Uber-lazy Symbolic Execution

Neha Rungta
NASA Ames

Eric Mercer and Benjamin Hillery
Brigham Young University

$$
\begin{aligned}
P &::= (\mu\ (C\ m))\\
\mu &::= (CL\ \ldots)\\
T &::= \textbf{bool} \mid C\\
CL &::= (\textbf{class}\ C\ ([T\ f]\ \ldots)\ (M\ \ldots))\\
M &::= (T\ m\ ([T\ x]\ \ldots)\ \mathrm{e})\\
e &::= x\\
&\mid v\\
&\mid (\textbf{new}\ C)\\
&\mid (e\ \$\ f)\\
&\mid (e\ @\ m\ (e\ \ldots))\\
&\mid (e\ \textbf{=}\ e)\\
&\mid (x\ \textbf{:=}\ e)\\
&\mid (x\ \$\ f\ \textbf{:=}\ e)\\
&\mid (\textbf{if}\ e\ e\ \textbf{else}\ e)\\
&\mid (\textbf{var}\ T\ x\ \textbf{:=}\ e\ \textbf{in}\ e)\\
&\mid (\textbf{begin}\ e\ \ldots)\\
x &::= \textbf{this} \mid id\\
f &::= id\\
m &::= id\\
C &::= id\\
v &::= r \mid \textbf{null} \mid \textbf{true} \mid \textbf{false}\\
r &::= \textbf{number}\\
id &::= \textbf{variable-not-otherwise-mentioned}
\end{aligned}
$$

Fig. 1. The Javalite surface syntax.

$$e ::= (\ldots. \mid (\textbf{raw}\ v\ @\ m\ (v\ \ldots)))$$

Fig. 2. The machine syntax for Javalite.

***Abstract*—The abstract goes here.**

I. Pseudo-code

Figure 1 defines the surface syntax for the Javalite language [1]. The Figure 2 is the machine syntax. The semantics of Javalite is syntax based and defined as rewrites on a string. The semantics use a CEKS machine model with a (C)ontrol string representing the expression being evaluated, an (E)nvironment for local variables, a (K)ontinuation for what is to be executed next, and a (S)tore for the heap. This paper only defines salient features of the language and machine relevant to understanding the new algorithm.

- $x \in X$ is a variable
- $f \in F$ is a field name in an object
- $r \in V$ is a reference
- e is an expression.
  - v
  - x
  - $(e\ \$\ f)$
  - $(x\ := e)$
  -

  Expressions can be values v, sequencing $((begin\ e\ \ldots))$, field access $((x.f)$ or $(x.f := e))$, and other constructs to be defined soon,...
- μ is the text that defines classes and methods.
- P is a program $(\mu, (Cm))$ where C is a class defined in μ and m is a method in that class.
- (L, R) represent the heap and form a bipartite graph (see below).
- $\eta : X \rightarrow V$ is an environment
- k is a continuation to resolve evaluation order and return points from calls
  - `end`: nothing comes next
  - $(*\ \$\ f \rightarrow k)$: access field f once expression is reduced to a reference.
- $s = (\mu\ h\ \eta\ e\ k)$ is a machine state.
- $(\mu\ h'\ \eta'\ e'\ k') = \mathrm{execute}(\mu, h, \eta, e, k)$ is a function that returns a state by evaluating the expression.

The heap is a bipartite graph defined on locations L and references R. R and L include the special symbol $\perp$ to indicate an uninitialized reference and location respectively. Additionally, L includes a special location `NULL` for null references.

Locations are boxes in the graphical representation and indicated with the letter l in the math. References are circles in the graphical representation and indicated with the letter r in the math. Edges from locations are labeled with field names $f \in F$. Edges from the references are labeled with constraints $\phi \in \Phi$ (we assume Φ is a power set over individual constraints and phi is a set of constraints for the edge).

The heap is now defined as $h = (L\ R\ \mathit{ref}\ \mathit{loc})$ where

- $\mathit{ref} : L \times F \mapsto R$ is the connection between locations and references on a named field. The function can also be viewed as a set $\mathit{ref} \subseteq L \times F \times R$ in a set if that is preferred to a function.
- $\mathit{loc} : R \mapsto 2^{L \times \Phi}$ is the connection between a reference and a location on a constraint. The function loc can also be viewed as a set $\mathit{loc} \subseteq R \times \Phi \times L$.

Acknowledgment

References

[1] S. O. Wesonga, "Javalite - an operational semantics for modeling Java programs," Master's thesis, Brigham Young University, Provo UT, 2012.

VARIABLE LOOKUP

$$(L\ R\ \eta\ \mathrm{x}\ k) \rightarrow (L\ R\ \eta\ \eta(\mathrm{x})\ k)$$

FIELD ACCESS(EVAL)

$$(L\ R\ \eta\ (e\ \$\ \mathrm{f})\ k) \rightarrow (L\ R\ \eta\ e\ (*\ \$\ \mathrm{f} \rightarrow k))$$

FIELD ACCESS (NULL)

$$\frac{L(r) = \emptyset}{(L\ R\ \eta\ r\ (*\ \$\ \mathrm{f} \rightarrow k)) \rightarrow (L[r \mapsto \{(\bot, \top)\}]\ R\ \eta\ r\ (*\ \$\ \mathrm{f} \rightarrow k))}$$

FIELD ACCESS (NON-NULL)

$$\frac{\begin{array}{c} L(r) = \emptyset \qquad \mathrm{type}(r) = \mathrm{C}_r \qquad \mathrm{fresh}_l(\mathrm{C}_r) = l \\ R' = R[\forall \mathrm{f} \in \mathrm{C}_r\ ((l\ \mathrm{f}) \mapsto \mathit{fresh}_r(\mathit{type}(\mathrm{f})))] \end{array}}{(L\ R\ \eta\ r\ (*\ \$\ \mathrm{f} \rightarrow k)) \rightarrow (L[r \mapsto \{(l, \top)\}]\ R'\ \eta\ r\ (*\ \$\ \mathrm{f} \rightarrow k))}$$

FIELD ACCESS

$$\frac{\begin{array}{l} L(r) \neq \emptyset \qquad \mathrm{type}(r) = C_r \qquad \mathrm{fresh}_r(C_r) = r_f \\ Q = \{(r'\ \phi) \mid (l\ \phi) \in L(r) \wedge r' \in R(l\ \mathrm{f})\} \\ L_f = L[r_f \mapsto \cup_{(r'\ \phi) \in Q}\{(l\ \phi'\phi) \mid (l\ \phi') \in L(r')\}] \end{array}}{(L\ R\ \eta\ r\ (*\ \$\ \mathrm{f} \rightarrow k)) \rightarrow (L_f\ R\ \eta\ r_f\ k)}$$

Fig. 3. Uber-lazy state reductions