Visualizing Abstract Abstract Machines

Kyle Headley *Clark Ren*

slides: **kyleheadley.github.io**

Scheme workshop, Aug ’19

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

Abstract Machines

Abstract Machines

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

Abstract Machines

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

```
Eval⟨ (let ([u (lambda(x)(x x))]
            [i (lambda(y) y)])
         ((i i) u))            , Env , Stack ⟩
```

Abstract Machines

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

Eval ⟨ `(let ([u (lambda(x)(x x))] [i (lambda(y) y)]) ((i i) u))` , Env , Stack ⟩

→

Current

Finished

Next

Eval ⟨ `(lambda(x)(x x))` , ∅ , ⟨`let`,[*body*],[*i*]⟩ `halt` ⟩

Abstract Machines

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

$$\textbf{Eval}\left\langle \texttt{(lambda(x)(x x))}\ ,\ \varnothing\ ,\ \begin{array}{r}\langle\texttt{let},[\textit{body}],[\textit{i}]\rangle\\ \texttt{halt}\end{array} \right\rangle$$

$$\longrightarrow\!\!\!\!\longrightarrow\longrightarrow\!\!\!\!\longrightarrow$$

$$\textbf{Apply}\left\langle \texttt{app}\ ,\ [\textit{i},\textit{i}]\ ,\ \begin{array}{r}\langle\texttt{app},[\,],[\textit{u}],{}^{\texttt{u}}_{\texttt{i}}\rangle,\\ \texttt{halt}\end{array} \right\rangle$$

Scheme workshop, Aug '19

Abstract Machines

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

Apply ⟨ **app** , [*i*,*i*] , ⟨**app**,[],[*u*], $\substack{\texttt{u}\\ \texttt{i}}$⟩, **halt** ⟩

→ (**lambda**(y) y),∅

Extend Environment

Eval ⟨ **y** , **y = closure(*i*)** , ⟨**app**,[],[*u*], $\substack{\texttt{u}\\ \texttt{i}}$⟩, **halt** ⟩

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` $\longmapsto$ `(list` i u`)`

Succeeding Machine States:

`Apply` $\langle$ `app` , `[`i`,`i`]` , `⟨app,[],[`u`], ⟩,` `halt` $\rangle$

`Eval` $\langle$ `y` , `y = closure(`i`)` , `⟨app,[],[`u`], ⟩,` `halt` $\rangle$

`Eval` $\langle$ `y` , `y = closure(`u`)` , `⟨app,[],[`u`], ⟩,` `halt` $\rangle$

Soundness through non-determinism

Scheme workshop, Aug ’19

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

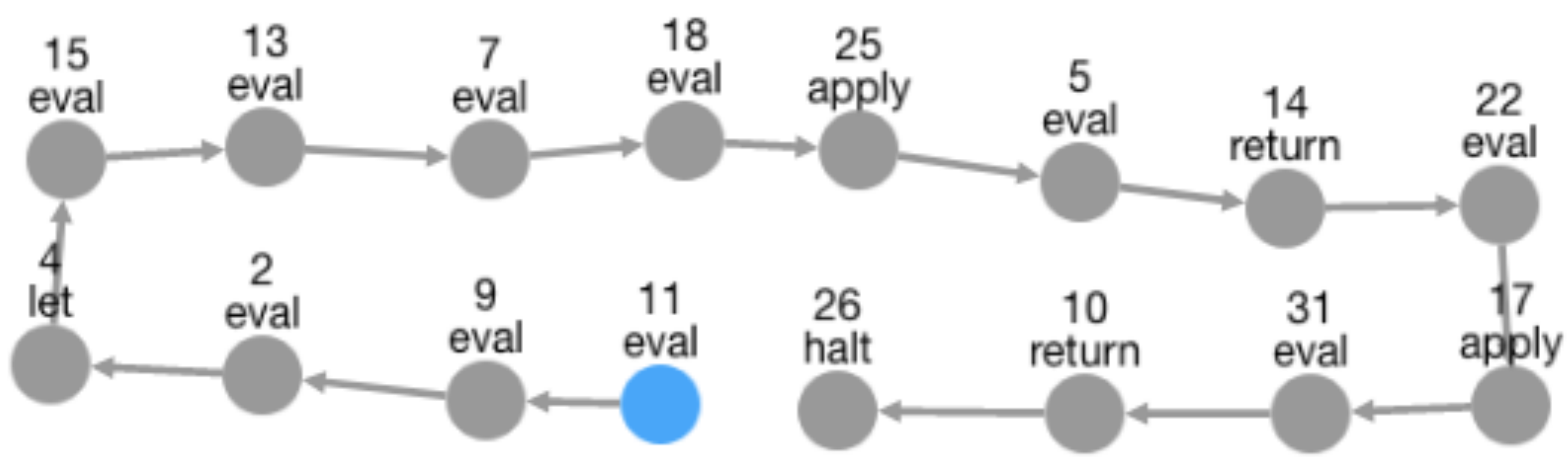
Instrumentation

$$\texttt{Eval}\left\langle \texttt{…}\ ,\ \begin{array}{l}\texttt{u = closure}\\ \texttt{i = closure}\end{array}\ ,\ \texttt{[(i i)]}\ ,\texttt{…}\right\rangle$$

Challenges of working with AAM

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

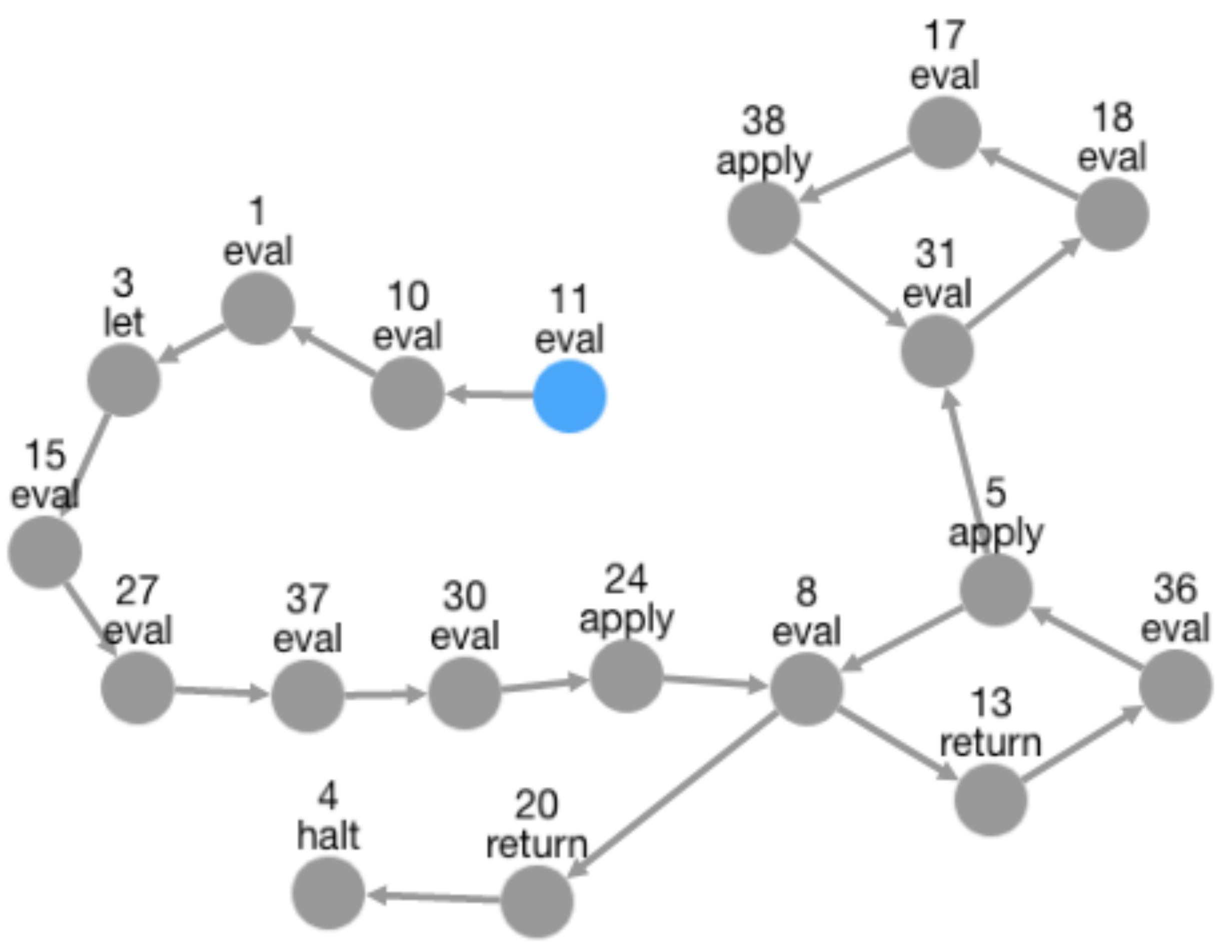
AAM Analysis 1-CFA



Challenges of working with AAM

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

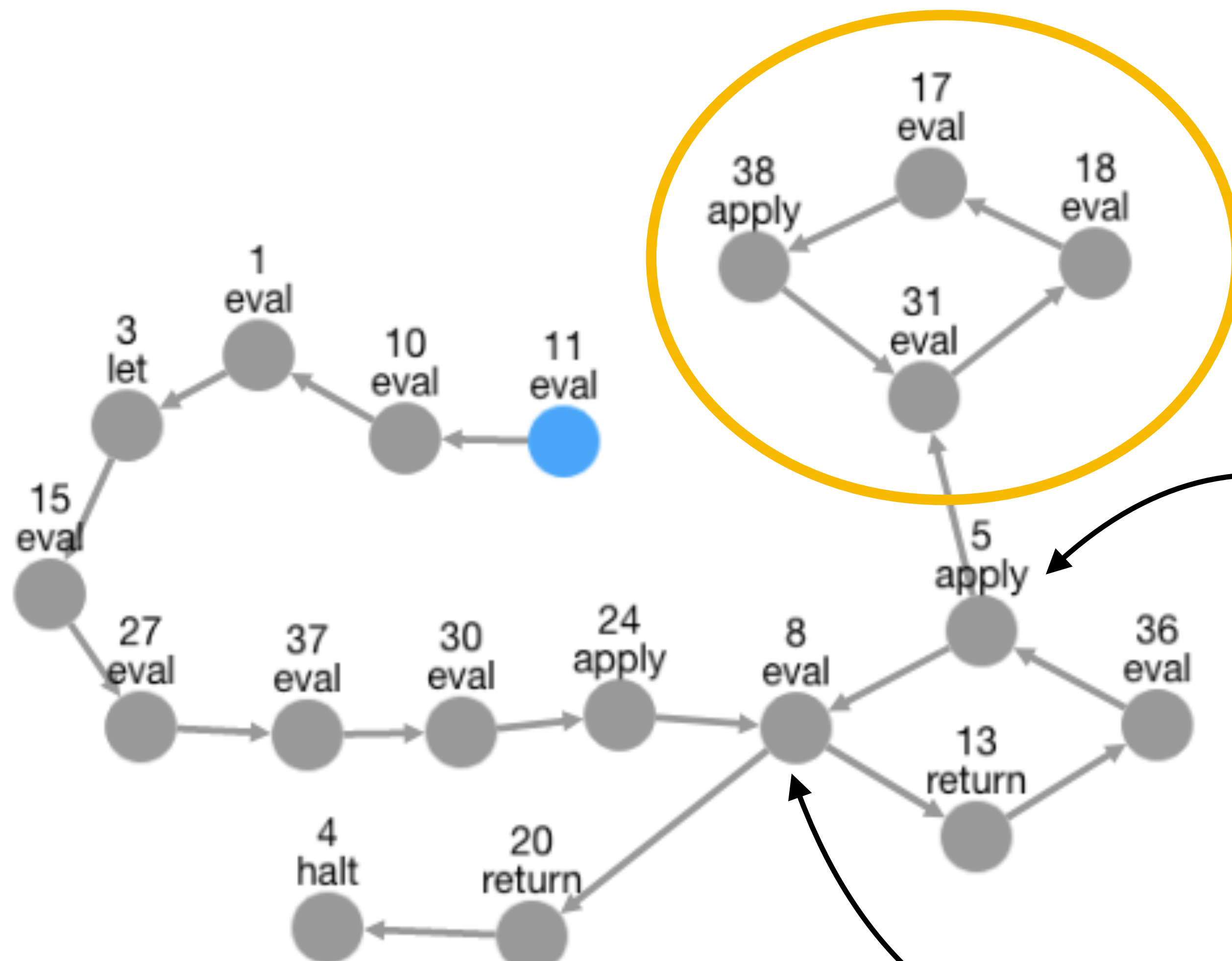
AAM Analysis - 0-CFA



Challenges of working with AAM

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

AAM Analysis - 0-CFA



create and evaluate closures

spurious `((x x)(x x))`

store contains `y` $\longmapsto$ `(list i u)`

loop to make second call

Result

evaluate id function (x2)

Segmentation Algorithm

Simplifying a CFG:

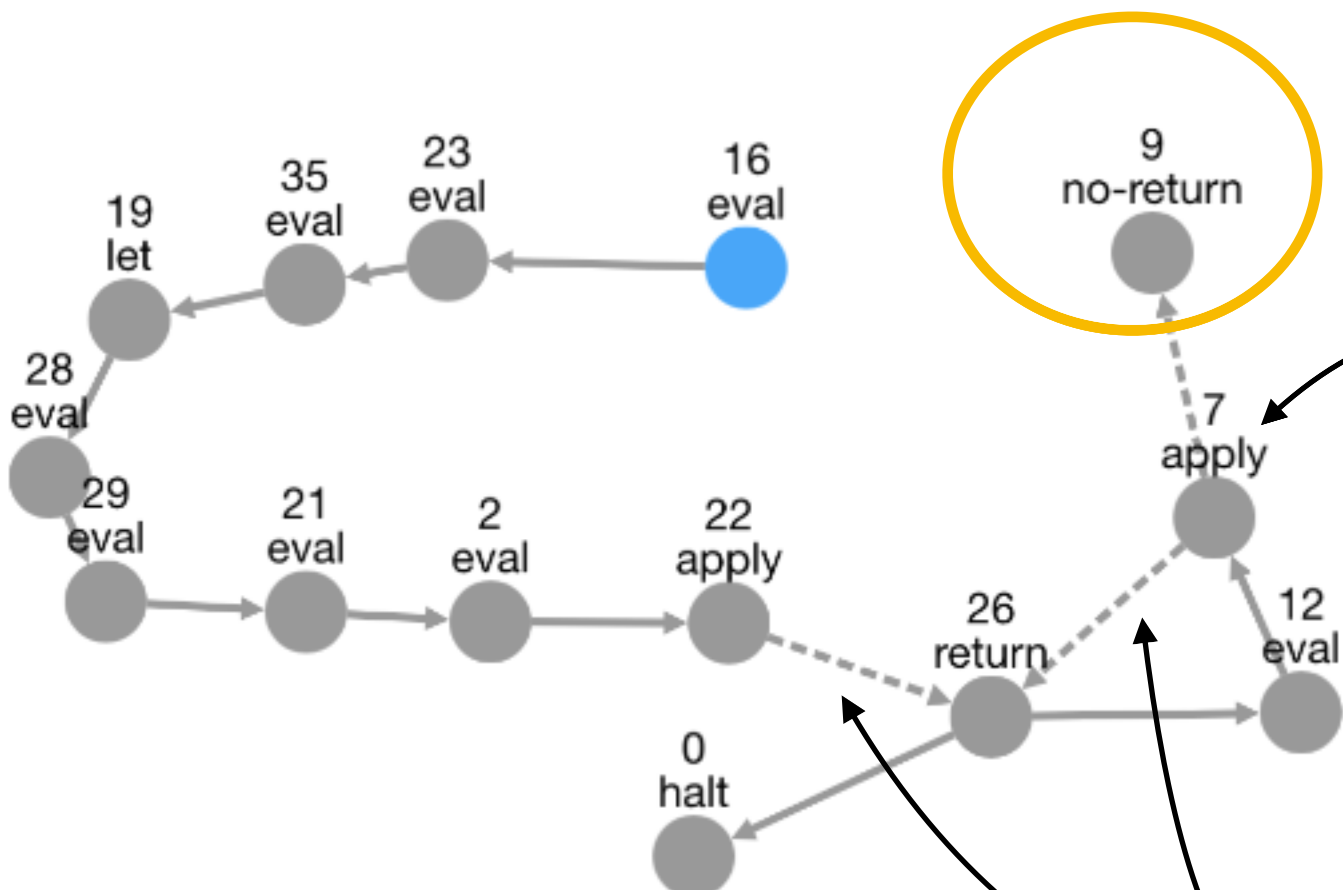
- Separate functions
- Create individual CFGs
- Summarize connections

Segmentation Algorithm

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

AAM Analysis - 0-CFA

create and evaluate closures



spurious `((x x)(x x))`

store contains `y` $\longmapsto$ `(list i u)`

loop to make second call

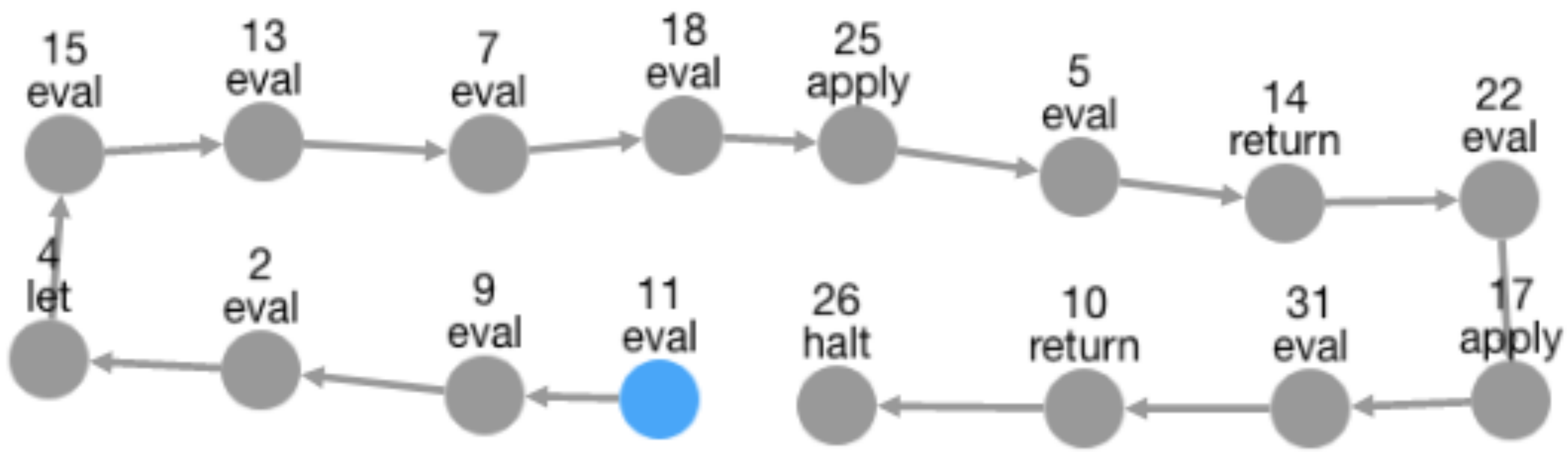
Result

evaluate id function (x2)

Segmentation Algorithm

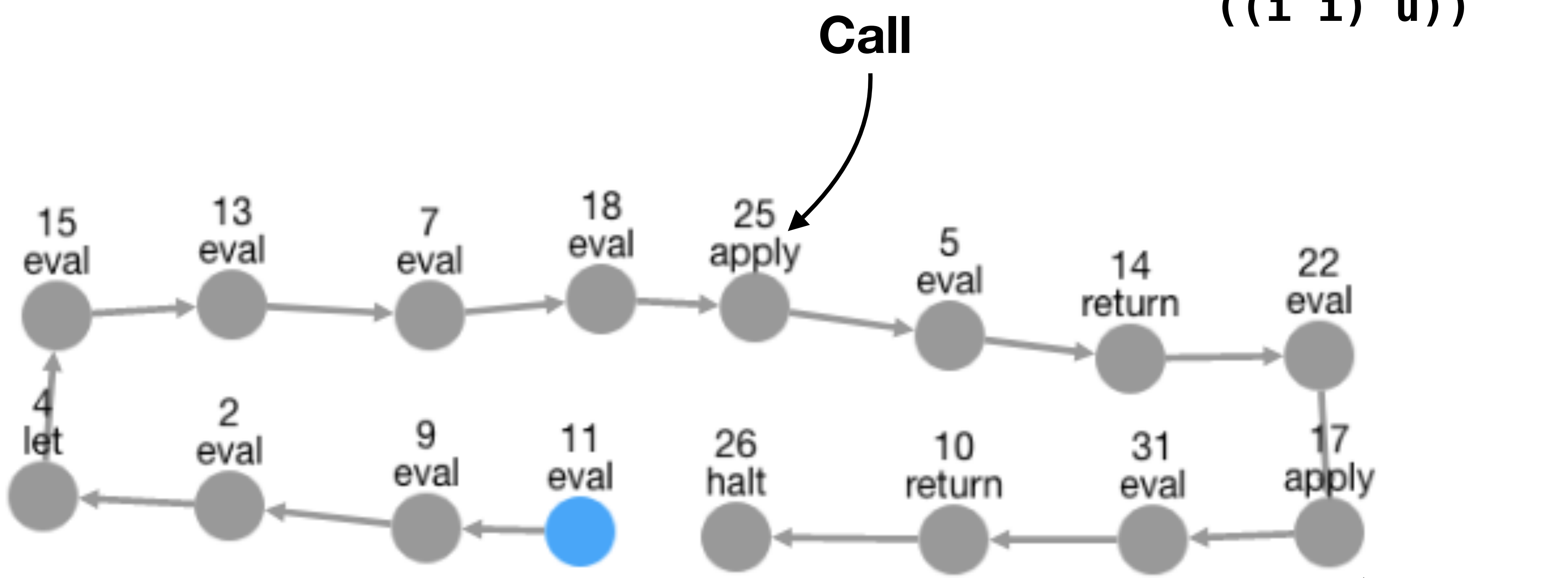
AAM Analysis - 1-CFA

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



Segmentation Algorithm

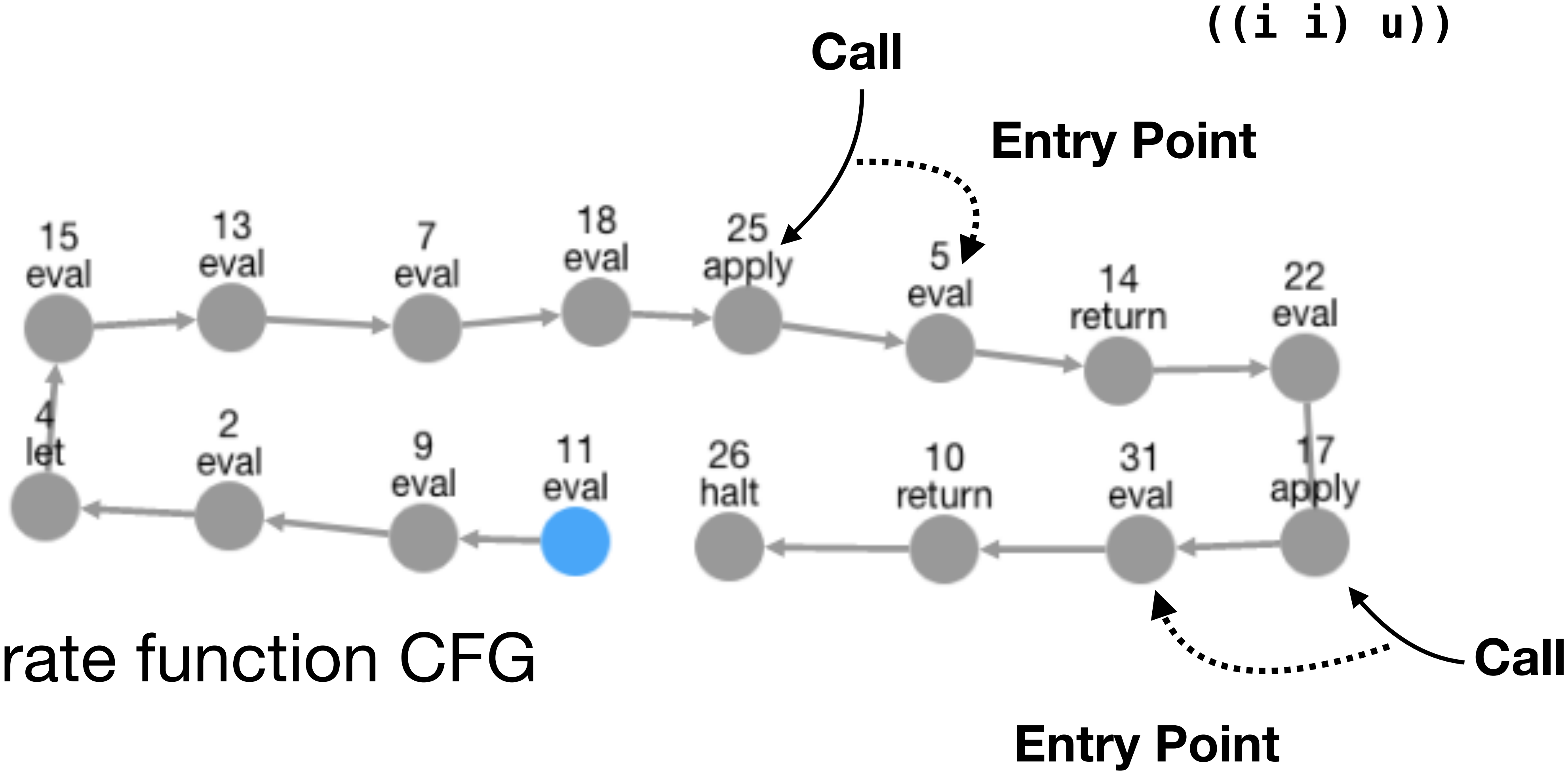
```
(let ([u (lambda(x)(x x))]
      [i (lambda(y) y)])
  ((i i) u))
```



Generate function CFG

Segmentation Algorithm

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



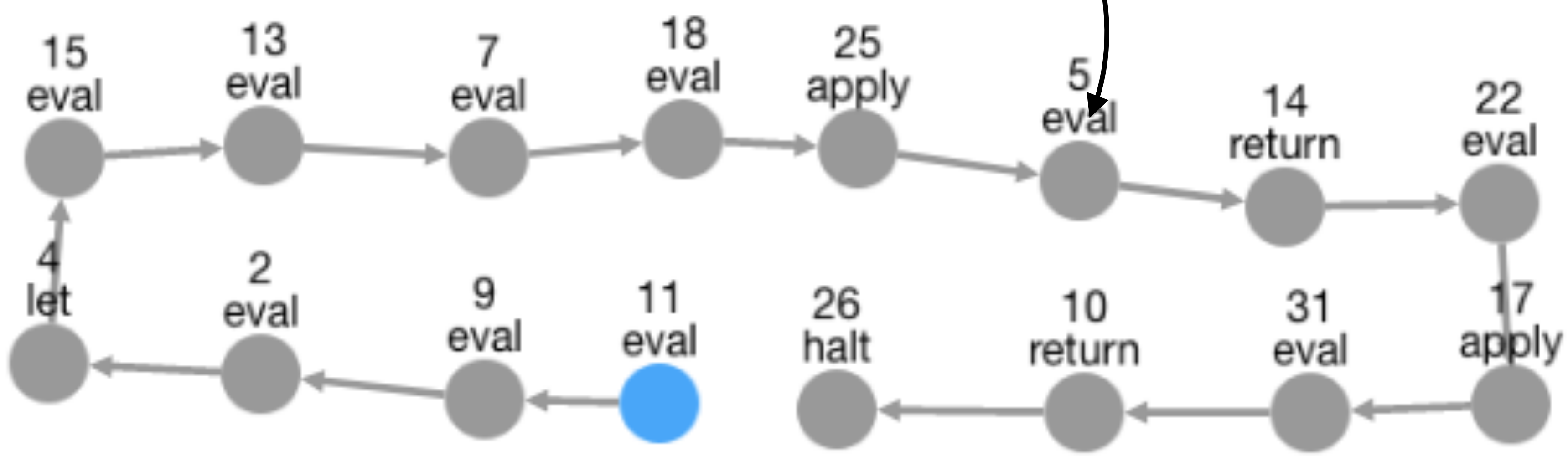
Generate function CFG

Scheme workshop, Aug '19

Segmentation Algorithm

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

Marker



Generate function CFG

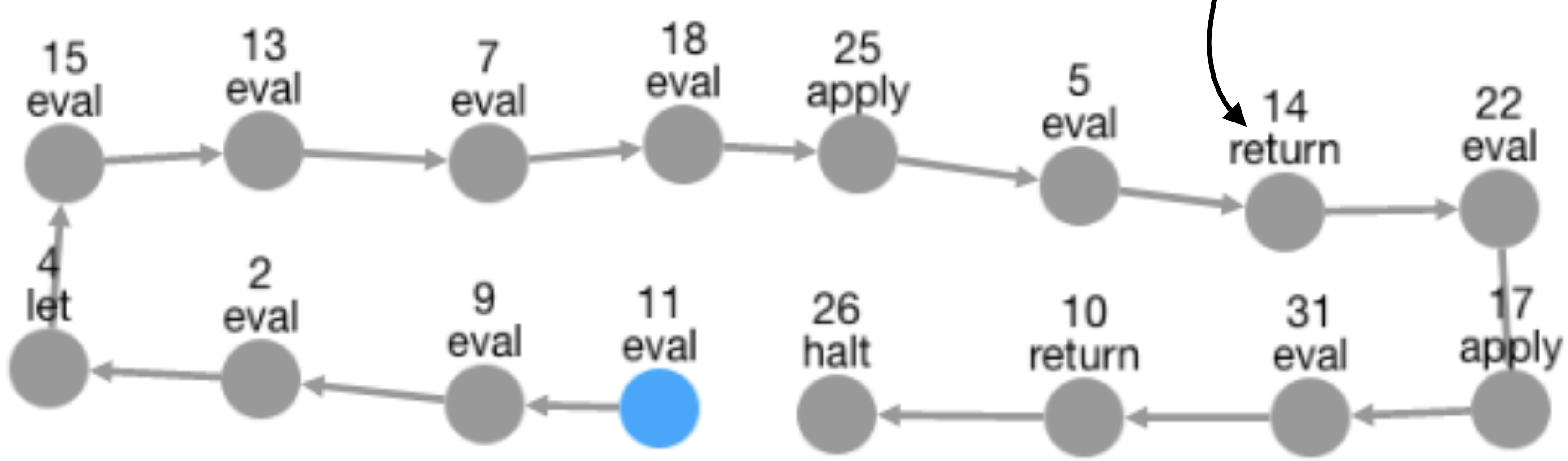
Marker

Combine Markers

Segmentation Algorithm

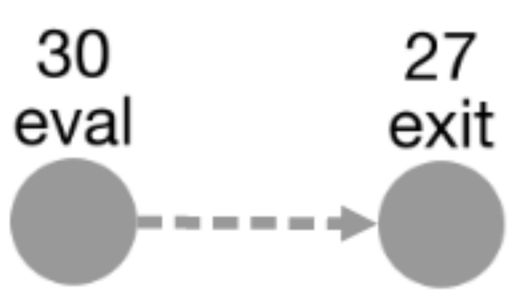
```
(let ([u (lambda(x)(x x))]
      [i (lambda(y) y)])
   ((i i) u))
```

Marker



Marker

Generate function CFG



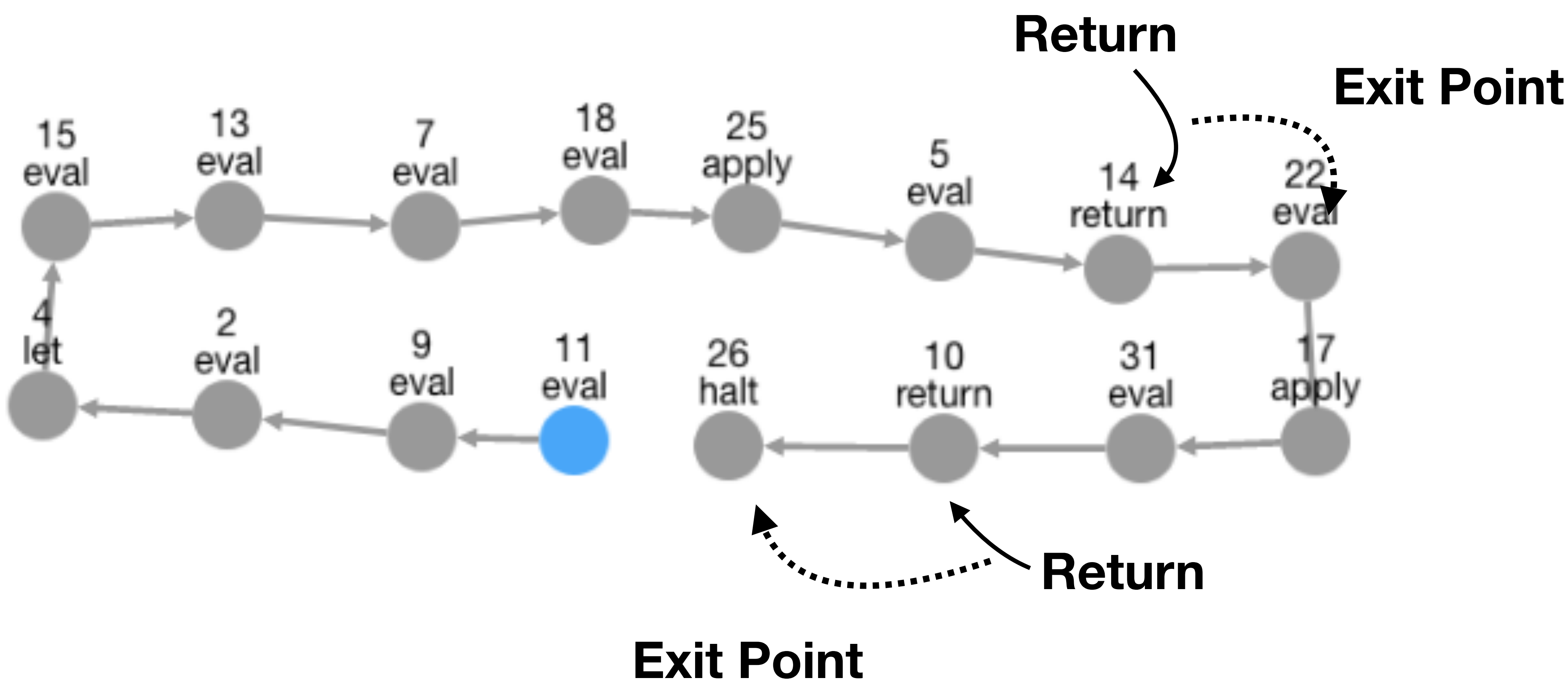
Combine Markers

Scheme workshop, Aug ’19

Segmentation Algorithm

Hide functions

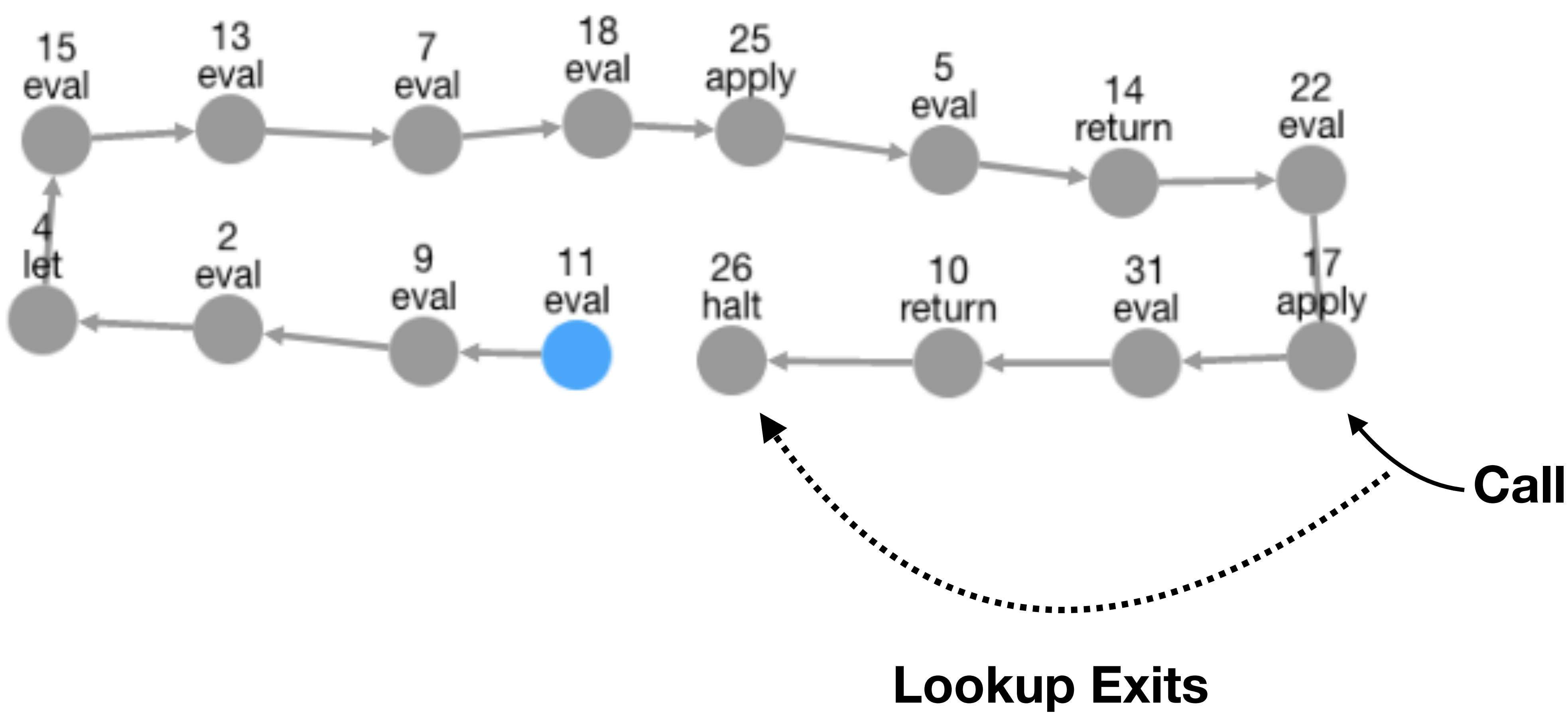
```
(let ([u (lambda(x)(x x))]
      [i (lambda(y) y)])
  ((i i) u))
```



Segmentation Algorithm

Hide functions

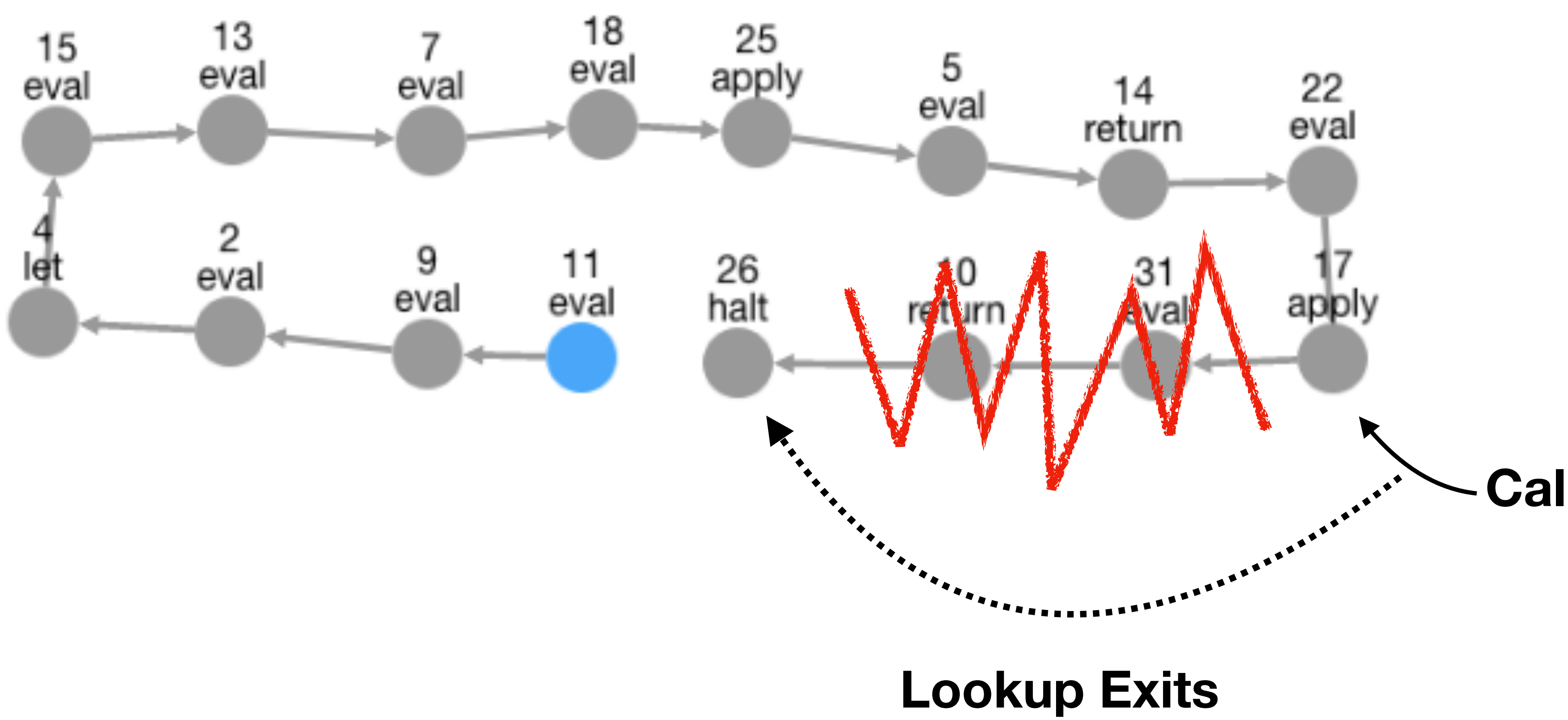
```
(let ([u (lambda(x)(x x))]
      [i (lambda(y) y)])
  ((i i) u))
```



Segmentation Algorithm

Hide functions

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



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

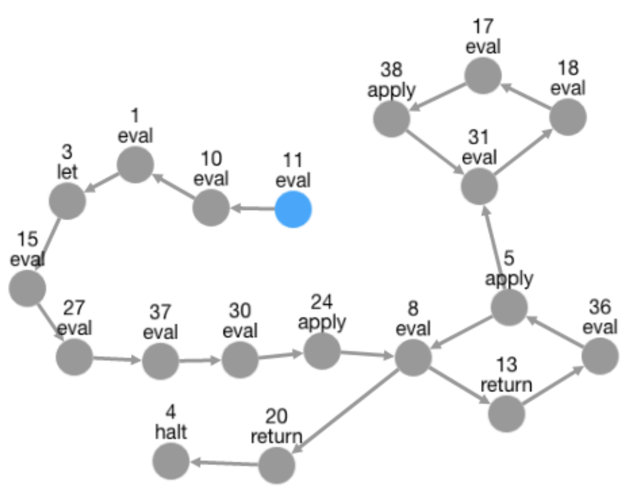
Analyse an Abstract Machine

Eval ⟨(lambda(x)(x x)) , ∅ , (let,[body],[i]) halt⟩

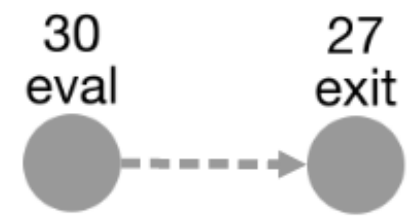
→ → → →

Apply ⟨ app , [i,i] , (app,[],[u],), halt⟩

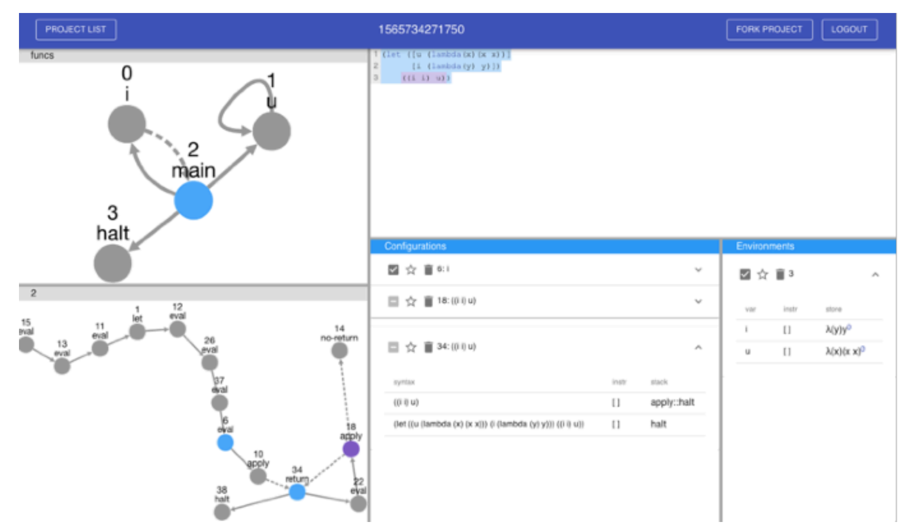
Finite analysis means a complex, imprecise graph



Segment into functions



Visualize!



analysisviz.gilray.net

github.com/harp-lab/aam-visualizer

kyleheadley.github.io

Scheme workshop, Aug ’19