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	
Additive	`+, -`	7	
Comparison	`==, !=, is, is not, <, <=, >, >=`	6	left
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:

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

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

Expressions $e ::=$ `None`   (ENone) $\mid e[e]$ (EGetItem)
$\mid n$ (ENum) $\mid \lambda(x, \ldots, x).e$ (ELambda)
$\mid b$ (EBool) $\mid e(e, \ldots, e)$ (EApp)
$\mid x$ (EId) $\mid e$ `if` e `else` e (ECond)
$\mid e \oplus e$ (EBOp) $\mid$ `iter`(e) (EIter)
$\mid [e, \ldots, e]$ (EList) $\mid$ `next`(e) (ENext)
$\mid e$.`append`(e) (EAppend)

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

where $\Big\{$ Booleans $\mathbb{B} \ni b ::=$ `True` $\mid$ `False`   (Boolean)   Identifiers $x \in \mathbb{X}$ (String)
Numbers $n, m, k \in \mathbb{Z}$ (BigInt)

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

$$\mathcal{D}[\![- e]\!] \quad = \mathcal{D}[\![e]\!] * (-1) \qquad \mathcal{D}[\![e_1 \text{ is not } e_2]\!] = \mathcal{D}[\![\text{not } (e_1 \text{ is } e_2)]\!]$$
$$\mathcal{D}[\![e_1 - e_2]\!] = \mathcal{D}[\![e_1]\!] + \mathcal{D}[\![- e_2]\!] \qquad \mathcal{D}[\![e_1 \text{ and } e_2]\!] \quad = \text{if } (\mathcal{D}[\![e_1]\!])\ \mathcal{D}[\![e_2]\!] \text{ else False}$$
$$\mathcal{D}[\![e_1 > e_2]\!] = \mathcal{D}[\![\text{not } (e_1 <= e_2)]\!] \qquad \mathcal{D}[\![e_1 \text{ or } e_2]\!] \quad\; = \text{if } (\mathcal{D}[\![e_1]\!]) \text{ True else } \mathcal{D}[\![e_2]\!]$$
$$\mathcal{D}[\![e_1 >= e_2]\!] = \mathcal{D}[\![\text{not } (e_1 < e_2)]\!] \qquad \mathcal{D}[\![\text{not } e]\!] \qquad\;\; = \text{if } (\mathcal{D}[\![e]\!]) \text{ False else True}$$
$$\mathcal{D}[\![e_1 \;!= e_2]\!] = \mathcal{D}[\![\text{not } (e_1 == e_2)]\!] \qquad \mathcal{D}[\![\text{yield from } e]\!] = \mathcal{D}[\![\text{for } x \text{ in } e \text{ yield } x]\!]$$

$$\mathcal{D}[\![\text{if } e\ B \text{ elif } e_1\ B_1 \ldots \text{elif } e_n\ B_n \text{ else } B']\!] = \mathcal{D}[\![\text{if } e\ B \text{ else if } e_1\ B_1 \ldots \text{else if } e_n\ B_n \text{ else } B']\!]$$
$$\mathcal{D}[\![\text{if } e\ B \text{ elif } e_1\ B_1 \ldots \text{elif } e_n\ B_n]\!] \qquad\quad = \mathcal{D}[\![\text{if } e\ B \text{ else if } e_1\ B_1 \ldots \text{else if } e_n\ B_n \text{ else pass}]\!]$$

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

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:

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

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

Value Stacks $\quad s ::= \blacksquare \mid v :: s \quad$ (Stack)

Control Handlers $\quad H \in \mathbb{H} = \mathbb{C} \xrightarrow{\text{fin}} \Psi$ (Handler) $\qquad$ Environments $\quad \sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{A}$ (Env)

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

Errors $\quad \omega ::= \text{RuntimeError} \mid \text{ZeroDivisionError} \mid \text{TypeError} \mid \text{IndexError}$
$\qquad\qquad \mid \text{StopIteration} \mid \text{NameError}[x]$

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

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

Controls $\quad c ::= \text{return} \qquad$ (Return) $\qquad \mid \text{throw} \qquad$ (Throw)
$\qquad\qquad\ \mid \text{break} \qquad$ (Break) $\qquad\ \mid \text{finally} \quad$ (Finally)
$\qquad\qquad\ \mid \text{continue} \quad$ (Continue) $\quad \mid \text{yield} \qquad$ (Yield)

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

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

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

4.1 Reduction Relations for IStmt

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

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

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

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

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

SWhile
$\langle (\sigma \vdash_S \text{while } e\ B) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle$
$\rightarrow$
$\langle (\sigma \vdash_e e) :: \text{jmp-if}[\psi_{\text{body}}] :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle$

where $\begin{cases} \psi_{\text{body}} &= \langle (\sigma \vdash_B B) :: (\sigma \vdash_S \text{while } e\ B) :: \kappa \mid\mid s \mid\mid H_{\text{body}} \rangle \\ H_{\text{body}} &= H[\text{continue} \mapsto \psi_{\text{continue}}, \text{break} \mapsto \psi_{\text{break}}] \\ \psi_{\text{continue}} &= \langle (\sigma \vdash_S \text{while } e\ B) :: \kappa \mid\mid s \mid\mid H \rangle \\ \psi_{\text{break}} &= \langle \kappa \mid\mid s \mid\mid H \rangle \end{cases}$

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

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

STry
$\langle (\sigma \vdash_S \text{try } B_0 \text{ except } B_1) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle$
$\rightarrow$
$\langle (\sigma \vdash_B B_0) :: \text{jmp}[\text{finally}] :: \square \mid\mid s \mid\mid H_{\text{body}} \mid\mid M \rangle$

where $\begin{cases} H_{\text{body}} &= H[\text{raise} \mapsto \psi_{\text{raise}}, \text{finally} \mapsto \psi_{\text{finally}}] \\ \psi_{\text{raise}} &= \langle (\sigma \vdash_B B_1) :: \kappa \mid\mid s \mid\mid H \rangle \\ \psi_{\text{finally}} &= \langle \kappa \mid\mid s \mid\mid H \rangle \end{cases}$

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

SDef
$\langle (\sigma \vdash_S \text{def } x_0\ (x_1, \ldots, x_n)\ B) :: \kappa \mid\mid s \mid\mid H \mid\mid M \rangle$
$\rightarrow$
$\langle \text{write}[\sigma(x_0)] :: \kappa \mid\mid a :: s \mid\mid H \mid\mid M[a \mapsto v] \rangle$

where $a \notin \text{Domain}(M)$ and $v = \begin{cases} \langle \lambda *(x_1, \ldots, x_n).B, \sigma \rangle & \text{if hasYield}(B) \\ \langle \lambda(x_1, \ldots, x_n).B, \sigma \rangle & \text{otherwise} \end{cases}$

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

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

4.2 Reduction Relations for IBlock

$$\text{IBlock} \quad \langle (\sigma \vdash_B \{S_1; \ldots; S_n\}) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$$
$$\rightarrow \quad \langle (\sigma \vdash_S S_1) :: \ldots :: (\sigma \vdash_S S_n) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$$

4.3 Reduction Relations for IExpr

ENone $\quad \langle (\sigma \vdash_e \text{None}) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle \kappa \mathbin{||} \text{None} :: s \mathbin{||} H \mathbin{||} M \rangle$

ENum $\quad \langle (\sigma \vdash_e n) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle \kappa \mathbin{||} n :: s \mathbin{||} H \mathbin{||} M \rangle$

EBool $\quad \langle (\sigma \vdash_e b) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle \kappa \mathbin{||} b :: s \mathbin{||} H \mathbin{||} M \rangle$

EId $\quad \langle (\sigma \vdash_e x) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \begin{cases} \langle \kappa \mathbin{||} M(\sigma(x)) :: s \mathbin{||} H \mathbin{||} M \rangle & \text{if } x \in \text{Domain}(\sigma) \\ \langle \text{raise}(\text{NameError}[x]) :: \square \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle & \text{otherwise} \end{cases}$

EBOp $\quad \langle (\sigma \vdash_e e_0 \oplus e_1) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: (\oplus) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$

EList $\quad \langle (\sigma \vdash_e [e_1, \ldots, e_n]) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle (\sigma \vdash_e e_1) :: \ldots :: (\sigma \vdash_e e_n) :: \text{list}[n] :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$

EAppend $\quad \langle (\sigma \vdash_e e_0.\text{append}(e_1)) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \text{append} :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$

EGetItem $\quad \langle (\sigma \vdash_e e_0[e_1]) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \text{get-item} :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$

ELambda $\quad \langle (\sigma \vdash_e \lambda(x_1, \ldots, x_n).e) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle \kappa \mathbin{||} a :: s \mathbin{||} H \mathbin{||} M[a \mapsto \langle \lambda(x_1, \ldots, x_n).\text{return } e, \sigma \rangle] \rangle$
$\quad \text{where } a \notin \text{Domain}(M)$

EApp $\quad \langle (\sigma \vdash_e e_0(e_1, \ldots, e_n)) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \ldots :: (\sigma \vdash_e e_n) :: \text{call}[n] :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$

ECond $\quad \langle (\sigma \vdash_e e_0 \text{ if } e_1 \text{ else } e_2) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle (\sigma \vdash_e e_1) :: \text{jmp-if}[\psi] :: (\sigma \vdash_e e_2) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\quad \text{where } \psi = \langle (\sigma \vdash_e e_0) :: \kappa \mathbin{||} s \mathbin{||} H \rangle$

EIter $\quad \langle (\sigma \vdash_e \text{iter}(e)) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle (\sigma \vdash_e e) :: \text{iter} :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$

ENext $\quad \langle (\sigma \vdash_e \text{next}(e)) :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$
$\rightarrow \quad \langle (\sigma \vdash_e e) :: \text{next} :: \kappa \mathbin{||} s \mathbin{||} H \mathbin{||} M \rangle$

4.4 Reduction Relations for IBop

Add $\langle (+) :: \kappa \,||\, n_2 :: n_1 :: s \,||\, H \,||\, M \rangle \quad \rightarrow \quad \langle \kappa \,||\, (n_1 + n_2) :: s \,||\, H \,||\, M \rangle$

Mul $\langle (*) :: \kappa \,||\, n_2 :: n_1 :: s \,||\, H \,||\, M \rangle \quad \rightarrow \quad \langle \kappa \,||\, (n_1 \times n_2) :: s \,||\, H \,||\, M \rangle$

Div_0 $\langle (//) :: \kappa \,||\, 0 :: n_1 :: s \,||\, H \,||\, M \rangle \quad \rightarrow \quad \langle \text{raise}(\text{ZeroDivisionError}) :: \square \,||\, s \,||\, H \,||\, M \rangle$

Div $\langle (//) :: \kappa \,||\, n_2 :: n_1 :: s \,||\, H \,||\, M \rangle \quad \rightarrow \quad \langle \kappa \,||\, (n_1 \div n_2) :: s \,||\, H \,||\, M \rangle$

Mod_0 $\langle (\%) :: \kappa \,||\, 0 :: n_1 :: s \,||\, H \,||\, M \rangle \quad \rightarrow \quad \langle \text{raise}(\text{ZeroDivisionError}) :: \square \,||\, s \,||\, H \,||\, M \rangle$

Mod $\langle (\%) :: \kappa \,||\, n_2 :: n_1 :: s \,||\, H \,||\, M \rangle \quad \rightarrow \quad \langle \kappa \,||\, (n_1 \% n_2) :: s \,||\, H \,||\, M \rangle$

Eq $\langle (==) :: \kappa \,||\, v_2 :: v_1 :: s \,||\, H \,||\, M \rangle \quad \rightarrow \quad \langle \kappa \,||\, \text{equal}(v_1, v_2, M) :: s \,||\, H \,||\, M \rangle$

Is $\langle (\text{is}) :: \kappa \,||\, v_2 :: v_1 :: s \,||\, H \,||\, M \rangle \quad \rightarrow \quad \langle \kappa \,||\, \text{is}(v_1, v_2) :: s \,||\, H \,||\, M \rangle$

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

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

4.5 Reduction Relations for Other Instructions

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

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

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

ILIST $\langle \texttt{list}[n] :: \kappa \mid\mid v_n :: \ldots :: v_1 :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$ $\langle \kappa \mid\mid a :: s \mid\mid H \mid\mid M + (a \mapsto [v_1, \ldots, v_n]) \rangle$
where $a \notin \text{Domain}(M)$

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

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

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

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

ICall $\langle \texttt{call}[n] :: \kappa \mid\mid v_n :: \ldots :: v_1 :: a :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$
$\begin{cases} \langle \kappa' \mid\mid \texttt{None} :: \blacksquare \mid\mid H_{\text{body}} \mid\mid M_2 \rangle & \text{if } M(a) = \langle \lambda(x_1, \ldots, x_n).B, \sigma' \rangle \\ \langle \kappa \mid\mid a_{\text{iter}} :: s \mid\mid H \mid\mid M_4 \rangle & \text{if } M(a) = \langle \lambda*(x_1, \ldots, x_n).e, \sigma' \rangle \\ \langle \texttt{raise(TypeError)} :: \square \mid\mid s \mid\mid H \mid\mid M \rangle & \text{otherwise} \end{cases}$

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

IReturn $\langle \texttt{return} :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$ $\langle \kappa' \mid\mid v :: s' \mid\mid H' \mid\mid M \rangle$
where $\langle \kappa' \mid\mid s' \mid\mid H' \rangle = H(\texttt{return})$

IYield $\langle \texttt{yield} :: \kappa \mid\mid v :: s \mid\mid H \mid\mid M \rangle$
$\rightarrow$ $\langle \kappa' \mid\mid \langle \kappa \mid\mid s \mid\mid H \rangle :: v :: s' \mid\mid H' \mid\mid M \rangle$
where $\langle \kappa' \mid\mid s' \mid\mid H' \rangle = H(\texttt{yield})$

IIter $\langle \texttt{iter} :: \kappa \,||\, a :: s \,||\, H \,||\, M \rangle$

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

INext $\langle \texttt{next} :: \kappa \,||\, a :: s \,||\, H \,||\, M \rangle$

$\rightarrow$
$$\begin{cases} \langle \kappa' \,||\, s' \,||\, H_{\text{next}} \,||\, M \rangle & \text{if } v = \texttt{iter}[a', m] \wedge v' = \langle \kappa' \,||\, s' \,||\, H' \rangle \\ \langle \kappa \,||\, v_m :: s \,||\, H \,||\, M_{\text{next}} \rangle & \text{if } v = \texttt{iter}[a', m] \wedge v' = [v_1, \ldots, v_n] \wedge m < n \\ \langle \texttt{raise(StopIteration)} :: \square \,||\, s \,||\, H \,||\, M \rangle & \text{if } v = \texttt{iter}[a', m] \wedge v' = [v_1, \ldots, v_n] \wedge n \le m \\ \langle \texttt{raise(TypeError)} :: \square \,||\, s \,||\, H \,||\, M \rangle & \text{otherwise} \end{cases}$$

where
$$\begin{cases} v & = M(a) \\ v' & = M(a') \\ \psi_{\text{yield}} & = \langle \texttt{write}[a'] :: \kappa \,||\, s \,||\, H \rangle \\ \psi_{\text{return}} & = \langle \texttt{drop} :: \texttt{raise(StopIteration)} :: \square \,||\, s \,||\, H \rangle \\ H_{\text{next}} & = H'[\texttt{yield} \mapsto \psi_{\text{yield}}, \texttt{return} \mapsto \psi_{\text{return}}] \\ M_{\text{next}} & = M[a \mapsto \texttt{iter}[a', m + 1]] \end{cases}$$

IDrop $\langle \texttt{drop} :: \kappa \,||\, v :: s \,||\, H \,||\, M \rangle$
$\rightarrow$ $\langle \kappa \,||\, s \,||\, H \,||\, M \rangle$

Other remaining cases raise a TypeError as follow:

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

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.

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

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

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

$\texttt{lessThan}(n_0, n_1, M) \qquad\qquad\qquad = n_0 < n_1$

$\texttt{lessThan}(a_0, a_1, M) \qquad\qquad\qquad = \texttt{lessThan}(M(a_0), M(a_1), M)$

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

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

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

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

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

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

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

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

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

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

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