Semantics for $\mathbf{DYNJIT}^{\mathbf{source}} \rightarrow \mathbf{DYNJIT}^{\mathbf{target}}$

July 21, 2020

CoreCPY can be always converted to $\mathbf{DYNJIT}^{\mathbf{source}}$ in a straightforward approach because, **the stack usage of the bytecode generated by valid Python source code is finite**.

Regarding the perspective of partial evaluation, we know **CoreCPY** is a flowchart language with a finite stack(a stack S_N has a size$N$).

By assigning the abstract value **Dyn** s to a named variable s, and assigning the abstract value **Slot** i to a datum stored in the i-th element from the bottom of stack(BOS) if initialized, we will have finite configurations:

$$\mathbf{Configurations} \subset \mathbf{Labels} \times [\mathbf{Dyn}\ s_1, \mathbf{Dyn}\ s_2, \cdots, \mathbf{Dyn}\ s_k]^k \times \mathcal{P}(\{\mathbf{Slot}\ 1, \cdots, \mathbf{Slot}\ N\})$$

where $\mathcal{P}$ means getting the power set, k is the number of named variables not matter what it is(cell, free, etc.) and, N, as we've mentioned above, is the size of the stack, which is finite for any given code.

Note that

$$\left|\mathbf{Labels} \times [\mathbf{Dyn}\ s_1, \mathbf{Dyn}\ s_2, \cdots, \mathbf{Dyn}\ s_k]^k \times \mathcal{P}(\{\mathbf{Slot}\ 1, \cdots, \mathbf{Slot}\ N\}\right|$$

is simply finite, this leads to finite **Configurations**, and also, finite basic blocks when translating the use of stack to some additional registers.

DYNJIT$^{\text{source}}$ and DYNJIT$^{\text{target}}$

DYNJIT$^{\text{source}}$ Syntax

$$
\begin{array}{rcl}
\langle \text{repr} \rangle & ::= & S\ \textbf{constant} \\
& | & D\ \textbf{var} \\
\langle \text{instr} \rangle & ::= & \langle \text{var} \rangle = \textit{call}\ \langle \text{repr} \rangle\ (\langle \text{repr} \rangle\ ^*) \\
& | & \langle \text{var} \rangle = \langle \text{repr} \rangle \\
& | & \textit{return}\ \langle \text{repr} \rangle \\
& | & \textit{goto}\ \textbf{label} \\
& | & \textit{if}\ \langle \text{repr} \rangle\ \textit{goto}\ \textbf{label}\ \textit{goto}\ \textbf{label} \\
\langle \text{move} \rangle & ::= & \langle \text{var} \rangle \leftarrow \langle \text{var} \rangle \\
\langle \Phi \rangle & ::= & \textbf{label}\ \textit{:}\ \langle \text{move} \rangle\ ^* \\
\langle \text{basicblock} \rangle & ::= & \textit{label}\ \textbf{label}\ \textit{:}\ \Phi\ [\ \langle \Phi \rangle\ ^*\ ]\ \langle \text{instr} \rangle\ ^+ \\
\langle \text{entryblock} \rangle & ::= & \textit{label entry :}\ \Phi\ [\ \langle \Phi \rangle\ ^*\ ]\ \langle \text{instr} \rangle\ ^+ \\
\langle \text{basicblocks} \rangle & ::= & \langle \text{entryblock} \rangle\ \langle \text{basicblock} \rangle\ ^*
\end{array}
$$

DYNJIT$^{\text{target}}$ Syntax

$$
\begin{array}{rcl}
\langle \text{absvalue} \rangle & ::= & (\langle \text{repr} \rangle\ ,\ \langle \text{type} \rangle) \\
\langle \text{expr} \rangle & ::= & \langle \text{absvalue} \rangle \\
& | & \textit{call}\ \langle \text{expr} \rangle\ (\langle \text{expr} \rangle\ ^*) \\
\langle \text{stmt} \rangle & ::= & \langle \text{var} \rangle = \langle \text{expr} \rangle \\
& | & \textit{goto}\ \textbf{label} \\
& | & \textit{return}\ \langle \text{expr} \rangle \\
& | & \textit{if}\ \langle \text{expr} \rangle\ \textit{goto}\ \textbf{label}\ \textit{goto}\ \textbf{label} \\
& | & \textit{do}\ \langle \text{expr} \rangle \\
& | & \textit{label}\ \textbf{label} \\
& | & \{\ \langle \text{stmts} \rangle\ \} \\
& | & \textit{switch}\ \langle \text{expr} \rangle\ \langle \text{case} \rangle\ ^* \\
\langle \text{case} \rangle & ::= & |\ \langle \text{type} \rangle \rightarrow \langle \text{stmt} \rangle \\
\langle \text{stmts} \rangle & ::= & \langle \text{stmt} \rangle\ ^*
\end{array}
$$

Types

```
type fptr = int
type meth = int
type t =
| TopT
| BottomT
| NoneT
| FPtrT   of fptr
| MethT   of meth
| NomT    of string * (string, t) map
| TupleT  of t list
| TypeT   of t list
| CellT   of t list
| UnionT  of t list
| IntrinT of intrinsic
```

Intrinsics and **Constants**

```
type intrinsic =
| IsInstanceOf
| TypeOf
| BuildTuple
| Upcast
| Downcast

type const =
| NoneC
| UndefC
| TypeL   of t
| FPtrC   of fptr
| MethC   of meth
| IntC    of int
| FloatC  of float
| StrC    of string
| TupleC  of const list
| IntrinsicC of intrinsic
```