MiniPython – A Small Subset of Python Language

1 INTRODUCTION

MiniPython is a toy language for the COSE212 course at Korea University. MiniPython is inspired by the Python programming language.[1] The name MiniPython indicates that it is a minimal subset of the Python language, and it supports the following features:

- **pass statements** (`pass`)
- **variable assignment statement** (`x = e`)
- **set item** (`x[e] = e`)
- **if-statements** (`if`-`elif`-`else`)
- **while-statements** (`while`)
- **for-statements** (`for`-`in`)
- **loop control statements** (`break` and `continue`)
- **try and raise statements** (`try`-`except` and `raise`)
- **function definitions** (`def f(...):`)
- **return statements** (`return`)
- **yield and yield from statements** (`yield` and `yield from`)
- **none value** (`None`)
- **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 (`and`), disjunction (`or`), and negation (`not`)
- **comparison operators**: equality (`==`, `!=`, `is`, and `is not`) and relational (`<`, `>`, `<=`, and `>=`)
- **lists** (`[e, ..., e]`)
- **list append method** (`e.append(e)`)
- **get item** (`x[e]`)
- **lambda functions** (`lambda ...: e`)
- **function applications** (`e(...)`)
- **conditional expressions** (`e if e else e`)
- **iter function** (`iter(e)`)
- **next function** (`next(e)`)

This document is the specification of MiniPython. 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 MiniPython.

2 CONCRETE SYNTAX

The concrete syntax of MiniPython 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. The `<INDENT>` and `<DEDENT>` denote the increase and decrease of the indentation level, respectively.

[1] https://www.python.org/

```
// basic elements
<digit>    ::= "0" | "1" | "2" | ... | "9"
<number>   ::= "-"? <digit>+
<alphabet> ::= "A" | "B" | "C" | ... | "Z" | "a" | "b" | "c" | ... | "z"
<idstart>  ::= <alphabet> | "_"
<idcont>   ::= <alphabet> | "_" | <digit>
<keyword>  ::= "False" | "None" | "True" | "and" | "break" | "continue" | "def"
             | "elif" | "else" | "except" | "for" | "from" | "if" | "in"
             | "lambda" | "not" | "or" | "pass" | "raise" | "return" | "try"
             | "while" | "yield"
<id>       ::= <idstart> <idcont>* butnot <keyword>
<program>  ::= <stmt>* <expr> // programs
<block>    ::= <INDENT> <stmt>+ <DEDENT> | <stmt> // blocks
// statements
<stmt> ::= "pass" | <expr> | <id> "=" <expr> | <expr> "[" <expr> "]" "=" <expr>
         // if-statements
         | "if" <expr> ":" <block> [ "elif" <expr> ":" <block> ]*
           [ "else" ":" <block> ]
         // while-statements and for-statements
         | "while" <expr> ":" <block> | "for" <id> "in" <expr> ":" <block>
         // loop control statements, try statements, and raise statements
         | "break" | "continue" | "try" ":" <block> "except" ":" <block>
         // function definitions
         | "def" <id> "(" [ <id> [ "," <id> ]* ] ")" ":" <block>
         // raise, return, yield, and yield from statements
         | "raise" | "return" <expr> | "yield" <expr> | "yield" "from" <expr>
// expressions
<expr> ::= "None" | <number> | "True" | "False" | <id>
         // unary/binary operators and parentheses
         | <uop> <expr> | <expr> <bop> <expr> | "(" <expr> ")"
         // lists
         | "[" [ <expr> [ "," <expr> ]* ] "]"
         // list append method and get item
         | <expr> "." "append" "(" <expr> ")" | <expr> "[" <expr> "]"
         // lambda functions
         | "lambda" [ <id> [ "," <id> ]* ] ":" <expr>
         // function applications
         | <expr> "(" [ <expr> [ "," <expr> ]* ] ")"
         // conditional expressions
         | <expr> "if" <expr> "else" <expr>
         // iter function and next function
         | "iter" "(" <expr> ")" | "next" "(" <expr> ")"
// operators
<uop> ::= "-" | "not"
<bop> ::= "+" | "-" | "*" | "//" | "%" | "and" | "or"
        | "==" | "!=" | "is" | "is not" | "<" | "<=" | ">" | ">="
```

The precedence and associativity of operators are defined as follows:

Description	Operator	Precedence	Associativity
Method Call	`_.append`	10	left
Prefix Unary	`-`	9	right
Multiplicative	`*`, `//`, `%`	8	left
Additive	`+`, `-`	7	
Comparison	`==`, `!=`, `is`, `is not`, `<`, `<=`, `>`, `>=`	6	
Logical Negation	`not`	3	
Logical Conjunction	`and`	4	
Logical Disjunction	`or`	3	
Conditional Expression	`if-else`	2	-
Lambda Function	`lambda`	1	-

3 ABSTRACT SYNTAX

The abstract syntax of MiniPython is defined as follows:

$$\text{Programs} \quad p ::= S; \ldots; S; e \quad (\texttt{Program}) \qquad \text{Blocks} \quad B ::= S \mid \{S; \ldots; S\} \quad (\texttt{Block})$$

$$\begin{array}{llll}
\text{Statements} & S ::= \texttt{pass} & (\texttt{SPass}) & \mid \texttt{continue} & (\texttt{SContinue}) \\
& \mid e & (\texttt{SExpr}) & \mid \texttt{try}\ B\ \texttt{except}\ B & (\texttt{STry}) \\
& \mid x = e & (\texttt{SAssign}) & \mid \texttt{raise} & (\texttt{SRaise}) \\
& \mid e[e] = e & (\texttt{SSetItem}) & \mid \texttt{def}\ x\ (x, \ldots, x)\ B & (\texttt{SDef}) \\
& \mid \texttt{if}\ e\ B\ \texttt{else}\ B & (\texttt{SIf}) & \mid \texttt{return}\ e & (\texttt{SReturn}) \\
& \mid \texttt{while}\ e\ B & (\texttt{SWhile}) & \mid \texttt{yield}\ e & (\texttt{SYield}) \\
& \mid \texttt{break} & (\texttt{SBreak}) & &
\end{array}$$

$$\begin{array}{llll}
\text{Expressions} & e ::= \texttt{None} & (\texttt{ENone}) & \mid e[e] & (\texttt{EGetItem}) \\
& \mid n & (\texttt{ENum}) & \mid \lambda(x, \ldots, x).e & (\texttt{ELambda}) \\
& \mid b & (\texttt{EBool}) & \mid e(e, \ldots, e) & (\texttt{EApp}) \\
& \mid x & (\texttt{EId}) & \mid e\ \texttt{if}\ e\ \texttt{else}\ e & (\texttt{ECond}) \\
& \mid e \oplus e & (\texttt{EBOp}) & \mid \texttt{iter}(e) & (\texttt{EIter}) \\
& \mid [e, \ldots, e] & (\texttt{EList}) & \mid \texttt{next}(e) & (\texttt{ENext}) \\
& \mid e.\text{append}(e) & (\texttt{EAppend}) & &
\end{array}$$

$$\text{Operators} \ \oplus \ ::= \texttt{+}\ (\texttt{Add}) \mid \texttt{*}\ (\texttt{Mul}) \mid \texttt{//}\ (\texttt{Div}) \mid \texttt{\%}\ (\texttt{Mod}) \mid \texttt{<}\ (\texttt{Lt}) \mid \texttt{<=}\ (\texttt{Lte}) \mid \texttt{==}\ (\texttt{Eq}) \mid \texttt{is}\ (\texttt{Is})$$

$$\text{where} \left\{ \text{Booleans} \quad \mathbb{B} \ni b ::= \texttt{True} \mid \texttt{False} \quad (\texttt{Boolean}) \right. \qquad \begin{array}{lrl} \text{Identifiers} & x \in \mathbb{X} & (\texttt{String}) \\ \text{Numbers} & n, m, k \in \mathbb{Z} & (\texttt{BigInt}) \end{array}$$

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

$$\begin{array}{ll|ll}
\mathcal{D}[\![- e]\!] & = \mathcal{D}[\![e]\!] * (-1) & \mathcal{D}[\![e_1\ \texttt{is not}\ e_2]\!] & = \mathcal{D}[\![\texttt{not}\ (e_1\ \texttt{is}\ e_2)]\!] \\
\mathcal{D}[\![e_1 - e_2]\!] & = \mathcal{D}[\![e_1]\!] + \mathcal{D}[\![- e_2]\!] & \mathcal{D}[\![e_1\ \texttt{and}\ e_2]\!] & = \texttt{if}\ (\mathcal{D}[\![e_1]\!])\ \mathcal{D}[\![e_2]\!]\ \texttt{else False} \\
\mathcal{D}[\![e_1 > e_2]\!] & = \mathcal{D}[\![\texttt{not}\ (e_1 <= e_2)]\!] & \mathcal{D}[\![e_1\ \texttt{or}\ e_2]\!] & = \texttt{if}\ (\mathcal{D}[\![e_1]\!])\ \texttt{True else}\ \mathcal{D}[\![e_2]\!] \\
\mathcal{D}[\![e_1 >= e_2]\!] & = \mathcal{D}[\![\texttt{not}\ (e_1 < e_2)]\!] & \mathcal{D}[\![\texttt{not}\ e]\!] & = \texttt{if}\ (\mathcal{D}[\![e]\!])\ \texttt{False else True} \\
\mathcal{D}[\![e_1\ != e_2]\!] & = \mathcal{D}[\![\texttt{not}\ (e_1 == e_2)]\!] & \mathcal{D}[\![\texttt{yield from}\ e]\!] & = \mathcal{D}[\![\texttt{for}\ x\ \texttt{in}\ e\ \texttt{yield}\ x]\!]
\end{array}$$

$$\mathcal{D}[\![\texttt{for}\ x\ \texttt{in}\ e\ B]\!] = \mathcal{D}[\![\underline{x} = e\ ;\ \texttt{while True}\ \{\texttt{try}\ x = \texttt{next}(\underline{x})\ \texttt{except break}\ ;\ B\}]\!]$$

where $\underline{x}$ 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:

$$\text{States}\quad \langle \kappa \mid\mid s \mid\mid H \mid\mid M \rangle \in \mathbb{K} \times \mathbb{S} \times \mathbb{H} \times \mathbb{M}\ \ (\texttt{State}) \qquad \text{Continuations}\quad \mathbb{K} \ni \kappa\ ::= \square \mid i :: \kappa\ (\texttt{Cont})$$

$$\begin{array}{llll}
\text{Instructions}\quad i ::= (\sigma \vdash_B B) & (\texttt{IBlock}) & \mid \texttt{jmp-if}[\psi] & (\texttt{IJmpIf}) \\
\mid (\sigma \vdash_S S) & (\texttt{IStmt}) & \mid \texttt{jmp}[c] & (\texttt{IJmp}) \\
\mid (\sigma \vdash_e e) & (\texttt{IExpr}) & \mid \texttt{raise}(\omega) & (\texttt{IRaise}) \\
\mid (\oplus) & (\texttt{IBop}) & \mid \texttt{call}[n] & (\texttt{ICall}) \\
\mid \texttt{write}[a] & (\texttt{IWrite}) & \mid \texttt{return} & (\texttt{IReturn}) \\
\mid \texttt{get-item} & (\texttt{IGetItem}) & \mid \texttt{yield} & (\texttt{IYield}) \\
\mid \texttt{set-item} & (\texttt{ISetItem}) & \mid \texttt{iter} & (\texttt{IIter}) \\
\mid \texttt{list}[n] & (\texttt{IList}) & \mid \texttt{next} & (\texttt{INext}) \\
\mid \texttt{append} & (\texttt{IAppend}) & \mid \texttt{drop} & (\texttt{IDrop})
\end{array}$$

$$\text{Value Stacks}\quad s ::= \blacksquare \mid v :: s \quad (\texttt{Stack})$$

$$\begin{array}{llll}
\text{Control Handlers} & H \in \mathbb{H} = \mathbb{C} \xrightarrow{\text{fin}} \Psi\ (\texttt{Handler}) & \text{Environments} & \sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{A}\ \ (\texttt{Env}) \\
\text{Memories} & M \in \mathbb{M} = \mathbb{A} \xrightarrow{\text{fin}} \mathbb{V}\ (\texttt{Mem}) & \text{Addresses} & a \in \mathbb{A} \quad\quad (\texttt{Addr})
\end{array}$$

$$\begin{array}{ll}
\text{Errors}\quad \omega ::= & \texttt{RuntimeError} \mid \texttt{ZeroDivisionError} \mid \texttt{TypeError} \mid \texttt{IndexError} \\
& \mid \texttt{StopIteration} \mid \texttt{NameError}[x]
\end{array}$$

$$\begin{array}{llll}
\text{Values}\quad v ::= \texttt{None} & (\texttt{NoneV}) & \mid \langle \lambda(x, \ldots, x).B, \sigma \rangle & (\texttt{CloV}) \\
\mid n & (\texttt{NumV}) & \mid \langle \lambda{*}(x, \ldots, x).B, \sigma \rangle & (\texttt{GenV}) \\
\mid b & (\texttt{BoolV}) & \mid \langle \kappa \mid\mid s \mid\mid H \rangle & (\texttt{ContV}) \\
\mid a & (\texttt{AddrV}) & \mid \texttt{iter}[a, n] & (\texttt{IterV}) \\
\mid [v, \ldots, v] & (\texttt{ListV}) & &
\end{array}$$

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

$$\begin{array}{llll}
\text{Controls}\quad c ::= \texttt{return} & (\texttt{Return}) & \mid \texttt{throw} & (\texttt{Throw}) \\
\mid \texttt{break} & (\texttt{Break}) & \mid \texttt{finally} & (\texttt{Finally}) \\
\mid \texttt{continue} & (\texttt{Continue}) & \mid \texttt{yield} & (\texttt{Yield})
\end{array}$$

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

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

The evaluation result of a program $p = S_1\,; \ldots ; S_n\,; e$ is the value v if

$$\langle (\sigma \vdash S_1) :: \ldots :: (\sigma \vdash S_n) :: (\sigma \vdash e) :: \square \mid\mid \blacksquare \mid\mid \varnothing \mid\mid \varnothing \rangle \rightarrow^* \langle \square \mid\mid v :: \blacksquare \mid\mid \_ \mid\mid \_ \rangle$$

$$\text{where} \begin{cases} \{x_1, \ldots, x_k\} & = \texttt{locals}(\{S_1, \ldots, S_n\}) \\ a_1, \ldots, a_k & = \text{(distinct fresh addresses)} \\ \sigma & = [x_1 \mapsto a_1, \ldots, x_k \mapsto a_k] \\ M & = [a_1 \mapsto \texttt{None}, \ldots, a_k \mapsto \texttt{None}] \end{cases}$$

4.1 Reduction Relations for `IStmt`

$$\texttt{SPass} \quad \langle(\sigma \vdash_S \texttt{pass}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$

$$\texttt{SExpr} \quad \langle(\sigma \vdash_S e) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle(\sigma \vdash_e e) :: \texttt{drop} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$

$$\texttt{SAssign} \quad \langle(\sigma \vdash_S x = e) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle(\sigma \vdash_e e) :: \texttt{write[}\sigma(x)\texttt{]} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$

$$\texttt{SSetItem} \quad \langle(\sigma \vdash_S e_0\texttt{[}e_1\texttt{]} = e_2) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle(\sigma \vdash_e e_2) :: (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \texttt{set-item} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$

$$\texttt{SIf} \quad \langle(\sigma \vdash_S \texttt{if } e\ B_0 \texttt{ else } B_1) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle(\sigma \vdash_e e) :: \texttt{jmp-if[}\langle(\sigma \vdash_B B_0) :: \kappa \mid\mid s \mid\mid H\rangle\texttt{]} :: (\sigma \vdash_B B_1) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$

$$\texttt{SWhile} \quad \langle(\sigma \vdash_S \texttt{while } e\ B) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle(\sigma \vdash_e e) :: \texttt{jmp-if[}\psi_{\text{body}}\texttt{]} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\text{where} \begin{cases} \psi_{\text{body}} &= \langle(\sigma \vdash_B B) :: (\sigma \vdash_S \texttt{while } e\ B) :: \kappa \mid\mid s \mid\mid H_{\text{body}}\rangle \\ H_{\text{body}} &= H[\texttt{continue} \mapsto \psi_{\text{continue}}, \texttt{break} \mapsto \psi_{\text{break}}] \\ \psi_{\text{continue}} &= \langle(\sigma \vdash_S \texttt{while } e\ B) :: \kappa \mid\mid s \mid\mid H\rangle \\ \psi_{\text{break}} &= \langle \kappa \mid\mid s \mid\mid H\rangle \end{cases}$$

$$\texttt{SBreak} \quad \langle(\sigma \vdash_S \texttt{break}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle\texttt{jmp[break]} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$

$$\texttt{SContinue} \quad \langle(\sigma \vdash_S \texttt{continue}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle\texttt{jmp[break]} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$

$$\texttt{STry} \quad \langle(\sigma \vdash_S \texttt{try } B_0 \texttt{ except } B_1) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle(\sigma \vdash_B B_0) :: \texttt{jmp[finally]} :: \square \mid\mid s \mid\mid H_{\text{body}} \mid\mid M\rangle$$
$$\text{where} \begin{cases} H_{\text{body}} &= H[\texttt{raise} \mapsto \psi_{\text{raise}}, \texttt{finally} \mapsto \psi_{\text{finally}}] \\ \psi_{\text{raise}} &= \langle(\sigma \vdash_B B_1) :: \kappa \mid\mid s \mid\mid H\rangle \\ \psi_{\text{finally}} &= \langle \kappa \mid\mid s \mid\mid H\rangle \end{cases}$$

$$\texttt{SRaise} \quad \langle(\sigma \vdash_S \texttt{raise}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle\texttt{raise(RuntimeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle$$

$$\texttt{SDef} \quad \langle(\sigma \vdash_S \texttt{def } x_0\ (x_1, \ldots, x_n)\ B) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle\texttt{write[}\sigma(x_0)\texttt{]} :: \kappa \mid\mid a :: s \mid\mid H \mid\mid M[a \mapsto v]\rangle$$
$$\text{where } a \notin \text{Domain}(M) \text{ and } v = \begin{cases} \langle\lambda*(x_1, \ldots, x_n).B, \sigma\rangle & \text{if } \texttt{hasYield}(B) \\ \langle\lambda(x_1, \ldots, x_n).B, \sigma\rangle & \text{otherwise} \end{cases}$$

$$\texttt{SReturn} \quad \langle(\sigma \vdash_S \texttt{return } e) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle(\sigma \vdash_e e) :: \texttt{return} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$

$$\texttt{SYield} \quad \langle(\sigma \vdash_S \texttt{yield } e) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$
$$\rightarrow \quad \langle(\sigma \vdash_e e) :: \texttt{yield} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle$$

4.2 Reduction Relations for `IBlock`

$$\begin{array}{ll} \texttt{IBlock} & \langle(\sigma \vdash_B \{S_1; \ldots; S_n\}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle(\sigma \vdash_S S_1) :: \ldots :: (\sigma \vdash_S S_n) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \end{array}$$

4.3 Reduction Relations for `IExpr`

$$\begin{array}{ll} \texttt{ENone} & \langle(\sigma \vdash_e \mathsf{None}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle\kappa \mid\mid \mathsf{None} :: s \mid\mid H \mid\mid M\rangle \end{array}$$

$$\begin{array}{ll} \texttt{ENum} & \langle(\sigma \vdash_e n) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle\kappa \mid\mid n :: s \mid\mid H \mid\mid M\rangle \end{array}$$

$$\begin{array}{ll} \texttt{EBool} & \langle(\sigma \vdash_e b) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle\kappa \mid\mid b :: s \mid\mid H \mid\mid M\rangle \end{array}$$

$$\begin{array}{ll} \texttt{EId} & \langle(\sigma \vdash_e x) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \begin{cases} \langle\kappa \mid\mid M(\sigma(x)) :: s \mid\mid H \mid\mid M\rangle & \text{if } x \in \text{Domain}(\sigma) \\ \langle\texttt{raise}(\mathsf{NameError}) :: \Box \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases} \end{array}$$

$$\begin{array}{ll} \texttt{EBOp} & \langle(\sigma \vdash_e e_0 \oplus e_1) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle(\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: (\oplus) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \end{array}$$

$$\begin{array}{ll} \texttt{EList} & \langle(\sigma \vdash_e \texttt{[}e_1, \ldots, e_n\texttt{]}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle(\sigma \vdash_e e_1) :: \ldots :: (\sigma \vdash_e e_n) :: \texttt{list[}n\texttt{]} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \end{array}$$

$$\begin{array}{ll} \texttt{EAppend} & \langle(\sigma \vdash_e e_0.\mathsf{append}(e_1)) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle(\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \mathsf{append} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \end{array}$$

$$\begin{array}{ll} \texttt{EGetItem} & \langle(\sigma \vdash_e e_0\texttt{[}e_1\texttt{]}) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle(\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \texttt{get-item} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \end{array}$$

$$\begin{array}{ll} \texttt{ELambda} & \langle(\sigma \vdash_e \lambda(x_1, \ldots, x_n).e) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle\kappa \mid\mid a :: s \mid\mid H \mid\mid M[a \mapsto \langle\lambda(x_1, \ldots, x_n).\mathsf{return}\ e, \sigma\rangle]\rangle \\ & \text{where } a \notin \text{Domain}(M) \end{array}$$

$$\begin{array}{ll} \texttt{EApp} & \langle(\sigma \vdash_e e_0(e_1, \ldots, e_n)) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle(\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \ldots :: (\sigma \vdash_e e_n) :: \texttt{call[}n\texttt{]} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \end{array}$$

$$\begin{array}{ll} \texttt{ECond} & \langle(\sigma \vdash_e e_0\ \texttt{if}\ e_1\ \texttt{else}\ e_2) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle(\sigma \vdash_e e_1) :: \texttt{jmp-if[}\psi\texttt{]} :: (\sigma \vdash_e e_2) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ & \text{where } \psi = \langle(\sigma \vdash_e e_0) :: \kappa \mid\mid s \mid\mid H\rangle \end{array}$$

$$\begin{array}{ll} \texttt{EIter} & \langle(\sigma \vdash_e \texttt{iter}(e)) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle(\sigma \vdash_e e) :: \texttt{iter} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \end{array}$$

$$\begin{array}{ll} \texttt{ENext} & \langle(\sigma \vdash_e \mathsf{next}(e)) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\ \rightarrow & \langle(\sigma \vdash_e e) :: \mathsf{next} :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \end{array}$$

4.4 Reduction Relations for IBop

$$
\begin{array}{llcl}
\texttt{Add} & \langle (+) :: \kappa \mid\mid n_2 :: n_1 :: s \mid\mid H \mid\mid M\rangle & \longrightarrow & \langle \kappa \mid\mid (n_1 + n_2) :: s \mid\mid H \mid\mid M\rangle \\
\texttt{Mul} & \langle (*) :: \kappa \mid\mid n_2 :: n_1 :: s \mid\mid H \mid\mid M\rangle & \longrightarrow & \langle \kappa \mid\mid (n_1 \times n_2) :: s \mid\mid H \mid\mid M\rangle \\
\texttt{Div}_0 & \langle (//) :: \kappa \mid\mid 0 :: n_1 :: s \mid\mid H \mid\mid M\rangle & \longrightarrow & \langle \texttt{raise(ZeroDivisionError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle \\
\texttt{Div} & \langle (//) :: \kappa \mid\mid n_2 :: n_1 :: s \mid\mid H \mid\mid M\rangle & \longrightarrow & \langle \kappa \mid\mid (n_1 \div n_2) :: s \mid\mid H \mid\mid M\rangle \\
\texttt{Mod}_0 & \langle (\%) :: \kappa \mid\mid 0 :: n_1 :: s \mid\mid H \mid\mid M\rangle & \longrightarrow & \langle \texttt{raise(ZeroDivisionError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle \\
\texttt{Mod} & \langle (\%) :: \kappa \mid\mid n_2 :: n_1 :: s \mid\mid H \mid\mid M\rangle & \longrightarrow & \langle \kappa \mid\mid (n_1 \,\%\, n_2) :: s \mid\mid H \mid\mid M\rangle \\
\texttt{Eq} & \langle (==) :: \kappa \mid\mid v_2 :: v_1 :: s \mid\mid H \mid\mid M\rangle & \longrightarrow & \langle \kappa \mid\mid \texttt{equal}(v_1, v_2, M) :: s \mid\mid H \mid\mid M\rangle \\
\texttt{Is} & \langle (\texttt{is}) :: \kappa \mid\mid v_2 :: v_1 :: s \mid\mid H \mid\mid M\rangle & \longrightarrow & \langle \kappa \mid\mid \texttt{is}(v_1, v_2) :: s \mid\mid H \mid\mid M\rangle
\end{array}
$$

$$
\begin{array}{ll}
\texttt{Lt} & \langle (<) :: \kappa \mid\mid v_2 :: v_1 :: s \mid\mid H \mid\mid M\rangle \\
\longrightarrow & \begin{cases} \langle \kappa \mid\mid b :: s \mid\mid H \mid\mid M\rangle & \text{if } \texttt{lessThan}(v_1, v_2, M) = b \\ \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{ll}
\texttt{Lte} & \langle (<=) :: \kappa \mid\mid v_2 :: v_1 :: s \mid\mid H \mid\mid M\rangle \\
\longrightarrow & \begin{cases} \langle \kappa \mid\mid (b \vee \texttt{equal}(v_1, v_2, M)) :: s \mid\mid H \mid\mid M\rangle & \text{if } \texttt{lessThan}(v_1, v_2, M) = b \\ \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases}
\end{array}
$$

4.5 Reduction Relations for Other Instructions

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

$$
\begin{array}{ll}
\texttt{IGetItem} & \langle \texttt{get-item} :: \kappa \mid\mid n :: a :: s \mid\mid H \mid\mid M\rangle \\
\longrightarrow & \begin{cases} \langle \kappa \mid\mid v_{m+n} :: s \mid\mid H \mid\mid M\rangle & \text{if } M(a) = [v_1, \ldots, v_m] \wedge -m \le n < 0 \\ \langle \kappa \mid\mid v_n :: s \mid\mid H \mid\mid M\rangle & \text{if } M(a) = [v_1, \ldots, v_m] \wedge 0 \le n < m \\ \langle \texttt{raise(IndexError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{if } M(a) = [v_1, \ldots, v_m] \wedge (n < -m \vee m \le n) \\ \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{ll}
\texttt{ISetItem} & \langle \texttt{set-item} :: \kappa \mid\mid n :: a :: v :: s \mid\mid H \mid\mid M\rangle \\
\longrightarrow & \begin{cases} \langle \kappa \mid\mid s \mid\mid H \mid\mid M + (a \mapsto v'_{m+n})\rangle & \text{if } M(a) = [v_1, \ldots, v_m] \wedge -m \le n < 0 \\ \langle \kappa \mid\mid s \mid\mid H \mid\mid M + (a \mapsto v'_n)\rangle & \text{if } M(a) = [v_1, \ldots, v_m] \wedge 0 \le n < m \\ \langle \texttt{raise(IndexError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{if } M(a) = [v_1, \ldots, v_m] \wedge (n < -m \vee m \le n) \\ \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases} \\
 & \text{where } v'_k = [v_1, \ldots, v_{k-1}, v, v_{k+1}, \ldots, v_m]
\end{array}
$$

$$\begin{array}{ll}
\texttt{IList} & \langle \texttt{list}[n] :: \kappa \mid\mid v_n :: \ldots :: v_1 :: s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle \kappa \mid\mid a :: s \mid\mid H \mid\mid M + (a \mapsto [v_1, \ldots, v_n])\rangle \\
& \text{where } a \notin \mathrm{Domain}(M)
\end{array}$$

$$\begin{array}{ll}
\texttt{IAppend} & \langle \mathsf{append} :: \kappa \mid\mid v :: a :: s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \begin{cases} \langle \kappa \mid\mid a :: s \mid\mid H \mid\mid M + (a \mapsto [v_1, \ldots, v_m, v])\rangle & \text{if } M(a) = [v_1, \ldots, v_m] \\ \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases}
\end{array}$$

$$\begin{array}{ll}
\texttt{IJmpIf} & \langle \texttt{jmp-if}[\langle \kappa' \mid\mid s' \mid\mid H'\rangle] :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \begin{cases} \langle \kappa' \mid\mid s' \mid\mid H' \mid\mid M\rangle & \text{if } \texttt{isTruthy}(v, M) = \mathsf{True} \\ \langle \kappa \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases}
\end{array}$$

$$\begin{array}{ll}
\texttt{IJmp} & \langle \texttt{jmp}[c] :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle \kappa' \mid\mid s' \mid\mid H' \mid\mid M\rangle \\
& \text{where } \langle \kappa' \mid\mid s' \mid\mid H'\rangle = H(c)
\end{array}$$

$$\begin{array}{ll}
\texttt{IRaise} & \langle \texttt{raise}(\omega) :: \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \begin{cases} \langle \texttt{jmp[raise]} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{if raise} \in \mathrm{Domain}(H) \\ \text{(runtime error with message with } \omega\text{'s name)} & \text{otherwise} \end{cases}
\end{array}$$

$$\begin{array}{ll}
\texttt{ICall} & \langle \texttt{call}[n] :: \kappa \mid\mid v_n :: \ldots :: v_1 :: a :: s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \begin{cases} \langle \kappa' \mid\mid \mathsf{None} :: \blacksquare \mid\mid H_{\text{body}} \mid\mid M_2\rangle & \text{if } M(a) = \langle \lambda(x_1, \ldots, x_n).B, \sigma'\rangle \\ \langle \kappa \mid\mid a_{\text{iter}} :: s \mid\mid H \mid\mid M_4\rangle & \text{if } M(a) = \langle \lambda{*}(x_1, \ldots, x_n).e, \sigma'\rangle \\ \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases} \\
& \text{where } \begin{cases} a_1, \ldots, a_n & \notin \mathrm{Domain}(M) \\ \{x'_1, \ldots, x'_m\} & = \texttt{locals}(B) \setminus \{x_1, \ldots, x_n\} \\ M_1 & = M[a_1 \mapsto v_1, \ldots, a_n \mapsto v_n] \\ a'_1, \ldots, a'_m & \notin \mathrm{Domain}(M_1) \\ M_2 & = M_1[a'_1 \mapsto \mathsf{None}, ldots, a'_m \mapsto \mathsf{None}] \\ \sigma_{\text{body}} & = \sigma'[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n, x'_1 \mapsto a'_1, \ldots, x'_m \mapsto a'_m] \\ H_{\text{body}} & = H[\texttt{return} \mapsto \psi_{\text{return}}] \setminus \{\mathsf{break}, \mathsf{continue}, \texttt{yield}\} \\ \psi_{\text{return}} & = \langle \kappa \mid\mid s \mid\mid H\rangle \\ \kappa' & = (\sigma_{\text{body}} \vdash_B B) :: \texttt{return} :: \square \\ \psi_{\text{next}} & = \langle \kappa' \mid\mid \mathsf{None} :: \blacksquare \mid\mid H_{\text{body}}\rangle \\ a_{\text{cont}} & \notin \mathrm{Domain}(M_2) \\ M_3 & = M_2[a_{\text{cont}} \mapsto \psi_{\text{next}}] \\ a_{\text{iter}} & \notin \mathrm{Domain}(M_3) \\ M_4 & = M_3[a_{\text{iter}} \mapsto \texttt{iter}[a_{\text{cont}}, 0]] \end{cases}
\end{array}$$

$$\begin{array}{ll}
\texttt{IReturn} & \langle \texttt{return} :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle \kappa' \mid\mid v :: s' \mid\mid H' \mid\mid M\rangle \\
& \text{where } \langle \kappa' \mid\mid s' \mid\mid H'\rangle = H(\texttt{return})
\end{array}$$

$$\begin{array}{ll}
\texttt{IYield} & \langle \texttt{yield} :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \langle \kappa' \mid\mid \langle \kappa \mid\mid s \mid\mid H\rangle :: v :: s' \mid\mid H' \mid\mid M\rangle \\
& \text{where } \langle \kappa' \mid\mid s' \mid\mid H'\rangle = H(\texttt{yield})
\end{array}$$

$$\begin{array}{ll}
\texttt{IIter} & \langle \texttt{iter} :: \kappa \mid\mid a :: s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \begin{cases} \langle \kappa \mid\mid a :: s \mid\mid H \mid\mid M\rangle & \text{if } M(a) = \texttt{iter[}_, i\texttt{]} \\ \langle \kappa \mid\mid a_{\text{iter}} :: s \mid\mid H \mid\mid M[a_{\text{iter}} \mapsto \texttt{iter[}a, 0\texttt{]}]\rangle & \text{if } M(a) = \texttt{[}_\texttt{]} \\ \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases} \\
& \text{where } a_{\text{iter}} \notin \text{Domain}(M)
\end{array}$$

$$\begin{array}{ll}
\texttt{INext} & \langle \texttt{next} :: \kappa \mid\mid a :: s \mid\mid H \mid\mid M\rangle \\
\rightarrow & \begin{cases} \langle \kappa' \mid\mid s' \mid\mid H_{\text{next}} \mid\mid M\rangle & \text{if } v = \texttt{iter[}a', m\texttt{]} \wedge v' = \langle \kappa' \mid\mid s' \mid\mid H'\rangle \\ \langle \kappa \mid\mid v_m :: s \mid\mid H \mid\mid M_{\text{next}}\rangle & \text{if } v = \texttt{iter[}a', m\texttt{]} \wedge v' = \texttt{[}v_1, \ldots, v_n\texttt{]} \wedge m < n \\ \langle \texttt{raise(StopIteration)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{if } v = \texttt{iter[}a', m\texttt{]} \wedge v' = \texttt{[}v_1, \ldots, v_n\texttt{]} \wedge n \le m \\ \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle & \text{otherwise} \end{cases} \\
& \text{where } \begin{cases} v & = M(a) \\ v' & = M(a') \\ \psi_{\text{yield}} & = \langle \texttt{write[}a'\texttt{]} :: \kappa \mid\mid s \mid\mid H\rangle \\ \psi_{\text{return}} & = \langle \texttt{drop} :: \texttt{raise(StopIteration)} :: \square \mid\mid s \mid\mid H\rangle \\ H_{\text{next}} & = H'[\texttt{yield} \mapsto \psi_{\text{yield}}, \texttt{return} \mapsto \psi_{\text{return}}] \\ M_{\text{next}} & = M[a \mapsto \texttt{iter[}a', m+1\texttt{]}] \end{cases}
\end{array}$$

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

Other remaining cases raise a TypeError as follow:

$$\langle \kappa \mid\mid s \mid\mid H \mid\mid M\rangle \rightarrow \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M\rangle$$

4.6 Auxiliary Definitions

The following auxiliary functions are used in the reduction rules. Note that $A \rightharpoonup B$ denotes a partial function from set A to set B.

$$\boxed{\textsf{equal} : (\mathbb{V} \times \mathbb{V} \times \mathbb{M}) \rightarrow \mathbb{B}}$$

$$\textsf{equal}(v, v', M) = \begin{cases} \textsf{True} & \texttt{is}(v, v', M) \\ \textsf{equal}(M(a), M(a'), M) & v = a \wedge v' = a' \\ \textsf{equal}(v_1, v'_1, M) \wedge \ldots \wedge \textsf{equal}(v_n, v'_n, M) & v = \texttt{[}v_1, \ldots, v_n\texttt{]} \wedge v' = \texttt{[}v'_1, \ldots, v'_n\texttt{]} \\ \textsf{False} & \text{otherwise} \end{cases}$$

$$\boxed{\texttt{lessThan} : (\mathbb{V} \times \mathbb{V} \times \mathbb{M}) \rightharpoonup \mathbb{B}}$$

$$\begin{array}{ll}
\texttt{lessThan}(n_0, n_1, M) & = n_0 < n_1 \\
\texttt{lessThan}(a_0, a_1, M) & = \texttt{lessThan}(M(a_0), M(a_1), M) \\
\texttt{lessThan}(\texttt{[}v_1, \ldots, v_n\texttt{]}, \texttt{[}v'_1, \ldots, v'_m\texttt{]}, M) & = \begin{cases} \textsf{False} & \text{if } n = m = 0 \\ \textsf{True} & \text{if } n = 0 < m \\ \textsf{False} & \text{if } n > 0 = m \\ \textsf{True} & \text{if } b \\ \textsf{False} & \text{if } \neg b \wedge \neg b' \\ \texttt{lessThan}(\texttt{[}v_2, \ldots, v_n\texttt{]}, \texttt{[}v'_2, \ldots, v'_m\texttt{]}, M) & \text{if } \neg b \wedge b' \end{cases} \\
& \text{where } b = \texttt{lessThan}(v_1, v'_1, M) \text{ and } b' = \textsf{equal}(v_1, v'_1, M)
\end{array}$$

$$\boxed{\texttt{is} : (\mathbb{V} \times \mathbb{V}) \rightarrow \mathbb{B}}$$

$$\texttt{is}(v, v') = \begin{cases} \mathsf{True} & \text{if } v = v' = \mathsf{None} \\ \mathsf{True} & \text{if } v = v' = n \\ \mathsf{True} & \text{if } v = v' = b \\ \mathsf{True} & \text{if } v = v' = a \\ \mathsf{False} & \text{otherwise} \end{cases}$$

$$\boxed{\texttt{isTruthy} : (\mathbb{V} \times \mathbb{M}) \rightarrow \mathbb{B}}$$

$$\texttt{isTruthy}(v, M) = \begin{cases} \mathsf{False} & \text{if } v = \mathsf{None} \\ n \neq 0 & \text{if } v = n \\ b & \text{if } v = b \\ \texttt{isTruthy}(M(a), M) & \text{if } v = a \\ 0 < n & \text{if } v = [v_1, \ldots, v_n] \\ \mathsf{True} & \text{otherwise} \end{cases}$$

$$\boxed{\texttt{locals}(B) : \mathcal{P}(\mathbb{X})}$$

$$\texttt{locals}(\{S_1; \ldots; S_n\}) = \texttt{locals}(S_1) \cup \ldots \cup \texttt{locals}(S_n)$$

$$\texttt{locals}(S) = \begin{cases} \{x\} & \text{if } S = x = e \\ \{x_0\} & \text{if } S = \texttt{def}\ x_0\ (x_1, \ldots, x_n)\ B \\ \texttt{locals}(B_0) \cup \texttt{locals}(B_1) & \text{if } S = \texttt{if}\ e\ B_0\ \texttt{else}\ B_1 \\ \texttt{locals}(B_0) \cup \texttt{locals}(B_1) & \text{if } S = \texttt{try}\ B_0\ \texttt{except}\ B_1 \\ \texttt{locals}(B) & \text{if } S = \texttt{while}\ e\ B \\ \varnothing & \text{otherwise} \end{cases}$$

$$\boxed{\texttt{hasYield}(B) : \mathbb{B}}$$

$$\texttt{hasYield}(\{S_1; \ldots; S_n\}) = \texttt{hasYield}(S_1) \vee \ldots \vee \texttt{hasYield}(S_n)$$

$$\texttt{hasYield}(S) = \begin{cases} \mathsf{True} & \text{if } S = \texttt{yield}\ e \\ \texttt{hasYield}(B) & \text{if } S = \texttt{def}\ x_0\ (x_1, \ldots, x_n)\ B \\ \texttt{hasYield}(B_0) \vee \texttt{hasYield}(B_1) & \text{if } S = \texttt{if}\ e\ B_0\ \texttt{else}\ B_1 \\ \texttt{hasYield}(B_0) \vee \texttt{hasYield}(B_1) & \text{if } S = \texttt{try}\ B_0\ \texttt{except}\ B_1 \\ \texttt{hasYield}(B) & \text{if } S = \texttt{while}\ e\ B \\ \mathsf{False} & \text{otherwise} \end{cases}$$