MiniPy – The Target Language of SWS128

MiniPy is the language analyzed in SWS128: Software Analysis at Korea University. It is a subset of Python: every MiniPy program is a valid Python program. The concrete syntax is Python's, published at https://docs.python.org/3/reference/grammar.html; this document defines the abstract syntax and the semantics, and the reference implementation is `syntax.py` and `interp.py`. The language is deliberately small: one function, no calls, and lists as the only heap objects.

1 ABSTRACT SYNTAX

Let $n \in \mathbb{Z}$, $s \in \mathbb{S}$, and $x, f \in \mathbb{X}$. The name in parentheses is the class in `syntax.py`.

Binary operators	$\oplus ::= \texttt{+} \mid \texttt{*} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{==} \mid \texttt{!=} \mid \texttt{and} \mid \texttt{or}$	
Unary operators	$\ominus ::= \texttt{-} \mid \texttt{not}$	
Expressions	$E ::= n$	(`Num`)
	$\mid s$	(`Str`)
	$\mid \texttt{true} \mid \texttt{false}$	(`Bool`)
	$\mid \texttt{None}$	(`NoneLit`)
	$\mid x$	(`Var`)
	$\mid E \oplus E$	(`BinOp`)
	$\mid \ominus E$	(`UnOp`)
	$\mid [E, \ldots, E]$	(`ListLit`)
	$\mid E[E]$	(`Index`)
Statements	$S ::= \texttt{pass}$	(`Pass`)
	$\mid x := E$	(`Assign`)
	$\mid x[E] := E$	(`IndexAssign`)
	$\mid \texttt{if}\ E\ B\ \texttt{else}\ B$	(`If`)
	$\mid \texttt{while}\ E\ B$	(`While`)
Blocks	$B ::= S; \ldots; S$	(`tuple[Stmt, ...]`)
Programs	$P ::= \texttt{def}\ f(x, \ldots, x)\ B\ \texttt{return}\ E$	(`Prog`)

A program is a single function, of any name, whose body ends with `return`. `return` is not a statement: it may appear only there. There are no calls: the parameters of the function are the input of a program and the returned value is its output.

2 SEMANTIC DOMAINS

Locations	$\ell \in \mathbb{L}$
Values	$v ::= n \mid s \mid \texttt{true} \mid \texttt{false} \mid \texttt{None} \mid \ell$
Heaps	$H \in \mathbb{H} = \mathbb{L} \xrightarrow{\text{fin}} \mathbb{V}^*$
States	$\sigma \in \Sigma = \mathbb{X} \xrightarrow{\text{fin}} \mathbb{V}$

A list is a heap object: a location ℓ whose contents $H(\ell)$ is a finite sequence of values. Two variables holding the same ℓ are aliases.

3 BIG-STEP SEMANTICS

$$\boxed{\langle H, \sigma\rangle \vdash E \Rightarrow \langle v, H'\rangle} \qquad \boxed{\langle H, \sigma\rangle \vdash S \Rightarrow \langle H', \sigma'\rangle} \qquad \boxed{\langle H, \sigma\rangle \vdash B \Rightarrow \langle H', \sigma'\rangle}$$

3.1 Expressions

Literals evaluate to themselves and leave the heap alone:

$$\frac{}{\langle H, \sigma\rangle \vdash n \Rightarrow \langle n, H\rangle} \qquad \frac{}{\langle H, \sigma\rangle \vdash s \Rightarrow \langle s, H\rangle} \qquad \frac{}{\langle H, \sigma\rangle \vdash \texttt{true} \Rightarrow \langle \texttt{true}, H\rangle}$$

$$\frac{}{\langle H, \sigma\rangle \vdash \texttt{false} \Rightarrow \langle \texttt{false}, H\rangle} \qquad \frac{}{\langle H, \sigma\rangle \vdash \texttt{None} \Rightarrow \langle \texttt{None}, H\rangle} \qquad \frac{x \in \mathrm{Domain}(\sigma)}{\langle H, \sigma\rangle \vdash x \Rightarrow \langle \sigma(x), H\rangle}$$

Arithmetic and comparison evaluate the left operand first. `+` is defined on two integers and on two strings (concatenation); `*`, `<`, and `<=` on integers only; `==` and `!=` on any two values that are not locations:

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle n_1, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle n_2, H_2\rangle}{\langle H, \sigma\rangle \vdash E_1 \texttt{ + } E_2 \Rightarrow \langle n_1 + n_2, H_2\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle s_1, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle s_2, H_2\rangle}{\langle H, \sigma\rangle \vdash E_1 \texttt{ + } E_2 \Rightarrow \langle s_1 \cdot s_2, H_2\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle n_1, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle n_2, H_2\rangle}{\langle H, \sigma\rangle \vdash E_1 \texttt{ * } E_2 \Rightarrow \langle n_1 \times n_2, H_2\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle n_1, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle n_2, H_2\rangle}{\langle H, \sigma\rangle \vdash E_1 \texttt{ < } E_2 \Rightarrow \langle n_1 < n_2, H_2\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle n_1, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle n_2, H_2\rangle}{\langle H, \sigma\rangle \vdash E_1 \texttt{ <= } E_2 \Rightarrow \langle n_1 \le n_2, H_2\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle v_1, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle v_2, H_2\rangle \qquad v_1, v_2 \notin \mathbb{L}}{\langle H, \sigma\rangle \vdash E_1 \texttt{ == } E_2 \Rightarrow \langle v_1 = v_2, H_2\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle v_1, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle v_2, H_2\rangle \qquad v_1, v_2 \notin \mathbb{L}}{\langle H, \sigma\rangle \vdash E_1 \texttt{ != } E_2 \Rightarrow \langle v_1 \neq v_2, H_2\rangle}$$

`and` and `or` short-circuit: the right operand is evaluated only when the left one does not decide the result.

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle \texttt{false}, H_1\rangle}{\langle H, \sigma\rangle \vdash E_1 \texttt{ and } E_2 \Rightarrow \langle \texttt{false}, H_1\rangle} \qquad \frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle \texttt{true}, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle v, H_2\rangle}{\langle H, \sigma\rangle \vdash E_1 \texttt{ and } E_2 \Rightarrow \langle v, H_2\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle \texttt{true}, H_1\rangle}{\langle H, \sigma\rangle \vdash E_1 \texttt{ or } E_2 \Rightarrow \langle \texttt{true}, H_1\rangle} \qquad \frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle \texttt{false}, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle v, H_2\rangle}{\langle H, \sigma\rangle \vdash E_1 \texttt{ or } E_2 \Rightarrow \langle v, H_2\rangle}$$

Unary operators:

$$\frac{\langle H, \sigma\rangle \vdash E \Rightarrow \langle n, H'\rangle}{\langle H, \sigma\rangle \vdash \texttt{-}\,E \Rightarrow \langle -n, H'\rangle} \qquad \frac{\langle H, \sigma\rangle \vdash E \Rightarrow \langle \texttt{true}, H'\rangle}{\langle H, \sigma\rangle \vdash \texttt{not}\,E \Rightarrow \langle \texttt{false}, H'\rangle} \qquad \frac{\langle H, \sigma\rangle \vdash E \Rightarrow \langle \texttt{false}, H'\rangle}{\langle H, \sigma\rangle \vdash \texttt{not}\,E \Rightarrow \langle \texttt{true}, H'\rangle}$$

A list literal allocates a fresh location; indexing reads from the heap:

$$\frac{\langle H_{i-1}, \sigma\rangle \vdash E_i \Rightarrow \langle v_i, H_i\rangle \quad (1 \le i \le k) \qquad \ell \notin \mathrm{Domain}(H_k)}{\langle H_0, \sigma\rangle \vdash \texttt{[}E_1, \ldots, E_k\texttt{]} \Rightarrow \langle \ell, H_k[\ell \mapsto (v_1, \ldots, v_k)]\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle \ell, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle n, H_2\rangle \qquad H_2(\ell) = (v_0, \ldots, v_{k-1}) \qquad 0 \le n < k}{\langle H, \sigma\rangle \vdash E_1\texttt{[}E_2\texttt{]} \Rightarrow \langle v_n, H_2\rangle}$$

3.2 Statements

$$\frac{}{\langle H, \sigma\rangle \vdash \texttt{pass} \Rightarrow \langle H, \sigma\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E \Rightarrow \langle v, H'\rangle}{\langle H, \sigma\rangle \vdash x \texttt{ := } E \Rightarrow \langle H', \sigma\texttt{[}x \mapsto v\texttt{]}\rangle}$$

$$\frac{\sigma(x) = \ell \qquad \langle H, \sigma\rangle \vdash E_1 \Rightarrow \langle n, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash E_2 \Rightarrow \langle v, H_2\rangle \qquad H_2(\ell) = (v_0, \ldots, v_{k-1}) \qquad 0 \le n < k}{\langle H, \sigma\rangle \vdash x\texttt{[}E_1\texttt{] := } E_2 \Rightarrow \langle H_2\texttt{[}\ell \mapsto (v_0, \ldots, v_{n-1}, v, v_{n+1}, \ldots, v_{k-1})\texttt{]}, \sigma\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E \Rightarrow \langle \texttt{true}, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash B_1 \Rightarrow \langle H', \sigma'\rangle}{\langle H, \sigma\rangle \vdash \texttt{if } E\ B_1 \texttt{ else } B_2 \Rightarrow \langle H', \sigma'\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E \Rightarrow \langle \texttt{false}, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash B_2 \Rightarrow \langle H', \sigma'\rangle}{\langle H, \sigma\rangle \vdash \texttt{if } E\ B_1 \texttt{ else } B_2 \Rightarrow \langle H', \sigma'\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E \Rightarrow \langle \texttt{false}, H_1\rangle}{\langle H, \sigma\rangle \vdash \texttt{while } E\ B \Rightarrow \langle H_1, \sigma\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash E \Rightarrow \langle \texttt{true}, H_1\rangle \qquad \langle H_1, \sigma\rangle \vdash B \Rightarrow \langle H_2, \sigma_2\rangle \qquad \langle H_2, \sigma_2\rangle \vdash \texttt{while } E\ B \Rightarrow \langle H', \sigma'\rangle}{\langle H, \sigma\rangle \vdash \texttt{while } E\ B \Rightarrow \langle H', \sigma'\rangle}$$

3.3 Blocks

$$\frac{}{\langle H, \sigma\rangle \vdash \varepsilon \Rightarrow \langle H, \sigma\rangle}$$

$$\frac{\langle H, \sigma\rangle \vdash S \Rightarrow \langle H_1, \sigma_1\rangle \qquad \langle H_1, \sigma_1\rangle \vdash B \Rightarrow \langle H', \sigma'\rangle}{\langle H, \sigma\rangle \vdash S\,; B \Rightarrow \langle H', \sigma'\rangle}$$

3.4 Programs

Running $P = \texttt{def } f(x_1, \ldots, x_k)\ B\ \texttt{return } E$ with $v_1, \ldots, v_k$ is

$$\langle \emptyset, \texttt{[}x_i \mapsto v_i\texttt{]}\rangle \vdash B \Rightarrow \langle H', \sigma'\rangle \qquad \langle H', \sigma'\rangle \vdash E \Rightarrow \langle v, H''\rangle,$$

yielding v.

4 DIFFERENCES FROM PYTHON

- **One function.** One `def`: no calls, no nested definitions, no closures; σ is a flat map.
- **`return` only at the end.** Part of the program form, not a statement: no early exit.
- **No builtins.** Input is the parameters; output is the returned value.
- **Comparison chaining.** `1 < 2 < 3` is rejected: comparison is binary.
- **Operator overloading.** Our operators are fixed functions.
- **Object identity.** No `is`; == and != have no derivation on lists.
- **Indexing.** Only $0 \le n < k$; negative indices have no derivation.
- **Errors.** A program with no derivation has no result; there are no exceptions.
- **Missing statements.** No `break`, `continue`, `for`, `try`, `class`, `import`, `lambda`, `dict`, or comprehensions.