Visualizing Abstract Abstract Machines

Kyle Headley  Clark Ren

slides:  kyleheadley.github.io
Visualizing Abstract Abstract Machines

Plan

• Quick intro to Visualiser
• Describe AAM Analysis
• Challenges of AAM
• Secondary Analysis
• Demo Features
• Demo Usage
• Conclusion
Visualizing Abstract Abstract Machines

https://analysisviz.gilray.net/

Login prompt is just for partitioning, share your name with a friend

Default “guest” login has some examples we liked

Select a project in the list

Click graph nodes, read detail in bottom right panes

Click to expand configurations items

https://github.com/harp-lab/aam-visualizer

Scheme workshop, Aug ’19
Abstract Machines
Abstract Machines

(let ([u (lambda(x)(x x))]
      [i (lambda(y) y)])
  ((i i) u))
(let ([u (lambda(x)(x x))]
   [i (lambda(y) y)])
  ((i i) u))
Abstract Machines

\[
\begin{align*}
&\text{let } ([u (\text{lambda}(x)(x \ x))] \\
&\quad [i (\text{lambda}(y) y)]) \\
&\quad ((i \ i) \ u))
\end{align*}
\]

Eval \[
\begin{cases}
\text{let } ([u (\text{lambda}(x)(x \ x))] \\
\quad [i (\text{lambda}(y) y)]) & \text{Env} \quad \text{Stack} \\
\quad ((i \ i) \ u))
\end{cases}
\]

\[
\rightarrow
\begin{cases}
\text{Current} \\
\text{Finished} \\
\text{Next}
\end{cases}
\]

Eval \[
\begin{cases}
\text{lambda}(x)(x \ x)) \quad \emptyset & \{\text{let}, [\text{body}, [i]]\} \\
\quad , \quad , \quad , \quad \text{halt}
\end{cases}
\]
Abstract Machines

(let ([u (lambda(x)(x x))]
       [i (lambda(y) y)])
       ((i i) u))

Eval \(\langle (\lambda(x)(x x)) \emptyset \langle \text{let,} [\text{body}], [i] \rangle \rangle\)

Apply \(\langle \text{app} [i, i] \langle \text{app}, [], [u], \emptyset \rangle \rangle\)

Scheme workshop, Aug ’19
Abstract Machines

(let ([u (lambda(x)(x x))]
       [i (lambda(y) y)])
       ((i i) u))
Abstract Machines

Abstract Machine

- Deluge of Information
- Formalized reduction semantics
- Analysis is list of states
- Potentially Infinite
Abstract Abstract Machines

We need computable analyses

Infinite:
• Stack size
• Addresses for store allocation

Finite:
• Program Expression
• Number of variables
Abstract Abstract Machines

Solution

Infinite:

• Stack size
• Store allocated stack
• Addresses for store allocation

Finite:

• Program Expression
• Number of variables
Abstract Abstract Machines

Solution

Not Infinite:
- Stack size
- Store allocated stack
- Addresses for store allocation

Finite:
- Program Expression
- Number of variables

Use variables and expressions as addresses
Deal with the implications of this approximation
Abstract Abstract Machines

Store:

\[ y \mapsto (\text{list } i \ u) \]

Succeeding Machine States:

\[ \text{Apply } \langle \text{app}, [i,i], (\text{app},[],[u],), \text{halt} \rangle \]

\[ \text{Eval } \langle y, y = \text{closure}(\bar{i}), (\text{app},[],[u],), \text{halt} \rangle \]

\[ \text{Eval } \langle y, y = \text{closure}(u), (\text{app},[],[u],), \text{halt} \rangle \]

Soundness through non-determinism
Abstract Abstract Machines

**AAM:**
- Unify sources of unboundedness
- Finitize the set of *abstract* addresses
- Soundly model nondeterminism

**Benefits of using AAM:**
- Systematic methodology for analysis
- Good story for tunability and precision
  (E.g., Sensitivity/Polyvariance, P4F)
- 0-CFA, 1-CFA, etc…

VanHorn and Might. 2010
Challenges of working with AAM

Abstract Abstract Machine

• Deluge of data; no obvious summary
• Trade precision for soundness
• Spurious states/values
• Current research on how to tune the imprecision
Challenges of working with AAM

Tuning Imprecision with instrumentation

\[ \text{Eval} \langle \ldots, u = \text{closure}, i = \text{closure}, [(i \ i)], \ldots \rangle \]
Challenges of working with AAM

AAM Analysis 1-CFA

(let ([u (\lambda(x)(x x))]
      [i (\lambda(y) y)])
  ((i i) u))
Challenges of working with AAM

AAM Analysis - 0-CFA

(let ([u (lambda(x)(x x))]
      [i (lambda(y) y)])
  ((i i) u))
Challenges of working with AAM

AAM Analysis - 0-CFA

(create and evaluate closures)

Result

evaluate id function (x2)

spurious ((x x)(x x))

store contains \( y \mapsto (\text{list } i \ u) \)

loop to make second call

(let ([u (lambda(x)(x x))]
      [i (lambda(y) y)])
   ((i i) u))
Segmentation Algorithm

Simplifying a CFG:

- Separate functions
- Create individual CFGs
- Summarize connections
AAM Analysis - 0-CFA

\[
\text{let } \left(\begin{array}{l}
[u \ (\lambda (x)(x 
      x))]
[i \ (\lambda (y) 
      y)]

((i \ i) \ u))
\right)
\]

spurious \((x \ x)(x \ x)\)

store contains \(y \mapsto (\text{list i u})\)

loop to make second call

evaluate id function \((x2)\)

create and evaluate closures

Result
Segmentation Algorithm

AAM Analysis - 1-CFA

(let ([u (lambda(x) (x x))])
  ([i (lambda(y) y)])
  ((i i) u))
Segmentation Algorithm

(let ([u (lambda(x)(x x))]
      [i (lambda(y) y)])
  ((i i) u))

Generate function CFG
Segmentation Algorithm

\[
\text{let} \ (\text{[[u \ (\text{lambda}(x)(x \ x))\]]} \\
\text{[i \ (\text{lambda}(y) \ y)]}) \\
(\text{((i \ i) \ u)})
\]

Generate function CFG

Entry Point

Call

Entry Point
Segmentation Algorithm

(let ([u (lambda(x) (x x))])
  [i (lambda(y) y)])
((i i) u))
Segmentation Algorithm

(let ([u (lambda(x)(x x))]
       [i (lambda(y) y)])
   ((i i) u))

Generate function CFG

Combine Markers
Segmentation Algorithm

Hide functions

(let ([u (lambda(x)(x x))] [i (lambda(y) y)]) ((i i) u))

Return

Exit Point

Return

Exit Point

Scheme workshop, Aug ’19
Segmentation Algorithm

Hide functions

(let ([u (lambda(x)(x x))]
       [i (lambda(y) y)])
       ((i i) u))

Lookup Exits

Call
Segmentation Algorithm

Hide functions

(let ([u (lambda(x)(x x))]
  [i (lambda(y) y)])
  ((i i) u))

Call

Lookup Exits

Scheme workshop, Aug ’19
Demo

https://analysisviz.gilray.net/
https://github.com/harp-lab/aam-visualizer

Additional features:

• Navigation
• Code highlighting
• Linked environments
Future Improvements

- More highlighting
- Improved stack visualizer
- More language features
- Additional Navigation options
- Suggestions?
Conclusion

Analyze an Abstract Machine

Evaluate:

\[
\begin{align*}
\text{Eval} \left\{ \lambda x (x \ x) \ , \ (\text{let},[\text{body}],[i]) \ , \ \text{halt} \right\} \\
\rightarrow \rightarrow \rightarrow \rightarrow
\end{align*}
\]

Apply:

\[
\begin{align*}
\text{app} \ , \ [i,i] \ , \ (\text{app},[],[u],\text{halt}) \\
\rightarrow \rightarrow \rightarrow \rightarrow
\end{align*}
\]

Finite analysis means a complex, imprecise graph

Segment into functions

Visualize!

30 eval 27 exit

[Diagram of a graph showing evaluation steps]

analysisviz.gilray.net

github.com/harp-lab/aam-visualizer

kyleheadley.github.io

Scheme workshop, Aug ’19