Syracuse University

So You Want To Analyze Scheme Programs With Datalog?

Davis Ross Silverman, Yihao Sun, Kristopher Micinski: Syracuse University

Thomas Gilray: University of Alabama, Birmingham

Introduction

- A Control Flow Analysis for a usable subset of Scheme.
  - Utilizing AAM and m-CFA
- A Datalog implementation that maps closely to operational semantics of a small step machine.
  - Utilizing the Souffle Datalog engine
- An evaluation of running our analysis with worst case terms

Control Flow In Scheme

```
(let* ([f (foo 42)]
       [g (bar 99)])
  (if (= a b) (g 30) (f g)))
```

What is the Control Flow Graph?

$$\begin{array}{l} \boxed{a = b} \nearrow g? \\ \phantom{\boxed{a = b}} \searrow f? \end{array}$$

g and f are not known statically

Abstract Interpretation

Abstract interpretation computes results approximately.

UNKNOWN

```
(if (= a b)
    1
    2)
```

Abstract interpret

{1,2}

When a conditional can not be known during abstract interpretation, both branches must be considered correct.

Abstract Machines

- Used to define a ‘model of computation’, in our case, interpreters.
- Discrete time steps with mathematically defined transition rules
- We utilize an adapted CESK machine.

C **E** **S** **K**

Control Environment Store Continuation

Abstracting The Abstract Machine

- Writing a sound and decidable analysis for a complex Abstract Machine is non trivial
- “Abstracting Abstract Machines” and soundness
- To follow AAM:
    - Remove recursion from environment
    - Store allocate continuations
    - Finitize the address domains and environments allocation functions (tick/alloc)
- In return, you get a sound and decidable analysis for your abstract machine

Van Horn, D., & Might, M. (2010, September). Abstracting abstract machines. In Proceedings of the 15th ACM SIGPLAN international conference on Functional programming (pp. 51-62).

Environments As Context

- Environments as a set of variables
- Environments as precision of states
- Context changes through variable binding.
    - Adding variables adds to the context.
    - In m-CFA, returning from a binding site will pop the context.
- Precision in Control Flow Analyses is generally tuned to small numbers (0, 1, 2).

Context Changes With Bound Variables

```
(let ([a #f])
  (let ([b 9])
    ((λ (x) x)
     (if a 0 b)if0)call0)let1)let0
```

The context is: $\underbrace{\texttt{[call0, let1, let0]}}_{m=3}$

Datalog Rules And Horn Clauses

Datalog: `cousin(a, c) :- parent(a, p), sibling(p, q), parent(c, q).`

Horn Clause: `cousin(a, c)  □  parent(a, p) ∧ sibling(p, q) ∧ parent(c, q)`

> Datalog syntax is based on Horn-SAT

```
ancestor(p, a) :- parent(p, a).
ancestor(p, b) :- ancestor(p, a), parent(a, b).
```

```
ancestor(p, a)  □  parent(p, a)
ancestor(p, b)  □  ancestor(p, a) ∧ parent(a, b)
```

Operational Semantics

$$
\begin{aligned}
e \in \mathrm{Exp} &::= æ \\
&\mid (\mathtt{if}\ e\ e\ e) \mid (\mathtt{set!}\ x\ e) \\
&\mid (\mathtt{call/cc}\ e) \mid let \\
&\mid (op\ e\ e) \mid (e\ e\ e\ \ldots) \\
æ \in \mathrm{AExp} &::= x \mid lam \mid b \mid n \\
x \in \mathrm{Var} &\triangleq \text{The set of identifiers} \\
let \in \mathrm{Let} &::= (\mathtt{let}\ ((x\ e)\ \ldots)\ e) \\
lam \in \mathrm{Lam} &::= (\lambda\ (x)\ e) \\
op \in \mathrm{Prim} &\triangleq \text{The set of primitives}
\end{aligned}
$$

$$
\begin{aligned}
&E\langle(\mathtt{if}\ e_g\ e_t\ e_f), \widehat{ctx}, \hat{\sigma}, \hat{a}_{\hat{\kappa}}\rangle \leadsto E\langle e_g, \widehat{ctx}, \hat{\sigma}', \hat{a}'_{\hat{\kappa}}\rangle \\
&\quad \text{where } \hat{a}'_{\hat{\kappa}} \triangleq \widehat{alloc_k}(\hat{\varsigma}, e_c, \widehat{ctx}) \qquad \textbf{(E-If)} \\
&\qquad \hat{\kappa} \triangleq \widehat{\mathbf{ifk}}(e_t, e_f, \widehat{ctx}, \hat{a}_{\hat{\kappa}}) \\
&\qquad \hat{\sigma}'_{\hat{\kappa}} \triangleq \hat{\sigma}_{\hat{\kappa}} \sqcup [\hat{a}'_{\hat{\kappa}} \mapsto \hat{\kappa}]
\end{aligned}
$$

$$
\begin{aligned}
&E\langle let, \widehat{ctx}, \hat{\sigma}, \hat{a}_{\hat{\kappa}}\rangle \leadsto E\langle e_i, \widehat{ctx}, \hat{\sigma}', \hat{a}'_{\hat{\kappa}}\rangle \\
&\quad \text{where } \widehat{ctx}' \triangleq \widehat{new}(\hat{\varsigma}) \qquad \textbf{(E-Let)} \\
&\qquad let = (\mathtt{let}\ ((x_0\ e_0)\ (x_s\ e_s)\ \ldots)\ e_b) \\
&\qquad (x_i, e_i) \in ([x_0 : x_s], [e_0 : e_s]) \\
&\qquad \hat{a}_v \triangleq \widehat{alloc_v}(x_i, \hat{\varsigma}) \\
&\qquad \hat{a}'_{\hat{\kappa}} \triangleq \widehat{alloc_k}(\hat{\varsigma}, e_i, \widehat{ctx}) \\
&\qquad \hat{\kappa} \triangleq \widehat{\mathbf{let}}(e_b, \hat{a}_v, \widehat{ctx}', \hat{a}_{\hat{\kappa}}) \\
&\qquad \hat{\sigma}'_{\hat{\kappa}} \triangleq \hat{\sigma}_{\hat{\kappa}} \sqcup [\hat{a}'_{\hat{\kappa}} \mapsto \hat{\kappa}]
\end{aligned}
$$

Operational Semantics

$$A\langle \hat{v}, \hat{\sigma}, \hat{a}_{\hat{\kappa}} \rangle \rightsquigarrow E\langle e_t, \widehat{ctx}, \hat{\sigma}, \hat{a}'_{\hat{\kappa}} \rangle$$
$$\text{where } \hat{\sigma}_{\hat{\kappa}}(\hat{a}_{\hat{\kappa}}) \ni \widehat{\mathbf{ifk}}(e_t, \_, \widehat{ctx}, \hat{a}'_{\hat{\kappa}}) \qquad \textbf{(A-IfT)}$$
$$\hat{v} \neq \mathbf{\#f}$$

$$A\langle \hat{v}, \hat{\sigma}, \hat{a}_{\hat{\kappa}} \rangle \rightsquigarrow E\langle e_f, \widehat{ctx}, \hat{\sigma}, \hat{a}'_{\hat{\kappa}} \rangle$$
$$\text{where } \hat{\sigma}_{\hat{\kappa}}(\hat{a}_{\hat{\kappa}}) \ni \widehat{\mathbf{ifk}}(\_, e_f, \widehat{ctx}, \hat{a}'_{\hat{\kappa}}) \qquad \textbf{(A-IfF)}$$
$$\hat{v} = \mathbf{\#f}$$

$$A\langle \hat{v}, \hat{\sigma}, \hat{a}_{\hat{\kappa}} \rangle \rightsquigarrow E\langle e_b, \widehat{ctx}, \hat{\sigma}', \hat{a}'_{\hat{\kappa}} \rangle$$
$$\text{where } \hat{\sigma}_{\hat{\kappa}}(\hat{a}_{\hat{\kappa}}) \ni \widehat{\mathbf{let}}(e_b, \hat{a}_v, \widehat{ctx}, \hat{a}'_{\hat{\kappa}}) \qquad \textbf{(A-Let)}$$
$$\hat{\sigma}'_{\hat{v}} \triangleq \hat{\sigma}_{\hat{v}} \sqcup [\hat{a}_v \mapsto \hat{v}]$$

$$A\langle \hat{v}, \hat{\sigma}, \hat{a}_{\hat{\kappa}} \rangle \rightsquigarrow E\langle e_b, \widehat{ctx}, \hat{\sigma}'', \hat{a}'_{\hat{\kappa}} \rangle$$
$$\text{where } \hat{\sigma}_{\hat{\kappa}}(\hat{a}_{\hat{\kappa}}) \ni \widehat{\mathbf{fn}}(\widehat{clo}, n, \widehat{ctx}, \hat{a}'_{\hat{\kappa}}) \qquad \textbf{(A-Call)}$$
$$\widehat{clo} = ((\lambda\ (x_s \ldots)\ e_b), \widehat{ctx}_{\widehat{clo}})$$
$$\hat{a}_v \triangleq \widehat{alloc}_v(x_n, \hat{\varsigma})$$
$$\hat{\sigma}'_{\hat{v}} \triangleq \widehat{copy}(\widehat{ctx}_{\widehat{clo}}, \widehat{ctx})$$
$$\hat{\sigma}''_{\hat{v}} \triangleq \hat{\sigma}_{\hat{v}} \sqcup [\hat{a}_v \mapsto \hat{v}]$$

Flattened Facts

We compile Scheme to CSV files

```
.decl let(Id: id, BindId: id, BodyId: id)
.input let
.decl let_list(Id: id, X: symbol, EId: id)
.input let_list
```

(let ([a 1^{e0}]
      [b 2^{e1}])$^{l\_l1}$
  a^{body})^{let0}

let.facts

let0	l_l1	body

let_list.facts

l_l1	a	e0
l_l1	b	e1

From Semantics To Code

$$E\langle (\texttt{if}\ e_g\ e_t\ e_f), \widehat{ctx}, \hat{\sigma}, \hat{a}_{\hat{\kappa}}\rangle \leadsto E\langle e_g, \widehat{ctx}, \hat{\sigma}', \hat{a}'_{\hat{\kappa}}\rangle$$

$$\text{where } \hat{a}'_{\hat{\kappa}} \triangleq \widehat{alloc_k}(\hat{\varsigma}, e_c, \widehat{ctx}) \qquad \textbf{(E-If)}$$

$$\hat{\kappa} \triangleq \widehat{\mathbf{ifk}}(e_t, e_f, \widehat{ctx}, \hat{a}_{\hat{\kappa}})$$

$$\hat{\sigma}'_{\hat{\kappa}} \triangleq \hat{\sigma}_{\hat{\kappa}} \sqcup [\hat{a}'_{\hat{\kappa}} \mapsto \hat{\kappa}]$$

```
state_e(eguard, ctx, ak),
stored_kont(ak, kont),
flow_ee(e, eguard) :-
   state_e(e, ctx, bk),
   if(e, eguard, et, ef),
   ak = $KAddress(eguard, ctx),
   kont = $If(et, ef, ctx, bk).
```

CFA Worst Case Analysis

```
((lambda (f)
    (let ((mm (f M)
          ...
          (m1 (f 1))
          (n0 (f 0)))
     mm))
 (lambda (z)
   (^call0(lambda (x) (+ z (+ z ...)))
       (lambda (x) x)^lam_x))^lam_z)
```

1-CFA loses precision!!!!!

```
Evaluated to:
{ (+ 0 0 ....) (+ 0 1 ....)
   ....
   (+ 0 M ....) (+ M 0 ....)
   .... }
```

Running Time

Term Size	Polyvariance (m)	0 Padding		1 Padding		2 Padding	
		Time	Memory	Time	Memory	Time	Memory
32/4	0	00:09:57	1.27GB	00:09:46	1.86GB	00:09:48	1.27GB
	1	< 1 sec	12.1MB	00:22:49	1.28MB	00:13:26	1.27GB
	1	< 1 sec	12.29MB	< 1 sec	12.5MB	00:19:09	1.27GB
86/3	0	01:08:58	3.55GB	01:09:36	3.55GB	01:02:51	3.56GB
	1	<1 sec	6.31MB	01:34:37	6.32MB	01:32:10	3.56GB
	2	<1 sec	6.31MB	<1 sec	6.32MB	02:13:10	3.56GB

Running time and memory usage of m-CFA in Datalog.
Term size N/K means N calls to f and K invocations of +.

Parallel Performance

	1 Thread		2 Threads		4 Threads		8 Threads	
m	time	memory	time	memory	time	memory	time	memory
0	09:48	1.27GB	13:05	1.43GB	18:39	1.55GB	22:38	1.61GB
1	13:26	1.27GB	14:36	1.43GB	20:36	1.55GB	22:54	1.62GB
2	19:09	1.27GB	25:14	1.46GB	36:11	1.54GB	46:57	1.60GB

Parallel performance of m-CFA Souffle implementation.
32 let clauses with 2 levels of padding.

Souffle's multi core execution is broken….. 😢

Future Work And Conclusion

- Implement with a more parallelizable Datalog dialect.
- Extend the semantics with more novel features e.g. Delimited Continuations.

- Abstracting Abstract Machines gives us the tools to write analyses for complex abstract machines.
- Datalog can be used to implement these semantics, and provides useful guarantees for an analysis.

Syracuse University

Q&A