

1 CoPylot

1.1 Grammar

The grammar of the language to be analyzed appears in Figure 2.

We assume throughout the rest of this document that a fixed program \hat{S} is under analysis. *(TODO: describe here the idea of a bijection between labels and statements in this fixed program. – ZP)*

1.2 Control Flow

The grammar of control flow graphs appears in Figure 3. *(Discuss construction of initial graph. – ZP)*

We write $\hat{o} \stackrel{?}{\blacktriangleleft} \hat{o}'$ to denote $(\hat{o} \blacktriangleleft \hat{o}' \in \hat{G}$ when \hat{G} is understood from context. Likewise, we write $\hat{o} \stackrel{?}{\ll} \hat{o}'$ to denote $(\hat{o} \ll \hat{o}' \in \hat{G}$ when \hat{G} is understood from context.

We define a relation \longrightarrow^1 to perform control flow graph closure.

Definition 1.7. Let $\hat{G} \longrightarrow^1 \hat{G}'$ be the least relation satisfying the rules appearing in Figure 4. Throughout these rules, the predicates $\stackrel{?}{\ll}$ and $\stackrel{?}{\blacktriangleleft}$ refer to graph \hat{G} .

1.3 Value Lookup

The value lookup function uses the additional grammar in Figure 5.

Definition 1.8. Given a control-flow graph \hat{G} , let $\hat{G}(\hat{o}_0, \hat{K})$ be the function returning the least set \hat{V} which satisfies the following conditions:

1. Value Manipulation

- (a) RESULT
If $\hat{K} = [\hat{v}]$, then $\hat{v} \in \hat{V}$.

2. Variable Lookup

- (a) VALUE DISCOVERY
If $\hat{o}_1 \stackrel{?}{\ll} \hat{o}_0$, $\hat{o}_1 = \hat{\ell}_1 : \hat{\ell}_2 : \hat{x} = \hat{v}$, and $\hat{K} = [\hat{x}] \parallel \hat{K}'$, then $\hat{G}(\hat{o}_1, [\hat{v}] \parallel \hat{K}') \subseteq \hat{V}$.
- (b) VALUE SKIP
If $\hat{o}_1 \stackrel{?}{\ll} \hat{o}_0$, $\hat{o}_1 = \hat{\ell}_1 : \hat{\ell}_2 : \hat{x}' = \hat{v}$, $\hat{K} = [\hat{x}] \parallel \hat{K}'$, and $\hat{x} \neq \hat{x}'$, then $\hat{G}(\hat{o}_1, \hat{K}) \subseteq \hat{V}$.
- (c) VALUE ALIASING
If $\hat{o}_1 \stackrel{?}{\ll} \hat{o}_0$, $\hat{o}_1 = \hat{\ell}_1 : \hat{\ell}_2 : \hat{x} = \hat{x}'$, and $\hat{K} = [\hat{x}] \parallel \hat{K}'$, then $\hat{G}(\hat{o}_1, [\hat{x}'] \parallel \hat{K}) \subseteq \hat{V}$.

x	$::=$	$\text{ALPHANUMERIC} \mid \star\text{ALPHANUMERIC}$	<i>variables</i>
ℓ	$::=$	$\ell \mid \text{END}$	<i>labels</i>
T	$::=$	$[t, \dots]$	<i>stack</i>
t	$::=$	$\ell \times S$	<i>stack frames</i>
S	$::=$	$[s, \dots]$	<i>programs</i>
d	$::=$	$x = e \mid \text{def } x(x, \dots) = \{S\} \mid \text{return } x \mid \text{goto } \ell \mid \text{goto } \ell \text{ if not } x$ $\mid \text{raise } x \mid \text{catch } x$	<i>directives</i>
B	$::=$	$\{x \mapsto m, \dots\}$	<i>bindings</i>
H	$::=$	$\{m \mapsto v, \dots\}$	<i>heap</i>
v	$::=$	$\mathbb{Z} \mid [m, \dots] \mid (m, \dots) \mid B \mid F \mid M \mid \text{undefined} \mid \text{None}$	<i>values</i>
e	$::=$	$v \mid x \mid x(x, \dots) \mid [x, \dots] \mid (x, \dots)$	<i>expressions</i>
P	$::=$	$m \mapsto m$	<i>parental map</i>
m			<i>memory locations</i>
F	$::=$	$\langle m, \text{def } (x) \rightarrow S \rangle \mid \mathfrak{F}$	<i>general functions</i>
M	$::=$	$\langle m, m, \text{def } (x) \rightarrow S \rangle \mid \langle m, \mathfrak{M} \rangle$	<i>general methods</i>
\mathfrak{F}			<i>magic functions</i>
\mathfrak{M}			<i>magic methods</i>

Definition 1.1.

$$\text{BIND}(H, m_0, x, m) = H' \text{ such that } B = H[m_0], B' = B[x \mapsto m], H' = H[m_0 \mapsto B']$$

Definition 1.2.

$$\begin{aligned} \text{BIND}(H, m_0, x_1, m_1, \dots, x_n, m_n) \\ = \text{BIND}(\dots((\text{BIND}(\text{BIND}(H, m_0, x_1, m_1)), m_0, x_2, m_2), \dots), m_0, x_n, m_n) \end{aligned}$$

Definition 1.3.

$$\text{LOOKUP}(m_0, P, H, x_2) =$$

Definition 1.4. $\ell : \ell' : d \in S, \ell' : \ell'' : d' \in S, m' = P[m]$

$$\text{CATCH}(\langle P, \ell, S, m \rangle) = \begin{cases} \ell', x, m' & \text{if } d' = \text{catch } x \\ \text{undefined}, m' & \text{otherwise} \end{cases} \quad (1)$$

Definition 1.5.

$$\text{GETOBJ}(H, m) = \begin{cases} B, & \text{if } v = B \\ B[\star x_{\text{value}} \mapsto v], & \text{otherwise} \end{cases}, H[m] = v \quad (2)$$

Definition 1.6. $H[m][_call_] = m', H[m'] = v$

$$\text{GETCALL}(H, m) = \begin{cases} \text{GETCALL}(H, m'), & \text{if } v = B \\ m', & \text{if } v = F \mid M \end{cases} \quad (3)$$

Figure 1: Operational Semantics

$$\begin{array}{c}
\text{LITERAL ASSIGNMENT} \\
\frac{
\begin{array}{l}
S(\ell) = \ell : \ell' : x = v \\
B_{\text{obj}} = \{\star x_{\text{value}} \mapsto m\} \quad H' = H[m \mapsto v, m' \mapsto B_{\text{obj}}] \\
m, m' \notin H \quad \text{BIND}(H', m_0, x, m') = H'' \quad \ell \stackrel{s}{\blacktriangleleft} \ell^{*''}
\end{array}
}{
\langle \ell, S \rangle \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell^{*''}, S \rangle \parallel T, H'', P, m_0]
} \\
\\
\text{NAME ASSIGNMENT} \\
\frac{
\begin{array}{l}
S(\ell) = \ell : \ell' : x_1 = v_2 \\
m = \text{LOOKUP}(m_0, P, H, x_2) \quad \text{BIND}(H, m_0, x_1, m) = H' \quad \ell \stackrel{s}{\blacktriangleleft} \ell^{*''}
\end{array}
}{
\langle \ell, S \rangle \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell^{*''}, S \rangle \parallel T, H', P, m_0]
} \\
\\
\text{LIST ASSIGNMENT} \\
\frac{
\begin{array}{l}
S(\ell) = \ell : \ell' : x = [x_1, \dots, x_n] \quad \forall i \in \{1, \dots, n\}, m_i = \text{LOOKUP}(m_0, P, H, x_i) \\
v = [m_1, \dots, m_n] \quad B_{\text{obj}} = \{\star x_{\text{value}} \mapsto m\} \quad H' = H[m \mapsto v, m' \mapsto B_{\text{obj}}] \\
m, m' \notin H \quad \text{BIND}(H', m_0, x, m') = H'' \quad \ell \stackrel{s}{\blacktriangleleft} \ell^{*''}
\end{array}
}{
\langle \ell, S \rangle \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell^{*''}, S \rangle \parallel T, H'', P, m_0]
} \\
\\
\text{TUPLE ASSIGNMENT} \\
\frac{
\begin{array}{l}
S(\ell) = \ell : \ell' : x = (x_1, \dots, x_n) \quad \forall i \in \{1, \dots, n\}, m_i = \text{LOOKUP}(m_0, P, H, x_i) \\
v = (m_1, \dots, m_n) \quad B_{\text{obj}} = \{\star x_{\text{value}} \mapsto m\} \quad H' = H[m \mapsto v, m' \mapsto B_{\text{obj}}] \\
m, m' \notin H \quad \text{BIND}(H', m_0, x, m') = H'' \quad \ell \stackrel{s}{\blacktriangleleft} \ell^{*''}
\end{array}
}{
\langle \ell, S \rangle \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell^{*''}, S \rangle \parallel T, H'', P, m_0]
} \\
\\
\text{FUNCTION ASSIGNMENT} \\
\frac{
\begin{array}{l}
S(\ell) = \ell : \ell' : x_1 = \text{def } (x_2, \dots, x_n) = \{S\} \\
v = \langle m_0, \text{def } (x_2, \dots, x_n) \rightarrow S \rangle \\
B_{\text{obj}} = \{\star x_{\text{value}} \mapsto m, \text{--call--} \mapsto \langle m', \mathfrak{M}_{\text{call}} \rangle\} \quad H' = H[m \mapsto v, m' \mapsto B_{\text{obj}}] \\
m, m' \notin H \quad \text{BIND}(H', m_0, x_1, m') = H'' \quad \ell \stackrel{s}{\blacktriangleleft} \ell^{*''}
\end{array}
}{
\langle \ell, S \rangle \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell^{*''}, S \rangle \parallel T, H'', P, m_0]
} \\
\\
\text{FUNCTION CALL ASSIGNMENT} \\
\frac{
\begin{array}{l}
S(\ell) = \ell : \ell' : x_1 = x_2(x_3, \dots, x_n) \quad m_{\text{raw}} = \text{LOOKUP}(m_0, P, H, x_2) \\
m = \text{GETCALL}(H, m_{\text{raw}}) \quad H[m][\star x_{\text{obj}}] = \langle m'_0, \text{def } (x'_3, \dots, x'_n) \rightarrow S' \rangle \\
m'_0 \notin H \quad H' = H[m'_0 \mapsto \{\}] \quad \forall i, 3 \leq i \leq n, m'_i = \text{LOOKUP}(m_0, P, H, x_i) \\
\text{BIND}(H', m'_0, x'_1, m'_1, \dots, x'_n, m'_n) = H'' \\
P' = P \cup \{m'_0 \mapsto m'_0\} \quad S' = [\ell''' : \ell''' : d, \dots]
\end{array}
}{
\langle \ell, S \rangle \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell'', S' \rangle, \langle \ell, S \rangle \parallel T, H'', P', m'_0]
}
\end{array}$$

Figure 1: Operational Semantics (cont.)

METHOD CALL ASSIGNMENT

$$\begin{array}{c}
S(\ell) = \ell : \ell' : x_1 = x_2(x_3, \dots, x_n) \\
m_{\text{raw}} = \text{LOOKUP}(m_0, P, H, x_2) \quad m = \text{GETCALL}(H, m_{\text{raw}}) \\
H[m][\star x_{\text{obj}}] = \langle m'_0, m_{\text{obj}}, \text{def } (x'_3, \dots, x'_n) \rightarrow S' \rangle \quad m''_0 \notin H \\
H' = H[m'_0 \mapsto \{x_{\text{self}} \mapsto m_{\text{obj}}\}] \quad \forall i, 3 \leq i \leq n, m'_i = \text{LOOKUP}(m_0, P, H, x_i) \\
\text{BIND}(H', m'_0, x'_1, m'_1, \dots, x'_n, m'_n) = H'' \\
P' = P \cup \{m'_0 \mapsto m'_0\} \quad S' = [\ell'' : \ell''' : d, \dots] \\
\hline
[\langle \ell, S \rangle] \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell'', S' \rangle, \langle \ell, S \rangle] \parallel T, H'', P', m'_0
\end{array}$$

ATTRIBUTE ASSIGNMENT

$$\begin{array}{c}
S(\ell) = \ell : \ell' : x_1 = x_2.x_3 \quad m = \text{LOOKUP}(m_0, P, H, x_2) \\
H[m] = B \quad B[x_3] = m' \quad B_{\text{obj}} = \text{GETOBJ}(H, m') \\
m'' \notin H \quad H' = H[m'' \mapsto B_{\text{obj}}] \quad \text{BIND}(H', m_0, x_1, m'') = H'' \quad \ell \stackrel{s}{\blacktriangleleft} \ell' \\
\hline
[\langle \ell, S \rangle] \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell', S \rangle] \parallel T, H'', P, m_0
\end{array}$$

(Gets values wrapped in objects. – TC)

RAISE EXCEPTION CAUGHT

$$\begin{array}{c}
S(\ell) = \ell : \ell' : \text{raise } x \quad T = [\langle \ell_1, S_1 \rangle, \dots, \langle \ell_n, S_n \rangle] \quad m_0^0 = m_0 \\
\forall i, 1 \leq i \leq k, k < n, \text{CATCH}(\langle P, \ell_i, S_i, m_0^{i-1} \rangle) = \text{undefined}, \text{undefined}, m_0^i \\
\text{CATCH}(\langle P, \ell_{k+1}, S_{k+1}, m_0^k \rangle) = \ell', x', m'_0 \\
m = \text{LOOKUP}(m'_0, P, H, x) \quad \text{BIND}(H, m_0, x', m) = H' \quad \ell'_s \stackrel{s_{k+1}}{\blacktriangleleft} \ell'' \\
\hline
[\langle \ell, S \rangle] \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell'', S_{k+1} \rangle] \parallel T, H', P, m'_0
\end{array}$$

RAISE EXCEPTION ESCAPED

$$\begin{array}{c}
S(\ell) = \ell : \ell' : \text{raise } x \quad T = [\langle \ell_1, S_1 \rangle, \dots, \langle \ell_n, S_n \rangle] \\
\ell_1 = \ell' \quad \forall i, 1 \leq i \leq n, \text{CATCH}(\langle \ell_i, S_i \rangle) = \text{undefined} \\
\hline
[\langle \ell, S \rangle] \parallel T, H, P, m_0 \longrightarrow^1 [], H, P, m_0
\end{array}$$

PASS

$$\begin{array}{c}
S(\ell) = \ell : \ell' : \text{pass} \quad \ell \stackrel{s}{\blacktriangleleft} \ell'' \\
\hline
[\langle \ell, S \rangle] \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell'', S \rangle] \parallel T, H, P, m_0
\end{array}$$

RETURN

$$\begin{array}{c}
S(\ell) = \ell : \ell' : \text{return } x \\
T = [t, \langle \ell'', S' \rangle] \parallel T' \quad m = \text{LOOKUP}(m_0, P, H, x) \quad m'_0 = P[m_0] \\
S'(\ell'') = \ell'' : \ell''' : x_1 = e \quad \text{BIND}(H, m'_0, x_1, m) = H' \quad \ell'' \stackrel{s'}{\blacktriangleleft} \ell''' \\
\hline
[t, \langle \ell'', S' \rangle] \parallel T', H, P, m_0 \longrightarrow^1 [\langle \ell''', S' \rangle] \parallel T', H', P, m'_0
\end{array}$$

Figure 1: Operational Semantics (cont.)

$$\begin{array}{c}
\text{GOTO} \\
\frac{S(\ell) = \ell : \ell' : \text{goto } \ell'' \quad (\ell'' : \ell''' : d) \in S}{[\langle \ell, S \rangle] \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell'', S \rangle] \parallel T, H, P, m_0} \\
\\
\text{GOTOIFNOT} \\
\frac{S(\ell) = \ell : \ell' : \text{goto } \ell'' \text{ if not } x \quad m = \text{LOOKUP}(m_0, P, H, x) \quad H[m][\star x_{\text{value}}] = \text{FALSE} \quad (\ell'' : \ell''' : d) \in S}{[\langle \ell, S \rangle] \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell'', S \rangle] \parallel T, H, P, m_0} \\
\\
\text{GOTOIFNOT FAILED} \\
\frac{S(\ell) = \ell : \ell' : \text{goto } \ell'' \text{ if not } x \quad m = \text{LOOKUP}(m_0, P, H, x) \quad H[m][\star x_{\text{value}}] = \text{TRUE} \quad \ell \overset{S'}{\blacktriangleleft} \ell''}{[\langle \ell, S \rangle] \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell'', S \rangle] \parallel T, H, P, m_0} \\
\\
\text{NAME STATEMENT} \\
\frac{S(\ell) = \ell : \ell' : e \quad B = H[m_0] \quad \forall x \in e, \exists B[x] \quad \ell \overset{S'}{\blacktriangleleft} \ell''}{[\langle \ell, S \rangle] \parallel T, H, P, m_0 \longrightarrow^1 [\langle \ell'', S \rangle] \parallel T, H, P, m_0} \\
\\
\text{END OF FUNCTION} \\
\frac{T = [\langle \text{END}, S \rangle, \langle \ell, S' \rangle] \parallel T' \quad m'_0 = P[m_0] \quad S'(\ell) = \ell : \ell'' : x = e \quad m \notin H \quad H' = H[m \mapsto \text{None}] \quad \text{BIND}(H', m'_0, x, m) = H'' \quad \ell \overset{S'}{\blacktriangleleft} \ell''}{[\langle \text{END}, S \rangle, \langle \ell, S' \rangle] \parallel T', H, P, m_0 \longrightarrow^1 [\langle \ell'', S' \rangle] \parallel T', H'', P, m'_0} \\
\\
\text{END OF PROGRAM} \\
\frac{T = [\langle \text{END}, S \rangle]}{T, H, P, m_0 \longrightarrow^1 [], H, P, m_0}
\end{array}$$

Figure 1: Operational Semantics (cont.)

$$\begin{array}{ll}
\hat{S} & ::= [\hat{s}, \dots] \quad \text{abstract programs} \\
\hat{s} & ::= \hat{\ell} : \hat{\ell} : \hat{d} \quad \text{abstract statements} \\
\hat{d} & ::= \hat{x} = \hat{v} \mid \hat{x} = \hat{x} \quad \text{abstract directives} \\
\hat{v} & ::= \text{int}^+ \mid \text{int}^- \mid \text{int}^0 \quad \text{abstract values} \\
\hat{x} & \quad \text{abstract variables} \\
\hat{\ell} & \quad \text{abstract labels}
\end{array}$$

Figure 2: Normalized Python Language Grammar

$$\begin{array}{lll}
\hat{G} & ::= & \{\hat{g}, \dots\} & \text{control flow graphs} \\
\hat{g} & ::= & \hat{o} \blacktriangleleft \hat{o} \mid \hat{o} \ll \hat{o} & \text{control flow graph edge} \\
\hat{o} & ::= & \text{START} \mid \text{END} \mid \hat{s} & \text{control flow graph nodes}
\end{array}$$

Figure 3: Control Flow Graph Grammar

$$\begin{array}{ll}
\text{LEXICAL START} & \text{LITERAL ASSIGNMENT} \\
\frac{\text{START} \stackrel{?}{\blacktriangleleft} \hat{o}}{\hat{G} \longrightarrow^1 \hat{G} \cup \{\text{START} \ll \hat{o}\}} & \frac{\hat{o}_1 = (\hat{x} = \hat{v}) \quad \hat{o}_1 \stackrel{?}{\blacktriangleleft} \hat{o}_2}{\hat{G} \longrightarrow^1 \hat{G} \cup \{\hat{o}_1 \ll \hat{o}_2\}} \\
\\
\text{VARIABLE ACCESSIBLE} & \\
\frac{\hat{o}_1 = (\hat{x} = \hat{x}') \quad \hat{o}_1 \stackrel{?}{\blacktriangleleft} \hat{o}_2 \quad \hat{v} \in \hat{G}(\hat{o}_1, [\hat{x}']) \quad \hat{v} \neq \text{UNDEFINED}}{\hat{G} \longrightarrow^1 \hat{G} \cup \{\hat{o}_1 \ll \hat{o}_2\}} &
\end{array}$$

Figure 4: Control Flow Graph Closure

$$\begin{array}{lll}
\hat{K} & ::= & [\hat{k}, \dots] & \text{lookup stacks} \\
\hat{k} & ::= & \hat{x} \mid \hat{v} \mid \text{CAPTURE}(\mathbb{N}) \mid \text{JUMP}(\hat{o}) & \text{lookup stack elements}
\end{array}$$

Figure 5: Value Lookup Grammar