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 **Gen**erators and **E**xcep**t**ions, and it supports the following features:

- **undefined value** (`undefined`):
- **number (integer) 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`)
- **variable assignment** (`=`)
- **sequences** (`;`)
- **augmented assignment** (`+=`, `-=`, `*=`, `/=`, and `%=`)
- **increment/decrement** (`++` and `--`)
- **conditionals** (`if-else`)
- **while loops** (`while`)
- **loop controls** (`break` and `continue`)
- **first-class functions** (`=>` or `function`)
- **return** (`return`)
- **try-catch** (`try-catch`)
- **throw** (`throw`)
- **generators** (`=>*` or `function*`)
- **yield** (`yield`)
- **iterator operations** (`_.next`, `_.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	left
Additive	`+`, `-`	4	
Relational	`<`, `<=`, `>`, `>=`	5	
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:

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

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

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

$$
\begin{array}{ll}
\mathcal{D}[\![\texttt{-}\ e]\!] & = \mathcal{D}[\![e]\!] * (-1) \\
\mathcal{D}[\![e_1\ \texttt{-}\ e_2]\!] & = \mathcal{D}[\![e_1]\!] + \mathcal{D}[\![\texttt{-}\ e_2]\!] \\
\mathcal{D}[\![e_1\ \texttt{\&\&}\ e_2]\!] & = \texttt{if}\ (\mathcal{D}[\![e_1]\!])\ \mathcal{D}[\![e_2]\!]\ \texttt{else false} \\
\mathcal{D}[\![e_1\ \texttt{||}\ e_2]\!] & = \texttt{if}\ (\mathcal{D}[\![e_1]\!])\ \texttt{true else}\ \mathcal{D}[\![e_2]\!] \\
\mathcal{D}[\![\texttt{!}\ e]\!] & = \texttt{if}\ (\mathcal{D}[\![e]\!])\ \texttt{false else true} \\
\mathcal{D}[\![e_1\ \texttt{!=}\ e_2]\!] & = \mathcal{D}[\![\texttt{!}\ (e_1 == e_2)]\!] \\
\mathcal{D}[\![e_1\ \texttt{<=}\ e_2]\!] & = \texttt{var}\ \underline{x_1} = \mathcal{D}[\![e_1]\!]; \\
& \quad \texttt{var}\ \underline{x_2} = \mathcal{D}[\![e_2]\!]; \\
& \quad \mathcal{D}[\![(\underline{x_1} == \underline{x_2})\ \texttt{||}\ (\underline{x_1} < \underline{x_2})]\!] \\
\mathcal{D}[\![e_1 > e_2]\!] & = \mathcal{D}[\![\texttt{!}\ (e_1 \texttt{<=} e_2)]\!] \\
\mathcal{D}[\![e_1\ \texttt{>=}\ e_2]\!] & = \mathcal{D}[\![\texttt{!}\ (e_1 < e_2)]\!]
\end{array}
\qquad
\begin{array}{ll}
\mathcal{D}[\![x\ \texttt{+=}\ e]\!] & = x = x + \mathcal{D}[\![e]\!] \\
\mathcal{D}[\![x\ \texttt{-=}\ e]\!] & = x = x - \mathcal{D}[\![e]\!] \\
\mathcal{D}[\![x\ \texttt{*=}\ e]\!] & = x = x * \mathcal{D}[\![e]\!] \\
\mathcal{D}[\![x\ \texttt{/=}\ e]\!] & = x = x\ /\ \mathcal{D}[\![e]\!] \\
\mathcal{D}[\![x\ \texttt{\%=}\ e]\!] & = x = x\ \%\ \mathcal{D}[\![e]\!] \\
\mathcal{D}[\![\texttt{++}\ x]\!] & = \mathcal{D}[\![x\ \texttt{+=}\ 1]\!] \\
\mathcal{D}[\![\texttt{--}\ x]\!] & = \mathcal{D}[\![x\ \texttt{-=}\ 1]\!] \\
\mathcal{D}[\![x\ \texttt{++}]\!] & = \texttt{var}\ \underline{x_1} = x;\ \mathcal{D}[\![x\ \texttt{+=}\ 1]\!];\ \underline{x_1} \\
\mathcal{D}[\![x\ \texttt{--}]\!] & = \texttt{var}\ \underline{x_1} = x;\ \mathcal{D}[\![x\ \texttt{-=}\ 1]\!];\ \underline{x_1}
\end{array}
$$

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

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

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

4 SEMANTICS

We use the following notations in the semantics:

$$\begin{array}{llll}\text{States} & \langle \kappa \mid\mid s \mid\mid H \mid\mid M\rangle & \in \mathbb{K} \times \mathbb{S} \times \mathbb{H} \times \mathbb{M} & \texttt{(State)}\end{array}$$

$$\begin{array}{llll}\text{Continuations} & \kappa & \in \mathbb{K} & \texttt{(Cont)}\\ & \kappa & ::= \square \mid i :: \kappa & \end{array}$$

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

$$\begin{array}{llll}\text{Value Stacks} & s & \in \mathbb{S} & \texttt{(Stack)}\\ & s & ::= \blacksquare \mid v :: s & \end{array}$$

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

$$\begin{array}{llll}\text{Control Handlers} & H & \in \mathbb{H} = \mathbb{C} \xrightarrow{\text{fin}} \Psi & \texttt{(Handler)}\end{array}$$

$$\begin{array}{lllllll}\text{Control Operators} & c & \in \mathbb{C} & \texttt{(Control)} & & \\ & c & ::= \texttt{return} & \texttt{(Return)} & \mid \texttt{throw} & \texttt{(Throw)}\\ & & \mid \texttt{break} & \texttt{(Break)} & \mid \texttt{finally} & \texttt{(Finally)}\\ & & \mid \texttt{continue} & \texttt{(Continue)} & \mid \texttt{yield} & \texttt{(Yield)}\end{array}$$

$$\begin{array}{llll}\text{Continuation Values} & \psi, \langle \kappa \mid\mid s \mid\mid H\rangle & \in \Psi = \mathbb{K} \times \mathbb{S} \times \mathbb{H} & \texttt{(KValue)}\\ \text{Memories} & M & \in \mathbb{M} = \mathbb{A} \xrightarrow{\text{fin}} \mathbb{V} & \texttt{(Mem)}\\ \text{Environments} & \sigma & \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{A} & \texttt{(Env)}\\ \text{Addresses} & a & \in \mathbb{A} & \texttt{(Addr)}\end{array}$$

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

$$\boxed{\langle \kappa \mid\mid s \mid\mid H \mid\mid M \rangle \longrightarrow \langle \kappa \mid\mid s \mid\mid H \mid\mid M \rangle}$$

4.1 Reduction Relations for `IEval`

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

4.1.1 Conditionals and While Loops.

$$
\begin{array}{ll}
\texttt{EIf} & \langle (\sigma \vdash \texttt{if } (e_1)\ e_2 \texttt{ else } e_3) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle \\
\longrightarrow & \langle (\sigma \vdash e_1) :: \texttt{jmp-if[}\langle (\sigma \vdash e_2) :: \kappa \mid\mid s \mid\mid H \rangle\texttt{]} :: (\sigma \vdash e_3) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle
\end{array}
$$

$$
\begin{array}{ll}
\texttt{EWhile} & \langle (\sigma \vdash \texttt{while } (e_1)\ e_2) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle \\
\longrightarrow & \langle (\sigma \vdash e_1) :: \texttt{jmp-if[}\psi_{\text{body}}\texttt{]} :: \kappa \mid\mid \texttt{undefined} :: s \mid\mid H \mid\mid M \rangle \\
 & \\
 & \text{where} \begin{cases} \psi_{\text{body}} & = \langle (\sigma \vdash e_2) :: \texttt{jmp[continue]} :: \square \mid\mid s \mid\mid H_{\text{body}} \rangle \\ H_{\text{body}} & = H[\texttt{continue} \mapsto \psi_{\text{continue}}, \texttt{break} \mapsto \psi_{\text{break}}] \\ \psi_{\text{continue}} & = \langle \texttt{pop} :: (\sigma \vdash \texttt{while } (e_1)\ e_2) :: \kappa \mid\mid s \mid\mid H \rangle \\ \psi_{\text{break}} & = \langle \kappa \mid\mid s \mid\mid H \rangle \end{cases}
\end{array}
$$

$$
\begin{array}{ll}
\texttt{EBreak} & \langle (\sigma \vdash \texttt{break}) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle \\
\longrightarrow & \langle \texttt{jmp[break]} :: \square \mid\mid \texttt{undefined} :: s \mid\mid H \mid\mid M \rangle
\end{array}
$$

$$
\begin{array}{ll}
\texttt{EContinue} & \langle (\sigma \vdash \texttt{continue}) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle \\
\longrightarrow & \langle \texttt{jmp[continue]} :: \square \mid\mid \texttt{undefined} :: s \mid\mid H \mid\mid M \rangle
\end{array}
$$

4.1.2 Functions and Return.

$$
\begin{array}{ll}
\texttt{EFun} & \langle (\sigma \vdash \lambda(x, \ldots, x).e) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle \\
\longrightarrow & \langle \kappa \mid\mid \langle \lambda(x, \ldots, x).e, \sigma \rangle :: s \mid\mid H \mid\mid M \rangle
\end{array}
$$

$$
\begin{array}{ll}
\texttt{EApp} & \langle (\sigma \vdash e(e_1, \ldots, e_n)) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle \\
\longrightarrow & \langle (\sigma \vdash e) :: (\sigma \vdash e_1) :: \ldots :: (\sigma \vdash e_n) :: \texttt{call[}n\texttt{]} :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle
\end{array}
$$

$$
\begin{array}{ll}
\texttt{EReturn} & \langle (\sigma \vdash \texttt{return } e) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle \\
\longrightarrow & \langle (\sigma \vdash e) :: \texttt{return} :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle
\end{array}
$$

4.1.3 Exceptions.

$$
\begin{array}{ll}
\texttt{ETry} & \langle(\sigma \vdash \texttt{try}\ e_1\ \texttt{catch}\ (x)\ e_2) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle(\sigma \vdash e_1) :: \texttt{jmp[finally]} :: \square \mid\mid s \mid\mid H_{\text{body}} \mid\mid M\rangle \\
& \\
& \text{where} \begin{cases} H_{\text{body}} & = H[\texttt{throw} \mapsto \psi_{\text{throw}}, \texttt{finally} \mapsto \psi_{\text{finally}}] \\ \psi_{\text{throw}} & = \langle \texttt{def}[x][\sigma \vdash e_2] :: \kappa \mid\mid s \mid\mid H\rangle \\ \psi_{\text{finally}} & = \langle \kappa \mid\mid s \mid\mid H\rangle \end{cases}
\end{array}
$$

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

4.1.4 Generators.

$$
\begin{array}{ll}
\texttt{EGen} & \langle(\sigma \vdash \lambda{*}(x_1, \ldots, x_n).e) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle \kappa \mid\mid \langle \lambda{*}(x_1, \ldots, x_n).e, \sigma\rangle :: s \mid\mid H \mid\mid M\rangle \\
& \\
\texttt{EIterNext}_1 & \langle(\sigma \vdash e_1\texttt{.next}()) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle(\sigma \vdash e_1) :: (\sigma \vdash \texttt{undefined}) :: \texttt{next} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
& \\
\texttt{EIterNext}_2 & \langle(\sigma \vdash e_1\texttt{.next}(e_2)) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle(\sigma \vdash e_1) :: (\sigma \vdash e_2) :: \texttt{next} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
& \\
\texttt{EYield} & \langle(\sigma \vdash \texttt{yield}\ e) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle(\sigma \vdash e) :: \texttt{yield} :: \square \mid\mid \texttt{false} :: \psi_{\text{next}} :: s \mid\mid H \mid\mid M\rangle \\
& \\
& \text{where} \quad \psi_{\text{next}} = \langle \kappa \mid\mid s \mid\mid H\rangle \\
& \\
\texttt{EValueField} & \langle(\sigma \vdash e\texttt{.value}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle(\sigma \vdash e) :: \texttt{value} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
& \\
\texttt{EDoneField} & \langle(\sigma \vdash e\texttt{.done}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle(\sigma \vdash e) :: \texttt{done} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle
\end{array}
$$

4.2 Reduction Relations for Other Instructions

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

$$
\text{where } \mathsf{eq}(v_1, v_2) = \begin{cases}
\texttt{true} & \text{if } v_1 = v_2 = \texttt{iter}[a] \qquad \texttt{true} \quad \text{if } v_1 = v_2 = \texttt{undefined} \\
\texttt{true} & \text{if } v_1 = v_2 = n \qquad\qquad\ \texttt{true} \quad \text{if } v_1 = v_2 = \{\texttt{value} : v, \texttt{done} : b\} \\
\texttt{true} & \text{if } v_1 = v_2 = b \qquad\qquad\ \texttt{false} \quad \text{otherwise}
\end{cases}
$$

$$\begin{array}{ll} \texttt{IDef} & \langle \texttt{def}[x_1,\ldots,x_n][\sigma \vdash e] :: \kappa \,||\, v_n :: \ldots :: v_1 :: s \,||\, H \,||\, M\rangle \\ \rightarrow & \langle (\sigma[x_1 \mapsto a_1,\ldots,x_n \mapsto a_n] \vdash e) :: \kappa \,||\, s \,||\, H \,||\, M[a_1 \mapsto v_1,\ldots,a_n \mapsto v_n]\rangle \\ & \\ & \text{where } \forall 1 \le p \le n.\ a_p \notin \text{Domain}(M) \wedge (\forall 1 \le q < p.\ a_q \ne a_p) \end{array}$$

$$\texttt{IWrite} \quad \langle \texttt{write}[a] :: \kappa \,||\, v :: s \,||\, H \,||\, M\rangle \quad \rightarrow \quad \langle \kappa \,||\, v :: s \,||\, H \,||\, M[a \mapsto v]\rangle$$

$$\texttt{IPop} \quad \langle \texttt{pop} :: \kappa \,||\, v :: s \,||\, H \,||\, M\rangle \quad \rightarrow \quad \langle \kappa \,||\, s \,||\, H \,||\, M\rangle$$

4.2.1 Control Flow Instructions.

$$\begin{array}{lll} \texttt{IJmpIf}_{\text{true}} & \langle \texttt{jmp-if}[\langle \kappa' \,||\, s' \,||\, H'\rangle] :: \_ \,||\, \texttt{true} :: \_ \,||\, \_ \,||\, M\rangle & \rightarrow \langle \kappa' \,||\, s' \,||\, H' \,||\, M\rangle \\ & & \\ \texttt{IJmpIf}_{\text{false}} & \langle \texttt{jmp-if}[\_] :: \kappa \,||\, \texttt{false} :: s \,||\, H \,||\, M\rangle & \rightarrow \langle \kappa \,||\, s \,||\, H \,||\, M\rangle \\ & & \\ \texttt{IJmp} & \langle \texttt{jmp}[c] :: \kappa \,||\, v :: s \,||\, H \,||\, M\rangle & \rightarrow \langle \kappa' \,||\, v :: s' \,||\, H'' \,||\, M\rangle \end{array}$$

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

4.2.2 Function Call/Return Instructions.

$$\begin{array}{ll} \texttt{ICall}_{\lambda} & \langle \texttt{call}[n] :: \kappa \,||\, v_n :: \ldots :: v_1 :: \langle \lambda(x_1,\ldots,x_m).e, \sigma'\rangle :: s \,||\, H \,||\, M\rangle \\ \rightarrow & \langle \texttt{def}[x_1,\ldots,x_m][\sigma' \vdash \texttt{return}\ e] :: \square \,||\, s_{\text{body}} \,||\, H_{\text{body}} \,||\, M\rangle \end{array}$$

$$\text{where} \begin{cases} v'_p & = \begin{cases} v_p & \text{if } p \le n \\ \texttt{undefined} & \text{otherwise} \end{cases} \quad \text{for all } 1 \le p \le m \\ s_{\text{body}} & = v'_m :: \ldots :: v'_1 :: \blacksquare \\ H_{\text{body}} & = H[\texttt{return} \mapsto \psi_{\text{return}}] \setminus \{\texttt{break}, \texttt{continue}, \texttt{yield}\} \\ \psi_{\text{return}} & = \langle \kappa \,||\, s \,||\, H\rangle \end{cases}$$

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

$$\begin{array}{ll} \texttt{ICall}_{\lambda*} & \langle \texttt{call}[n] :: \kappa \,||\, v_n :: \ldots :: v_1 :: \langle \lambda*(x_1,\ldots,x_m).e, \sigma'\rangle :: s \,||\, H \,||\, M\rangle \\ \rightarrow & \langle \kappa \,||\, \texttt{iter}[a] :: s \,||\, H \,||\, M[a \mapsto \psi_{\text{body}}]\rangle \end{array}$$

$$\text{where} \begin{cases} a & \notin \text{Domain}(M) \\ \psi_{\text{body}} & = \langle \kappa_{\text{body}} \,||\, s_{\text{body}} \,||\, \varnothing\rangle \\ \kappa_{\text{body}} & = \texttt{pop} :: \texttt{def}[x_1,\ldots,x_m][\sigma' \vdash \texttt{return}\ (\texttt{try}\ e\ \texttt{catch}\ (x)\ x)] :: \square \\ v'_p & = \begin{cases} v_p & \text{if } p \le n \\ \texttt{undefined} & \text{otherwise} \end{cases} \quad \text{for all } 1 \le p \le m \\ s_{\text{body}} & = v'_m :: \ldots :: v'_1 :: \blacksquare \\ x & \text{could be any identifier.} \end{cases}$$

$$\begin{array}{ll} \texttt{IReturn} & \langle \texttt{return} :: \kappa \,||\, v :: s \,||\, H \,||\, M\rangle \\ \rightarrow & \begin{cases} \langle \texttt{yield} :: \square \,||\, v :: \texttt{true} :: \psi_{\text{done}} :: s \,||\, H \,||\, M\rangle & \text{if } \texttt{yield} \in \text{Domain}(H) \\ \langle \texttt{jmp}[\texttt{return}] :: \square \,||\, v :: \blacksquare \,||\, H \,||\, M\rangle & \text{otherwise} \end{cases} \\ & \text{where} \quad \psi_{\text{done}} = \langle \texttt{return} :: \square \,||\, \blacksquare \,||\, \varnothing\rangle \end{array}$$

4.2.3 Generator Instructions.

$$
\begin{array}{ll}
\texttt{INext} & \langle \texttt{next} :: \kappa \mid\mid v :: \texttt{iter[}a\texttt{]} :: s \mid\mid H \mid\mid M \rangle \\
\rightarrow & \langle \kappa' \mid\mid v :: s' \mid\mid H_{\text{body}} \mid\mid M \rangle \\
& \\
& \text{where} \begin{cases} M(a) & = \langle \kappa' \mid\mid s' \mid\mid H' \rangle \\ H_{\text{body}} & = H'[\texttt{yield} \mapsto \psi, \texttt{return} \mapsto \psi] \\ \psi & = \langle \kappa \mid\mid \texttt{iter[}a\texttt{]} :: s \mid\mid H \rangle \end{cases} \\
& \\
\texttt{IYield} & \langle \texttt{yield} :: \_ \mid\mid v :: b :: v' :: \_ \mid\mid H \mid\mid M \rangle \\
\rightarrow & \langle \kappa' \mid\mid \{\texttt{value} : v, \texttt{done} : b\} :: s' \mid\mid H' \mid\mid M[a \mapsto v'] \rangle \\
& \\
& \text{where} \quad H(\texttt{yield}) = \langle \kappa' \mid\mid \texttt{iter[}a\texttt{]} :: s' \mid\mid H' \rangle \\
& \\
\texttt{IValueField} & \langle \texttt{value} :: \kappa \mid\mid \{\texttt{value} : v, \texttt{done} : \_\} :: s \mid\mid H \mid\mid M \rangle \\
\rightarrow & \langle \kappa \mid\mid v :: s \mid\mid H \mid\mid M \rangle \\
& \\
\texttt{IDoneField} & \langle \texttt{done} :: \kappa \mid\mid \{\texttt{value} : \_, \texttt{done} : b\} :: s \mid\mid H \mid\mid M \rangle \\
\rightarrow & \langle \kappa \mid\mid b :: s \mid\mid H \mid\mid M \rangle
\end{array}
$$

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

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

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

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

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