\sigma \vdash \mathsf{while}\ (e_1)\ e_2) :: \kappa \mathbin{\|} s \mathbin{\|} H\rangle \\ \psi_{\mathrm{break}} &= \langle\kappa \mathbin{\|} s \mathbin{\|} H\rangle \end{cases}$$

EBreak $\quad\langle(\sigma \vdash \mathsf{break}) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
$\rightarrow \quad\langle\mathsf{jmp}[\mathsf{break}] :: \square \mathbin{\|} \mathsf{undefined} :: s \mathbin{\|} H \mathbin{\|} M\rangle$

EContinue $\quad\langle(\sigma \vdash \mathsf{continue}) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
$\rightarrow \quad\langle\mathsf{jmp}[\mathsf{continue}] :: \square \mathbin{\|} \mathsf{undefined} :: s \mathbin{\|} H \mathbin{\|} M\rangle$

4.1.2 Functions and Return.

EFun $\quad\langle(\sigma \vdash \lambda(x, \ldots, x).e) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
$\rightarrow \quad\langle\kappa \mathbin{\|} \langle\lambda(x, \ldots, x).e, \sigma\rangle :: s \mathbin{\|} H \mathbin{\|} M\rangle$

EApp $\quad\langle(\sigma \vdash e(e_1, \ldots, e_n)) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
$\rightarrow \quad\langle(\sigma \vdash e) :: (\sigma \vdash e_1) :: \ldots :: (\sigma \vdash e_n) :: \mathsf{call}[n] :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$

EReturn $\quad\langle(\sigma \vdash \mathsf{return}\ e) :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$
$\rightarrow \quad\langle(\sigma \vdash e) :: \mathsf{return} :: \kappa \mathbin{\|} s \mathbin{\|} H \mathbin{\|} M\rangle$

4.1.3 Exceptions.

$$\begin{array}{ll} \text{ETry} & \langle(\sigma \vdash \text{try } e_1 \text{ catch } (x) \ e_2) :: \kappa \ || \ s \ || \ H \ || \ M\rangle \\ \rightarrow & \langle(\sigma \vdash e_1) :: \text{jmp[finally]} :: \Box \ || \ s \ || \ H_{\text{body}} \ || \ M\rangle \end{array}$$

$$\text{where} \left\{ \begin{array}{l} H_{\text{body}} = H[\text{throw} \mapsto \psi_{\text{throw}}, \text{finally} \mapsto \psi_{\text{finally}}] \\ \psi_{\text{throw}} = \langle \text{def}[x][\sigma \vdash e_2] :: \kappa \ || \ s \ || \ H\rangle \\ \psi_{\text{finally}} = \langle \kappa \ || \ s \ || \ H\rangle \end{array} \right.$$

$$\begin{array}{ll} \text{EThrow} & \langle(\sigma \vdash \text{throw } e) :: \kappa \ || \ s \ || \ H \ || \ M\rangle \\ \rightarrow & \langle(\sigma \vdash e) :: \text{jmp[throw]} :: \Box \ || \ s \ || \ H \ || \ M\rangle \end{array}$$

4.1.4 Generators.

$$\begin{array}{ll} \text{EGen} & \langle(\sigma \vdash \lambda{*}(x_1, \ldots, x_n).e) :: \kappa \ || \ s \ || \ H \ || \ M\rangle \\ \rightarrow & \langle\kappa \ || \ \langle\lambda{*}(x_1, \ldots, x_n).e, \sigma\rangle :: s \ || \ H \ || \ M\rangle \end{array}$$

$$\begin{array}{ll} \text{EIterNext}_1 & \langle(\sigma \vdash e_1.\text{next}()) :: \kappa \ || \ s \ || \ H \ || \ M\rangle \\ \rightarrow & \langle(\sigma \vdash e_1) :: (\sigma \vdash \text{undefined}) :: \text{next} :: \kappa \ || \ s \ || \ H \ || \ M\rangle \end{array}$$

$$\begin{array}{ll} \text{EIterNext}_2 & \langle(\sigma \vdash e_1.\text{next}(e_2)) :: \kappa \ || \ s \ || \ H \ || \ M\rangle \\ \rightarrow & \langle(\sigma \vdash e_1) :: (\sigma \vdash e_2) :: \text{next} :: \kappa \ || \ s \ || \ H \ || \ M\rangle \end{array}$$

$$\begin{array}{ll} \text{EYield} & \langle(\sigma \vdash \text{yield } e) :: \kappa \ || \ s \ || \ H \ || \ M\rangle \\ \rightarrow & \langle(\sigma \vdash e) :: \text{yield} :: \Box \ || \ \text{false} :: \psi_{\text{next}} :: s \ || \ H \ || \ M\rangle \end{array}$$

$$\text{where} \quad \psi_{\text{next}} = \langle\kappa \ || \ s \ || \ H\rangle$$

$$\begin{array}{ll} \text{EValueField} & \langle(\sigma \vdash e.\text{value}) :: \kappa \ || \ s \ || \ H \ || \ M\rangle \\ \rightarrow & \langle(\sigma \vdash e) :: \text{value} :: \kappa \ || \ s \ || \ H \ || \ M\rangle \end{array}$$

$$\begin{array}{ll} \text{EDoneField} & \langle(\sigma \vdash e.\text{done}) :: \kappa \ || \ s \ || \ H \ || \ M\rangle \\ \rightarrow & \langle(\sigma \vdash e) :: \text{done} :: \kappa \ || \ s \ || \ H \ || \ M\rangle \end{array}$$

4.2 Reduction Relations for Other Instructions

$$\begin{array}{llll} \text{IAdd} & \langle(+) :: \kappa \ || \ n_2 :: n_1 :: s \ || \ H \ || \ M\rangle & \rightarrow & \langle\kappa \ || \ (n_1 + n_2) :: s \ || \ H \ || \ M\rangle \\ \text{IMul} & \langle(*) :: \kappa \ || \ n_2 :: n_1 :: s \ || \ H \ || \ M\rangle & \rightarrow & \langle\kappa \ || \ (n_1 * n_2) :: s \ || \ H \ || \ M\rangle \\ \text{IDiv} & \langle(/) :: \kappa \ || \ n_2 :: n_1 :: s \ || \ H \ || \ M\rangle & \rightarrow & \langle\kappa \ || \ (n_1 / n_2) :: s \ || \ H \ || \ M\rangle & \text{if } n_2 \neq 0 \\ \text{IMod} & \langle(\%) :: \kappa \ || \ n_2 :: n_1 :: s \ || \ H \ || \ M\rangle & \rightarrow & \langle\kappa \ || \ (n_1 \% n_2) :: s \ || \ H \ || \ M\rangle & \text{if } n_2 \neq 0 \\ \text{IEq} & \langle(==) :: \kappa \ || \ v_2 :: v_1 :: s \ || \ H \ || \ M\rangle & \rightarrow & \langle\kappa \ || \ \text{eq}(v_1, v_2) :: s \ || \ H \ || \ M\rangle \\ \text{ILt} & \langle(<) :: \kappa \ || \ n_2 :: n_1 :: s \ || \ H \ || \ M\rangle & \rightarrow & \langle\kappa \ || \ (n_1 < n_2) :: s \ || \ H \ || \ M\rangle \end{array}$$

$$\text{where eq}(v_1, v_2) = \left\{ \begin{array}{llll} \text{true} & \text{if } v_1 = v_2 = \text{iter}[a] & \text{true} & \text{if } v_1 = v_2 = \text{undefined} \\ \text{true} & \text{if } v_1 = v_2 = n & \text{true} & \text{if } v_1 = v_2 = \{\text{value} : v, \text{done} : b\} \\ \text{true} & \text{if } v_1 = v_2 = b & \text{false} & \text{otherwise} \end{array} \right.$$

IDef $\langle \mathtt{def}[x_1, \ldots, x_n][\sigma \vdash e] :: \kappa \mid\mid v_n :: \ldots :: v_1 :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$ $\langle (\sigma[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n] \vdash e) :: \kappa \mid\mid s \mid\mid H \mid\mid M[a_1 \mapsto v_1, \ldots, a_n \mapsto v_n] \rangle$

where $\forall 1 \leq p \leq n.\ a_p \notin \text{Domain}(M) \wedge (\forall 1 \leq q < p.\ a_q \neq a_p)$

IWrite $\langle \mathtt{write}[a] :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$ $\langle \kappa \mid\mid v :: s \mid\mid H \mid\mid M[a \mapsto v] \rangle$

IPop $\langle \mathtt{pop} :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$ $\langle \kappa \mid\mid s \mid\mid H \mid\mid M \rangle$

4.2.1 Control Flow Instructions.

$\mathtt{IJmpIf_{true}}$ $\langle \mathtt{jmp\text{-}if}[\langle \kappa \mid\mid s \mid\mid H \rangle] :: \_ \mid\mid \mathtt{true} :: \_ \mid\mid \_ \mid\mid M \rangle \rightarrow \langle \kappa \mid\mid s \mid\mid H \mid\mid M \rangle$

$\mathtt{IJmpIf_{false}}$ $\langle \mathtt{jmp\text{-}if}[\_] :: \kappa \mid\mid \mathtt{false} :: s \mid\mid H \mid\mid M \rangle \qquad \rightarrow \langle \kappa \mid\mid s \mid\mid H \mid\mid M \rangle$

$\mathtt{IJmp}$ $\langle \mathtt{jmp}[c] :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M \rangle \qquad\qquad \rightarrow \langle \kappa' \mid\mid v :: s' \mid\mid H'' \mid\mid M \rangle$

where $\begin{cases} H(c) = \langle \kappa' \mid\mid s' \mid\mid H' \rangle \\ H'' = \begin{cases} H'[\mathtt{yield} \mapsto H(\mathtt{yield})] & \text{if } \mathtt{yield} \in \text{Domain}(H) \\ H' & \text{otherwise} \end{cases} \end{cases}$

4.2.2 Function Call/Return Instructions.

$\mathtt{ICall_\lambda}$ $\langle \mathtt{call}[n] :: \kappa \mid\mid v_n :: \ldots :: v_1 :: \langle \lambda(x_1, \ldots, x_m).e, \sigma' \rangle :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$ $\langle \mathtt{def}[x_1, \ldots, x_m][\sigma' \vdash \mathtt{return}\ e] :: \square \mid\mid s_{\text{body}} \mid\mid H_{\text{body}} \mid\mid M \rangle$

where $\begin{cases} s_{\text{body}} = \begin{cases} v_m :: \ldots :: v_1 :: \blacksquare & \text{if } n >= m \\ \mathtt{undefined} :: \ldots :: \mathtt{undefined} :: v_n :: \ldots :: v_1 :: \blacksquare & \text{otherwise} \end{cases} \\ H_{\text{body}} = H[\mathtt{return} \mapsto \psi_{\text{return}}] \setminus \{\mathtt{break}, \mathtt{continue}, \mathtt{yield}\} \\ \psi_{\text{return}} = \langle \kappa \mid\mid s \mid\mid H \rangle \end{cases}$

(Note that $A \setminus B$ denotes the removal of B's elements from A)

$\mathtt{ICall_{\lambda*}}$ $\langle \mathtt{call}[n] :: \kappa \mid\mid v_n :: \ldots :: v_1 :: \langle \lambda*(x_1, \ldots, x_m).e, \sigma' \rangle :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$ $\langle \kappa \mid\mid \mathtt{iter}[a] :: s \mid\mid H \mid\mid M[a \mapsto \psi_{\text{body}}] \rangle$

where $\begin{cases} a \notin \text{Domain}(M) \\ \psi_{\text{body}} = \langle \kappa_{\text{body}} \mid\mid s_{\text{body}} \mid\mid \varnothing \rangle \\ \kappa_{\text{body}} = \mathtt{pop} :: \mathtt{def}[x_1, \ldots, x_m][\sigma' \vdash \mathtt{return}\ (\mathtt{try}\ e\ \mathtt{catch}\ (x)\ x)] :: \square \\ s_{\text{body}} = \begin{cases} v_m :: \ldots :: v_1 :: \blacksquare & \text{if } n >= m \\ \mathtt{undefined} :: \ldots :: \mathtt{undefined} :: v_n :: \ldots :: v_1 :: \blacksquare & \text{otherwise} \end{cases} \\ x \text{ could be any identifier.} \end{cases}$

$\mathtt{IReturn}$ $\langle \mathtt{return} :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$ $\begin{cases} \langle \mathtt{yield} :: \square \mid\mid v :: \mathtt{true} :: \psi_{\text{done}} :: s \mid\mid H \mid\mid M \rangle & \text{if } \mathtt{yield} \in \text{Domain}(H) \\ \langle \mathtt{jmp}[\mathtt{return}] :: \square \mid\mid v :: \blacksquare \mid\mid H \mid\mid M \rangle & \text{otherwise} \end{cases}$

where $\psi_{\text{done}} = \langle \mathtt{return} :: \square \mid\mid \blacksquare \mid\mid \varnothing \rangle$

4.2.3 Generator Instructions.

$$\text{INext} \quad \langle \text{next} :: \kappa \,||\, v :: \text{iter}[a] :: s \,||\, H \,||\, M \rangle$$
$$\rightarrow \quad \langle \kappa' \,||\, v :: s' \,||\, H_{\text{body}} \,||\, M \rangle$$

$$\text{where} \begin{cases} M(a) = \langle \kappa' \,||\, s' \,||\, H' \rangle \\ H_{\text{body}} = H'[\text{yield} \mapsto \psi, \text{return} \mapsto \psi] \\ \psi \quad = \langle \kappa \,||\, \text{iter}[a] :: s \,||\, H \rangle \end{cases}$$

$$\text{IYield} \quad \langle \text{yield} :: \_ \,||\, v :: b :: v' :: \_ \,||\, H \,||\, M \rangle$$
$$\rightarrow \quad \langle \kappa' \,||\, \{\text{value} : v, \text{done} : b\} :: s' \,||\, H' \,||\, M[a \mapsto v'] \rangle$$

$$\text{where} \quad H(\text{yield}) = \langle \kappa' \,||\, \text{iter}[a] :: s' \,||\, H' \rangle$$

$$\text{IValueField} \quad \langle \text{value} :: \kappa \,||\, \{\text{value} : v, \text{done} : \_\} :: s \,||\, H \,||\, M \rangle$$
$$\rightarrow \quad \langle \kappa \,||\, v :: s \,||\, H \,||\, M \rangle$$

$$\text{IDoneField} \quad \langle \text{done} :: \kappa \,||\, \{\text{value} : \_, \text{done} : b\} :: s \,||\, H \,||\, M \rangle$$
$$\rightarrow \quad \langle \kappa \,||\, b :: s \,||\, H \,||\, M \rangle$$

And $\rightarrow^*$ is the reflexive-transitive closure of $\rightarrow$ and denotes the repeated reduction:

$$\langle \kappa \,||\, s \,||\, H \,||\, M \rangle \rightarrow^* \langle \kappa' \,||\, s' \,||\, H' \,||\, M' \rangle$$

$$\frac{\langle \kappa \,||\, s \,||\, H \,||\, M \rangle \rightarrow \langle \kappa' \,||\, s' \,||\, H' \,||\, M' \rangle \qquad \langle \kappa' \,||\, s' \,||\, H' \,||\, M' \rangle \rightarrow^* \langle \kappa'' \,||\, s'' \,||\, H'' \,||\, M'' \rangle}{\langle \kappa \,||\, s \,||\, H \,||\, M \rangle \rightarrow^* \langle \kappa'' \,||\, s'' \,||\, H'' \,||\, M'' \rangle}$$

The evaluation result of an expression e is the value v if

$$\langle (\varnothing \vdash e) :: \square \,||\, \blacksquare \,||\, \varnothing \,||\, \varnothing \rangle \rightarrow^* \langle \square \,||\, v :: \blacksquare \,||\, \_ \,||\, \_ \rangle$$