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 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 and binary operators
         | <uop> <expr> | <expr> <bop> <expr>
         // parentheses
         | "(" <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
         | <params> "=>" <expr> | "function" <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
         | <params> "=>" "*" <expr>
         | "function" "*" <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:

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

3 ABSTRACT SYNTAX

The abstract syntax of MAGNET is defined as follows:

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

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

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

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

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

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

4.1 Reduction Relations for `IEval`

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

4.1.1 Conditionals and While Loops.

`EIf` $\langle(\sigma \vdash \texttt{if}\ (e_1)\ e_2\ \texttt{else}\ e_3) :: \kappa \parallel s \parallel H \parallel M\rangle$

$\rightarrow$ $\langle(\sigma \vdash e_1) :: \texttt{jmp-if[}\langle(\sigma \vdash e_2) :: \kappa \parallel s \parallel H\rangle\texttt{]} :: (\sigma \vdash e_3) :: \kappa \parallel s \parallel H \parallel M\rangle$

`EWhile` $\langle(\sigma \vdash \texttt{while}\ (e_1)\ e_2) :: \kappa \parallel s \parallel H \parallel M\rangle$

$\rightarrow$ $\langle(\sigma \vdash e_1) :: \texttt{jmp-if[}\psi_{\text{body}}\texttt{]} :: \kappa \parallel \texttt{undefined} :: s \parallel H \parallel M\rangle$

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

`EBreak` $\langle(\sigma \vdash \texttt{break}) :: \kappa \parallel s \parallel H \parallel M\rangle$

$\rightarrow$ $\langle\texttt{jmp[break]} :: \square \parallel \texttt{undefined} :: s \parallel H \parallel M\rangle$

`EContinue` $\langle(\sigma \vdash \texttt{continue}) :: \kappa \parallel s \parallel H \parallel M\rangle$

$\rightarrow$ $\langle\texttt{jmp[continue]} :: \square \parallel \texttt{undefined} :: s \parallel H \parallel M\rangle$

4.1.2 Functions and Return.

`EFun` $\langle(\sigma \vdash \lambda(x, \ldots, x).e) :: \kappa \parallel s \parallel H \parallel M\rangle$

$\rightarrow$ $\langle\kappa \parallel \langle\lambda(x, \ldots, x).e, \sigma\rangle :: s \parallel H \parallel M\rangle$

`EApp` $\langle(\sigma \vdash e(e_1, \ldots, e_n)) :: \kappa \parallel s \parallel H \parallel M\rangle$

$\rightarrow$ $\langle(\sigma \vdash e) :: (\sigma \vdash e_1) :: \ldots :: (\sigma \vdash e_n) :: \texttt{call[}n\texttt{]} :: \kappa \parallel s \parallel H \parallel M\rangle$

`EReturn` $\langle(\sigma \vdash \texttt{return}\ e) :: \kappa \parallel s \parallel H \parallel M\rangle$

$\rightarrow$ $\langle(\sigma \vdash e) :: \texttt{return} :: \kappa \parallel s \parallel H \parallel M\rangle$

4.1.3 *Exceptions.*

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

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

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

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

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

$$\begin{array}{ll}\texttt{IWrite} & \langle \texttt{write}[a] :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle \kappa \mid\mid v :: s \mid\mid H \mid\mid M[a \mapsto v]\rangle\end{array}$$

$$\begin{array}{ll}\texttt{IPop} & \langle \mathsf{pop} :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle \kappa \mid\mid s \mid\mid H \mid\mid M\rangle\end{array}$$

4.2.1 *Control Flow Instructions.*

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

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

4.2.2 *Function Call/Return Instructions.*

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

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

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

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

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

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

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