Developing Reasonable Programs

Matt Might
University of Utah
matt.might.net
The future of...
The future of...

...programs
The future of...

...programs

...languages
The future of...

...programs

...languages

...compilers
WARNING
DSLs

Static Analysis
DSLs

Static Analysis
</shortversion>
\[ R = \frac{2v^2 \cos(\theta) \sin(\theta)}{g} \]
35 divisions per second.
2.9 divisions per second.
Performance mattered.
Performance still matters.
## CHECKING ACCOUNT: 300545668

### Expenditure by Category

- Home: 11%
- Food: 37%
- Gas: 6%
- Credit Card: 10%
- Entertainment: 30%

### ACCOUNT CATEGORIES

<table>
<thead>
<tr>
<th>Category</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>$872.40</td>
</tr>
<tr>
<td>Food</td>
<td>$226.00</td>
</tr>
<tr>
<td>Gas</td>
<td>$137.50</td>
</tr>
<tr>
<td>Credit Card</td>
<td>$850.00</td>
</tr>
<tr>
<td>Entertainment</td>
<td>$245.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$2,330.90</strong></td>
</tr>
</tbody>
</table>

### TRANSACTIONS

<table>
<thead>
<tr>
<th>Type</th>
<th>Date</th>
<th>Description</th>
<th>Category</th>
<th>Beginning Balance</th>
<th>Amount</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10/1/09</td>
<td>Rent</td>
<td>Home</td>
<td>$775.00</td>
<td></td>
<td>$4,650.00</td>
</tr>
<tr>
<td>2</td>
<td>10/15/09</td>
<td>Utilities</td>
<td>Home</td>
<td>$97.40</td>
<td></td>
<td>$3,875.60</td>
</tr>
<tr>
<td>3</td>
<td>10/16/09</td>
<td>Fill up SUV for camping trip</td>
<td>Gas</td>
<td>$75.00</td>
<td></td>
<td>$3,702.60</td>
</tr>
<tr>
<td>4</td>
<td>10/22/09</td>
<td>Groceries</td>
<td>Food</td>
<td>$101.00</td>
<td></td>
<td>$3,601.60</td>
</tr>
<tr>
<td>5</td>
<td>10/24/09</td>
<td>Dinner with Paul and Jane</td>
<td>Food</td>
<td>$125.00</td>
<td></td>
<td>$3,476.60</td>
</tr>
<tr>
<td>6</td>
<td>10/25/09</td>
<td>Movies</td>
<td>Entertainment</td>
<td>$35.00</td>
<td></td>
<td>$3,441.60</td>
</tr>
<tr>
<td>7</td>
<td>10/29/09</td>
<td>Insurance refund</td>
<td>Deposit</td>
<td>$155.00</td>
<td></td>
<td>$3,576.00</td>
</tr>
<tr>
<td>8</td>
<td>10/30/09</td>
<td>Paycheck</td>
<td>Deposit</td>
<td>$1,525.00</td>
<td></td>
<td>$5,101.00</td>
</tr>
<tr>
<td>9</td>
<td>11/1/09</td>
<td>Fill up SUV again</td>
<td>Gas</td>
<td>$62.50</td>
<td></td>
<td>$5,038.50</td>
</tr>
<tr>
<td>10</td>
<td>11/1/09</td>
<td>Credit card payment</td>
<td>Credit Card</td>
<td>$850.00</td>
<td></td>
<td>$4,189.10</td>
</tr>
<tr>
<td>11</td>
<td>11/1/09</td>
<td>Security deposit return</td>
<td>Deposit</td>
<td>$300.00</td>
<td></td>
<td>$4,489.10</td>
</tr>
<tr>
<td>12</td>
<td>11/2/09</td>
<td>Night on the town</td>
<td>Entertainment</td>
<td>$210.00</td>
<td></td>
<td>$4,279.10</td>
</tr>
</tbody>
</table>
Correctness matters.
Correctness really matters.
Security matters.
What makes software slow, buggy and insecure?
We can’t predict it.
We can’t reason.
We can’t engineer.
Software “engineering”
A fatal exception 0E has occurred at 0137:BFFA21C9. The current application will be terminated.

* Press any key to terminate the current application.
* Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue
The system has recovered from a serious error.

A log of this error has been created.

Please tell Microsoft about this problem.
We have created an error report that you can send to help us improve Microsoft Windows. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

Send Error Report  Don't Send
We need engineering.
We need reasonable programs.
We need prediction.
So, why can’t we predict what software will do?
So, why can’t we predict what software will do?

Because Alan Turing said we can’t.

Halt!
“Thou shalt not write a program which determines whether a program halts.”
while $P(x)$
Interesting question?
Interesting question?
Undecidable.
But,
there’s a loop hole...
there's a loop hole...

...in the loop hole.
Yes
?
No
The static analysis game
The static analysis game
The static analysis game

MAX++ -> *0 -> *a++ =
The static analysis game

\( \text{MAX}++ \)

\( \ast a++ = \ast 0 \)
The static analysis game

MAX++

*a++ =

*0
Another way?
Don’t use Turing machines.
Static analysis

Sub-Turing languages
How do you play the static analysis game?
How to approximate?
Make it finite!
\( q_1 \) \( (1,0,L) \) \( q_2 \) \( (0,1,R) \) \( q_3 \)

\( (0,1,R) \)

\( \{0,1\} \)
\[
q_1 \xrightarrow{(1,0,L)} q_2 \xrightarrow{(0,1,R)} q_3
\]

\[
\{0,1\}
\]
Why is static analysis hard?
What happens here?

animal.eat(food);
What happens here?

What is animal?

animal.eat(food);

What is food?
What happens here?

```java
void process (Animal animal) {
    food = world.gather();
    animal.eat(food);
}
```
What happens here?

Who calls `process`?

```java
void process (Animal animal) {
    food = world.gather() ;
    animal.eat(food);
}
```

What is `world`?
Control-flow

Data-flow
Why so entangled?
Value = Object
Value = Object
      = Class + Record
Value = Object
    = Class + Record
    ⊆ Code + Data
Old idea:
Untie code & data.
(In ten minutes)
What language exemplifies code + data?
$\lambda$-calculus.
λ-calculus (Church, 1928)
λ-calculus (Church, 1928)

- Minimalist, universal language

Alonzo Church
\( \lambda \)-calculus (Church, 1928)

- Minimalist, universal language
- Three expression types:
  - \( v \) [variable]
  - \( e_1(e_2) \) [function application]
  - \( \lambda v. e \) [anonymous function]
\(\lambda\)-calculus (Church, 1928)

- Minimalist, universal language
- Three expression types:
  - \(v\) [variable]
  - \(e_1(e_2)\) [function application]
\textbf{\( \lambda \)-calculus (Church, 1928)}

- Minimalist, universal language
- Three expression types:
  
  \( v \) [variable]

  \( e_1( e_2 ) \) [function application]

  \( \lambda v. e \) [anonymous function]
\((\lambda x. x^2)(3) = 9\)
Lisp and Scheme

- \( v \equiv v \)
- \( f(e) \equiv (f \ e) \)
- \( \lambda v.e \equiv (\text{lambda } (v) \ e) \)
Python

• $v \equiv v$

• $f(e) \equiv f(e)$

• $\lambda v. e \equiv \text{lambda } v: e$
Ruby

• \( v = v \)
• \( f(e) = f(e) \)
• \( \lambda \, v . \, e = \text{lambda} \{ \mid v \mid \text{ return e } \} \)
JavaScript

- $v \equiv v$
- $f(e) \equiv f(e)$
- $\lambda v. e \equiv \text{function} \ (v) \ {\text{\{ return e ; \}}}$
• \( v \equiv v \)

• \( f(e) \equiv f.\text{call}(e) \)

• \( \lambda v. e \equiv \text{new Value}() \{ \text{public Value call (Value v) \{ return e \}} \} \)
λ-fortified

- Lisp
- SML
- Haskell
- Scala
- Java
- C#
- C++
- Python
- Ruby
- Smalltalk
- JavaScript
- PHP(!)
Value = Closure
Value = Closure
      = Lambda + Env
Value = Closure
= Lambda + Env
⊆ Code + Data
Assertion: If we can do λ’s, we can do objects.
How to bound control?
Control-flow question

Given a call site $f(x)$, what could $f$ be?
\( f(x) \)
let f = \(z \mapsto z\)
in f(x)
\( \lambda f. f(x) \)
Classical approach
The approximation

• Value = Code x Data
• Closure = Lambda x Env
• Object = Class x Record
The approximation

- Value = Code
- Closure = Lambda
- Object = Class
How do $\lambda$’s flow?
$e_1(e_2)$
$\lambda v. e_b$\n
$e_1(e_2)$
\( \lambda v. e_b \)

\( e_1(e_2) \)

val
\( \lambda v.e_b \)

\( e_1(e_2) \)

val
\[
\lambda v. e_b \in \text{FlowsTo}[e_1] \quad \text{and} \quad \text{val} \in \text{FlowsTo}[e_2]
\]
\[
\text{val} \in \text{FlowsTo}[v]
\]
\( \lambda v. e_b \)
\[ \lambda v. e_b \]

\[ e_1(e_2) \]

\[ \text{val} \]
$\lambda v. e_b \in \text{FlowsTo}[e_1]$ and $\text{val} \in \text{FlowsTo}[e_b]$
\( \lambda v. e_b \in \text{FlowsTo}[e_1] \) and \( \text{val} \in \text{FlowsTo} [e_b] \)  
val \in \text{FlowsTo}[e_1(e_2)]
\(\lambda v.e_b \in \text{FlowsTo}[\lambda v.e_b]\)

\(\lambda v.e_b \in \text{FlowsTo}[e_1]\) and \(\text{val} \in \text{FlowsTo}[e_b]\)

\(\text{val} \in \text{FlowsTo}[e_1(e_2)]\)

\(\lambda v.e_b \in \text{FlowsTo}[e_1]\) and \(\text{val} \in \text{FlowsTo}[e_2]\)

\(\text{val} \in \text{FlowsTo}[v]\)
0CFA (Shivers, 1988)

\[
\{\lambda v.e_b\} \subseteq \text{FlowsTo}[\lambda v.e_b]
\]

\[
\lambda v.e_b \in \text{FlowsTo}[e_1] \\
\text{FlowsTo}[e_b] \subseteq \text{FlowsTo}[e_1(e_2)]
\]

\[
\lambda v.e_b \in \text{FlowsTo}[e_1] \\
\text{FlowsTo}[e_2] \subseteq \text{FlowsTo}[v]
\]
0CFA (Shivers, 1988)

\[ \{ \lambda v. e_b \} \subseteq \text{FlowsTo}[\lambda v. e_b] \]

\[ \lambda v. e_b \in \text{FlowsTo}[e_1] \]

\[ \text{FlowsTo}[e_b] \subseteq \text{FlowsTo}[e_1(e_2)] \]

\[ \lambda v. e_b \in \text{FlowsTo}[e_1] \]

\[ \text{FlowsTo}[e_2] \subseteq \text{FlowsTo}[v] \]
0CFA (Shivers, 1988)

\[ \{ \lambda v.e_b \} \subseteq \text{FlowsTo}[\lambda v.e_b] \]

\[ \lambda v.e_b \in \text{FlowsTo}[e_1] \]
\[ \text{FlowsTo}[e_b] \subseteq \text{FlowsTo}[e_1(e_2)] \]

\[ \lambda v.e_b \in \text{FlowsTo}[e_1] \]
\[ \text{FlowsTo}[e_2] \subseteq \text{FlowsTo}[v] \]

+ Constraint Solver

= Control-flow analysis
But...
It’s slow.
It’s weak.
It’s imprecise.
Problem: Cross-flow

map f list
Problem: Cross-flow

fireMissile(n)  []

map f list

petBunny(n)  [1,2,3]
Problem: Cross-flow

```java
fireMissile(n) []
map f list
```

```java
petBunny(n) [1,2,3]
```
Problem: Cross-flow

```plaintext
fireMissile(n)

[]

map f list

petBunny(n) [1,2,3]
```
Problem: Cross-flow

- `fireMissile(n)`
- `map f list` with an empty list `[]`
- `petBunny(n)` with the list `[1,2,3]`
Problem: Cross-flow

- `fireMissile(n)`
- `petBunny(n)`
- `map f list` with input `[]` and output `[1,2,3]`
Problem: Cross-flow

fireMissile(n) => map f list => petBunny(n) => [1,2,3]
Problem: Cross-flow

```plaintext
fireMissile(n) → []
map f list
petBunny(n) ← [1,2,3]
```
Problem: Cross-flow

\[
\text{fireMissile}(n) \quad \rightarrow \quad []
\]

\[
\text{map \ f \ list}
\]

\[
\text{petBunny}(n) \quad \leftarrow \quad [1,2,3]
\]
Problem: Cross-flow

fireMissile(n) \rightarrow \emptyset

map f list

petBunny(n) \rightarrow [1,2,3]
No attention to order.
Monotonic.
A different approach: Small-step analysis

(Joint work with David Van Horn)
Easier to understand.
Simpler to derive.
Faster to compute.
A program is an infinite state machine.
An analysis is a finite state machine.
Small-step machine
Small-step machine

- Convert program \( e \) into machine state \( s_0 \)
Small-step machine

• Convert program $e$ into machine state $s_0$

• Transition from state $s_n$ to state $s_{n+1}$
Analysis machine

\[ e \downarrow \]

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow \ldots \]
Analysis machine

e
S₀ → S₁ → S₂ → S₃ → S₄ → ...
Analysis machine

\[ \xrightarrow{e} s_0 \quad \xrightarrow{\hat{s}_0} s_0 \quad \xrightarrow{s_1} s_1 \quad \xrightarrow{s_2} s_2 \quad \xrightarrow{s_3} s_3 \quad \xrightarrow{s_4} s_4 \quad \cdots \]
Analysis machine
Analysis machine

\[ \begin{align*}
&\overset{e}{s_0} \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow \ldots \\
&\overset{\hat{s}_0}{\hat{s}_0} \rightarrow \hat{s}_1 \rightarrow \hat{s}_2 \rightarrow \hat{s}_3 \rightarrow \hat{s}_4
\end{align*} \]
Analysis machine
Theorem: The analysis simulates the machine.
“Concrete”  “Abstract”
Small-step analysis...

Infinite state-space
Small-step analysis...

Infinite state-space

Finite state-space
...is bounded graph search.
...is bounded graph search.

Finite state-space
Example:
Small steps for CPS
Continuation-passing style

\[ f, e \in \text{Exp} = \text{Var} + \text{Lam} + \text{App} \]
Continuation-passing style

\[ f, e \in \text{Exp} = \text{Var} + \text{Lam} \]
Continuation-passing style

\[ f, e \in \text{Exp} = \text{Var} + \text{Lam} \]

\[ \text{lam} \in \text{Lam} ::= (\lambda (v_1 \ldots v_n) \text{call}) \]
Continuation-passing style

\[
f, e \in \text{Exp} = \text{Var} + \text{Lam} \\
lam \in \text{Lam} ::= (\lambda (v_1 \ldots v_n) \text{call}) \\
call \in \text{Call} ::= (f \ e_1 \ldots e_n)
\]
No call returns

Callers pass callbacks

Still Turing-complete
Concrete state-space

\[ \zeta \in \Sigma = \text{Call} \times \text{Env} \]
Concrete state-space

\[ \varsigma \in \Sigma = \text{Call} \times \text{Env} \]

\[ \rho \in \text{Env} = \text{Var} \rightarrow \text{Clo} \]
Concrete state-space

\[ \varsigma \in \Sigma = \text{Call} \times \text{Env} \]

\[ \rho \in \text{Env} = \text{Var} \rightarrow \text{Clo} \]

\[ \text{clo} \in \text{Clo} = \text{Lam} \times \text{Env} \]
Concrete semantics

\((\Rightarrow) \subseteq \Sigma \times \Sigma\)
Concrete semantics

\[ \varepsilon : \text{Exp} \times \text{Env} \rightarrow \text{Clo} \]
\[ \mathcal{E}(\text{lam}, \rho) = (\text{lam}, \rho) \]
\[ \mathcal{E}(v, \rho) = \rho(v) \]
\((\llbracket (f \ e_1 \ldots e_n) \rrbracket, \rho) \Rightarrow (\text{call}, \rho''), \text{ where}\)
\((\llbracket (f\ e_1 \ldots e_n)\rrbracket, \rho) \Rightarrow (\text{call}, \rho'')\), where
\((\llbracket (\lambda (v_1 \ldots v_n) \text{ call})\rrbracket, \rho') = \mathcal{E}(f, \rho)\)
\(((\llbracket f \; e_1 \ldots e_n \rrbracket), \rho) \Rightarrow \text{(call, } \rho''\text{)}\), where
\((\llbracket (\lambda (v_1 \ldots v_n) \; \text{call}) \rrbracket), \rho') = \mathcal{E}(f, \rho)
\quad \text{clo}_i = \mathcal{E}(e_i, \rho)\)
\[
([[(f \; e_1 \ldots e_n)]], \rho) \Rightarrow (\text{call}, \rho'') , \text{ where } \\
([[(\lambda \; (v_1 \ldots v_n) \; \text{call})]], \rho') = \mathcal{E}(f, \rho) \\
clo_i = \mathcal{E}(e_i, \rho) \\
\rho'' = \rho'[v_i \mapsto clo_i]
\]
To analyze?
Make it finite!
Abstract state-space

\[ \varsigma \in \Sigma = \text{Call} \times \text{Env} \]

\[ \rho \in \text{Env} = \text{Var} \rightarrow \text{Clo} \]

\[ clo \in \text{Clo} = \text{Lam} \times \text{Env} \]
Abstract state-space

\[ \varsigma \in \Sigma = \text{Call} \times \text{Env} \]

\[ \rho \in \text{Env} = \text{Var} \rightarrow \text{Clo} \]

\[ clo \in \text{Clo} = \text{Lam} \times \text{Env} \]
Abstract state-space

\[ \zeta \in \Sigma = \text{Call} \times \text{Env} \]

\[ \rho \in \text{Env} = \text{Var} \rightarrow \text{Clo} \]

\[ \text{clo} \in \text{Clo} = \text{Lam} \times \text{Env} \]
Abstract state-space

\[ \zeta \in \Sigma = \text{Call} \times \text{Env} \]

\[ \rho \in \text{Env} = \text{Var} \rightarrow \text{Clo} \]

\[ clo \in \text{Clo} = \text{Lam} \times \text{Env} \]
Abstract state-space

\[ \zeta \in \Sigma = \text{Call} \times \text{Env} \]

\[ \rho \in \text{Env} = \text{Var} \rightarrow \text{Clo} \]

\[ clo \in \text{Clo} = \text{Lam} \times \text{Env} \]
Abstract state-space

$$\zeta \in \Sigma = \text{Call} \times \text{Env}$$

$$\rho \in \text{Env} = \text{Var} \rightarrow \text{Clo}$$

$$\text{clo} \in \text{Clo} = \text{Lam}$$
Abstract state-space

\[ \zeta \in \Sigma = \text{Call} \times \text{Env} \]
\[ \rho \in \text{Env} = \text{Var} \rightarrow \mathcal{P} \left( \text{Clo} \right) \]
\[ \text{clo} \in \text{Clo} = \text{Lam} \]
Abstract state-space

\[ \hat{\xi} \in \hat{\Sigma} = \text{Call} \times \hat{\text{Env}} \]
\[ \hat{\rho} \in \hat{\text{Env}} = \text{Var} \rightarrow \mathcal{P} (\hat{\text{Clo}}) \]
\[ \hat{\text{clo}} \in \hat{\text{Clo}} = \text{Lam} \]
\[ \alpha(lam, \rho) = lam \]
\( \alpha(call, \rho) = (call, \alpha(\rho)) \)
\[ \alpha(\rho) = \lambda v. \{ \alpha(\rho'(v)) : \rho' \text{ is reachable in } \rho \} \]
Abstract semantics

\[
(\sim \rightarrow) \subseteq \hat{\Sigma} \times \hat{\Sigma}
\]
Abstract semantics

\[ \mathcal{E} : \text{Exp} \times \text{Env} \rightarrow \text{Clo} \]
Abstract semantics

\[ \hat{\mathcal{E}} : \text{Exp} \times \hat{\text{Env}} \rightarrow \mathcal{P} (\hat{\text{Clo}}) \]
\[ \hat{E}(\text{lam}, \hat{\rho}) = (\text{lam}, \hat{\rho}) \]

\[ \hat{E}(v, \hat{\rho}) = \hat{\rho}(v) \]
\[ \hat{\mathcal{E}}(\text{lam}, \hat{\rho}) = \{ \text{lam} \} \]

\[ \hat{\mathcal{E}}(\nu, \hat{\rho}) = \hat{\rho}(\nu) \]
\(([(f \ e_1 \ldots e_n)], \hat{\rho}) \leadsto (\text{call}, \hat{\rho}'),\) \ where

\[C_i = \hat{\rho} \left[ v_i \right] \]
\[(\mathcal{E}(\lambda (v_1 \ldots v_n) \text{ call}) \in \hat{\mathcal{E}}(f, \hat{\rho}))\]
\[
(\llbracket (f\ e_1 \ldots e_n)\rrbracket, \hat{\rho}) \rightsquigarrow (\text{call}, \hat{\rho}'), \text{ where } \\
\llbracket (\lambda (v_1 \ldots v_n) \text{ call})\rrbracket \in \hat{\mathcal{E}}(f, \hat{\rho}) \\
\hat{C}_i = \hat{\mathcal{E}}(e_i, \hat{\rho})
\]
\[(\[(f \ e_1 \ldots \ e_n)\], \hat{\rho}) \sim (\text{call}, \hat{\rho}'), \text{ where} \]

\[
[(\lambda (v_1 \ldots v_n) \text{ call})] \in \hat{\mathcal{E}}(f, \hat{\rho})
\]

\[
\hat{C}_i = \hat{\mathcal{E}}(e_i, \hat{\rho})
\]

\[
\hat{\rho}' = \hat{\rho} \sqcup [v_i \mapsto \hat{C}_i]
\]
\( \llbracket (f \; e_1 \ldots e_n) \rrbracket, \hat{\rho} \sim (\text{call}, \hat{\rho}') \), where

\[
\llbracket (\lambda \; (v_1 \ldots v_n) \; \text{call}) \rrbracket \in \hat{\mathcal{E}}(f, \hat{\rho})
\]

\[
\hat{C}_i = \hat{\mathcal{E}}(e_i, \hat{\rho})
\]

\[
\hat{\rho}' = \hat{\rho} \sqcup \lfloor v_i \mapsto \hat{C}_i \rfloor
\]
\((\llbracket (f \ e_1 \ldots e_n) \rrbracket, \hat{\rho}) \rightsquigarrow (\text{call}, \hat{\rho}'), \text{ where}\)

\[
\llbracket (\lambda (v_1 \ldots v_n) \text{ call}) \rrbracket \in \hat{\mathcal{E}}(f, \hat{\rho})
\]

\[
\hat{C}_i = \hat{\mathcal{E}}(e_i, \hat{\rho})
\]

\[
\hat{\rho}' = \hat{\rho} \uplus [v_i \mapsto \hat{C}_i]
\]

\[
(\llbracket (f \ e_1 \ldots e_n) \rrbracket, \rho) \Rightarrow (\text{call}, \rho''), \text{ where}\]

\[
(\llbracket (\lambda (v_1 \ldots v_n) \text{ call}) \rrbracket, \rho') = \mathcal{E}(f, \rho)
\]

\[
clo_i = \mathcal{E}(e_i, \rho)
\]

\[
\rho'' = \rho'[v_i \mapsto clo_i]
\]
Soundness

\[ \begin{array}{ccc}
\zeta & \Rightarrow & \zeta' \\
\alpha & & \alpha \\
\square & & \square \\
\hat\zeta & \sim & \hat\zeta' \\
\end{array} \]

**Theorem:** If the concrete takes a step, then the abstract can take a matching step.
Running 0CFA
Running 0CFA
Running 0CFA

call ↦ (call, ⊥)
Running 0CFA

(call, ⊥) → \(\hat{\varsigma}\) → \(\hat{\varsigma}\) → \(\hat{\varsigma}\) → \(\hat{\varsigma}\) → \(\hat{\varsigma}\) → \(\hat{\varsigma}\) → \(\hat{\varsigma}\)
Order between states is preserved.

Monotonic growth not required.
How about the next level?
\[ f, \alpha \in AExp = \text{Var} + \text{Lam} \]

\[ e \in \text{Exp} ::= (\text{let } ((v \ call)) \ e') \]

\[ \mid \text{call} \]

\[ \mid \alpha \]

\[ \mid \alpha \]

\[ \text{call} \in \text{Call} ::= (f \ \alpha_1 \ldots \alpha_n) \]
\[ \Sigma = \text{Exp} \times \text{Env} \times \text{Store} \times \text{Kont} \]

\[ \text{Env} = \text{Var} \rightarrow \text{Addr} \]

\[ \text{Store} = \text{Addr} \rightarrow \text{Clo} \]

\[ \text{Kont} = \text{Var} \times \text{Exp} \times \text{Env} \times \text{Kont} + \{\text{halt}\} \]

\text{Addr} \text{ is an infinite set of addresses}
\[ \Sigma = \text{Exp} \times \text{Env} \times \text{Store} \times \text{Kont} \]

\[ \text{Env} = \text{Var} \rightarrow \text{Addr} \]

\[ \text{Store} = \text{Addr} \rightarrow \text{Clo} \]

\[ \text{Kont} = \text{Var} \times \text{Exp} \times \text{Env} \times \text{Kont} + \{\text{halt}\} \]

\(\text{Addr}\) is an infinite set of addresses
\[ \Sigma = \text{Exp} \times \text{Env} \times \text{Store} \times \text{Kont} \]

\[ \text{Env} = \text{Var} \rightarrow \text{Addr} \]

\[ \text{Store} = \text{Addr} \rightarrow \text{Clo} \]

\[ \text{Kont} = \text{Var} \times \text{Exp} \times \text{Env} \times \text{Kont} + \{\text{halt}\} \]

\text{Addr is an infinite set of addresses}
\[ \Sigma = \text{Exp} \times \text{Env} \times \text{Store} \times \text{Kont} \]

\[ \text{Env} = \text{Var} \rightarrow \text{Addr} \]

\[ \text{Store} = \text{Addr} \rightarrow \text{Clo} \]

\[ \text{Kont} = \text{Var} \times \text{Exp} \times \text{Env} \times \text{Kont} + \{\text{halt}\} \]

\text{Addr} \text{ is an infinite set of addresses}
\[ \Sigma = \text{Exp} \times \text{Env} \times \text{Store} \times \text{Kont} \]

\[ \text{Env} = \text{Var} \rightarrow \text{Addr} \]

\[ \text{Store} = \text{Addr} \rightarrow \text{Clo} \]

\[ \text{Kont} = \text{Var} \times \text{Exp} \times \text{Env} \times \text{Addr} + \{ \text{halt} \} \]

\text{Addr} \text{ is an infinite set of addresses}
\[ \Sigma = \text{Exp} \times \text{Env} \times \text{Store} \times \text{Kont} \]

\[ \text{Env} = \text{Var} \rightarrow \text{Addr} \]

\[ \text{Store} = \text{Addr} \rightarrow \text{Clo} + \text{Kont} \]

\[ \text{Kont} = \text{Var} \times \text{Exp} \times \text{Env} \times \text{Addr} + \{\text{halt}\} \]

\text{Addr} \text{ is an infinite set of addresses}
\[ \Sigma = \text{Exp} \times \text{Env} \times \text{Store} \times \text{Kont} \]

\[ \text{Env} = \text{Var} \rightarrow \text{Addr} \]

\[ \text{Store} = \text{Addr} \rightarrow \text{Clo} + \text{Kont} \]

\[ \text{Kont} = \text{Var} \times \text{Exp} \times \text{Env} \times \text{Addr} + \{\text{halt}\} \]

\text{Addr} \text{ is an finite set of addresses}
\[
\Sigma = \text{Exp} \times \text{Env} \times \text{Store} \times \text{Kont}
\]

\[
\text{Env} = \text{Var} \rightarrow \text{Addr}
\]

\[
\text{Store} = \text{Addr} \rightarrow \mathcal{P} (\text{Clo} + \text{Kont})
\]

\[
\text{Kont} = \text{Var} \times \text{Exp} \times \text{Env} \times \text{Addr} + \{\text{halt}\}
\]

\[
\text{Addr} \text{ is an finite set of addresses}
\]
And, other machines?
**CEK (F&F, 1986)**

\[ \xymatrix{ s \ar[r] & \text{CEK} \; s' } \]

| \langle x, \rho, \kappa \rangle | \langle v, \rho', \kappa \rangle \text{ where } \rho(x) = (v, \rho') |
| \langle (e_0e_1), \rho, \kappa \rangle | \langle e_0, \rho, \text{ar}(e_1, \rho, \kappa) \rangle |
| \langle v, \rho, \text{ar}(e, \rho', \kappa) \rangle | \langle e, \rho', \text{fn}(v, \rho, \kappa) \rangle |
| \langle v, \rho, \text{fn}(\lambda x.e), \rho', \kappa \rangle | \langle e, \rho'[x \mapsto (v, \rho)], \kappa \rangle |
Krivine (ICFP 2010)

\[
\begin{align*}
\varsigma & \in \Sigma = \text{Exp} \times \text{Env} \times \text{Store} \times \text{Kont} \\
\sigma & \in \text{Storable} ::= \mathbf{d}(e, \rho) \mid \mathbf{c}(v, \rho) \\
\kappa & \in \text{Kont} ::= \mathbf{mt} \mid \mathbf{c}_1(a, \kappa) \mid \mathbf{c}_2(a, \kappa)
\end{align*}
\]
CM (ICFP 2010)

\[ \varsigma \mapsto_{CM} \varsigma' \]

| \langle \text{fail}, \rho, \sigma, \kappa \rangle | \langle \text{fail}, \rho, \sigma, \text{mt}^0 \rangle |
| \langle \text{frame } R \ e \rangle, \rho, \sigma, \kappa \rangle | \langle e, \rho, \sigma, \kappa [R \mapsto \text{deny}] \rangle |
| \langle \text{grant } R \ e \rangle, \rho, \sigma, \kappa \rangle | \langle e, \rho, \sigma, \kappa [R \mapsto \text{grant}] \rangle |
| \langle \text{test } R \ e_0 \ e_1 \rangle, \rho, \sigma, \kappa \rangle | \begin{cases} \langle e_0, \rho, \sigma, \kappa \rangle & \text{if } \text{OK}(R, \kappa), \\ \langle e_1, \rho, \sigma, \kappa \rangle & \text{otherwise} \end{cases} |
| \text{OK}(\emptyset, \kappa) |
| \text{OK}(R, \text{mt}^m) | (R \cap m^{-1}(\text{deny}) = \emptyset) |
| \text{OK}(R, \text{fn}^m(v, \rho, \kappa)) | (R \cap m^{-1}(\text{deny}) = \emptyset) \land \text{OK}(R \setminus m^{-1}(\text{grant}), \kappa) |
| \text{OK}(R, \text{ar}^m(e, \rho, \kappa)) | \text{OK}(R, \text{mt}^m) |
Java (PLDI 2010)

\[ \varsigma \in \Sigma = \text{Stmt} \times \text{BEnv} \times \text{Store} \times \text{KontPtr} \times \text{Time} \]

\[ \beta \in \text{BEnv} = \text{Var} \rightarrow \text{Addr} \]

\[ \sigma \in \text{Store} = \text{Addr} \rightarrow \text{D} \]

\[ d \in \text{D} = \text{Val} \]

\[ \text{val} \in \text{Val} = \text{Obj} + \text{Kont} \]

\[ o \in \text{Obj} = \text{ClassName} \times \text{BEnv} \]

\[ \kappa \in \text{Kont} = \text{Var} \times \text{Stmt} \times \text{BEnv} \times \text{KontPtr} \]

\[ a \in \text{Addr} \text{ is a set of addresses} \]

\[ p^\kappa \in \text{KontPtr} \subseteq \text{Addr} \]

\[ t \in \text{Time} \text{ is a set of time-stamps.} \]
C/LLVM

\[ \varsigma \in \text{State} = \text{Eval} + \text{Apply} + \text{AppCont} + \text{AppFun} \]
\[ \text{Eval} = \text{STMT}^* \times \text{FrmPtr} \times \text{Conf} \times \text{StkPtr} \]
\[ \text{Apply} = \text{LHS}^* \times D^* \times \text{Eval} \]
\[ \text{AppFun} = \text{FUN} \times D^* \times \text{FrmPtr} \times \text{Conf} \times \text{StkPtr} \]
\[ \text{AppCont} = \text{Cont} \times D \times \text{Conf} \]

- \( d \in D = \text{Val} \)
- \( \text{val} \in \text{Val} = \text{Cont} + \text{FUN} + \text{Loc} + \text{Bas} \)
- \( \kappa \in \text{Cont} = \text{LHS} \times \text{STMT}^* \times \text{FrmPtr} \times \text{StkPtr} \)
- \( \text{bas} \in \text{Bas} = \) a set of basic values

- \( \text{loc} \in \text{Loc} = \text{Addr} + \text{StkPtr} + \text{Bind} \)
- \( a \in \text{Addr} = \) an infinite set of heap pointers
- \( sp \in \text{StkPtr} = \) an infinite set of stack pointers
- \( fp \in \text{FrmPtr} = \text{StkPtr} \)
- \( b \in \text{Bind} = \text{VAR} \times \text{FrmPtr} \)

- \( c \in \text{Conf} = \text{Store} \times \text{Succ} \times \text{Pred} \)
- \( \sigma \in \text{Store} = \text{Loc} \rightarrow D \)
- \( \sigma_+ \in \text{Succ} = \text{Loc} \rightarrow \text{Loc} \)
- \( \sigma_- \in \text{Pred} = \text{Loc} \rightarrow \text{Loc} \)
Up next
JavaScript

\[ \varsigma \in \Sigma = (\text{Stmt} + \text{Body}) \times \text{BEnv} \times \text{Store} \times \text{FPtr} \]

- \( \beta \in \text{BEnv} = \text{Var} \rightarrow \text{Addr} \)
- \( \sigma \in \text{Store} = \text{Addr} \rightarrow D \)
- \( d \in D = \text{Val} \)
- \( \text{val} \in \text{Val} = \text{Bas} + \text{Clo} + \text{Kont} + \text{Loc} \)
- \( \text{bas} \in \text{Bas} = \text{String} + \text{Num} + \text{Boolean} \)
- \( \text{clo} \in \text{Clo} = \text{Fun} \times \text{BEnv} \)
- \( \kappa \in \text{Kont} ::= \text{ret}(v, \beta, s, fp) \)
  \[ \mid \text{ex}(v, \beta, s, fp, s') \]
- \( a \in \text{Addr} = \text{Bind} + \text{Field} + \text{FPtr} \)
- \( b \in \text{Bind} = \text{Var} \times \text{Contour} \)
- \( \text{field} \in \text{Field} = \text{Loc} \times \text{String} \)
- \( fp \in \text{FPtr} = \text{Contour} \)

\( cn \in \text{Contour} \) is an **infinite** set of contours

\( loc \in \text{Loc} \) is an **infinite** set of locations

[states]

[binding environments]

[stores]

[denotable values]

[values]

[basic values]

[closures]

[return continuations]

[exceptional continuations]

[addresses]

[bindings]

[object fields]

[frame pointers]
Bonus: Compositionality
Direct products
Direct products
Direct products
Direct products
Application: Array-bounds checks
Logic-flow analysis
Logic-flow analysis
Logic-flow analysis

\[ i < \text{length}(a) \]
Logic-flow analysis

\[ i < \text{length}(a) \]
What about sub-Turing domain-specific languages?
Regex
Yacc
Datalog
SQL
Avoid halting problem.
RFC 2616 (HTTP 1.1)
2.1 Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [9]. Implementors will need to be familiar with the notation in order to understand this specification. The augmented BNF includes the following constructs:
LWS = [CRLF] 1*( SP | HT )

separators = "(" | ")" | "<" | ">" | ">" | "@" |
| "," | ";" | ":" | "\" | ">"
| "/" | "[" | "]" | "?" | "=" |
| "{" | "}" | SP | HT

http_URL = "http: "//" host [ ":" port ] [ abs_path [ "?" query ]]

Chunked-Body = *chunk
last-chunk
trailer
CRLF

chunk = chunk-size [ chunk-extension ] CRLF
chunk-data CRLF

chunk-size = 1*HEX
last-chunk = 1*"0" [ chunk-extension ] CRLF

chunk-extension= *( ";" chunk-ext-name [ "=" chunk-ext-val ] )

chunk-ext-name = token
chunk-ext-val = token | quoted-string
chunk-data = chunk-size(OCTET)
trailer = *(entity-header CRLF)

HTTP-Date = rfc1123-date | rfc850-date | asctime-date
rfc1123-date = wkday "," SP date1 SP time SP "GMT"
rfc850-date = weekday "," SP date2 SP time SP "GMT"
asctime-date = wkday SP date3 SP time SP 4DIGIT
date1 = 2DIGIT SP month SP 4DIGIT
| day month year (e.g., 02 Jun 1982)
date2 = 2DIGIT "-" month "-" 2DIGIT
| day-month-year (e.g., 02-Jun-82)
date3 = month SP ( 2DIGIT | ( SP 1DIGIT ) )
| month day (e.g., Jun 2)
time = 2DIGIT ":" 2DIGIT ":" 2DIGIT
| 00:00:00 - 23:59:59
wkday = "Mon" | "Tue" | "Wed"
| "Thu" | "Fri" | "Sat" | "Sun"
weekday = "Monday" | "Tuesday" | "Wednesday"
| "Thursday" | "Friday" | "Saturday" | "Sunday"
month = "Jan" | "Feb" | "Mar" | "Apr"
| "May" | "Jun" | "Jul" | "Aug"
| "Sep" | "Oct" | "Nov" | "Dec"
RFC 3501 (IMAPv4)
RFC 2812 (IRC)
The Augmented BNF representation for this is:

```
message    =  [ "\:\" prefix SPACE ] command [ params ] crlf
prefix     =  servername / ( nickname [ "!" user ] "@" host )
command    =  1*letter / 3digit
params     =  *14( SPACE middle ) [ SPACE ":\" trailing ]
               / 14( SPACE middle ) [ SPACE [ ":\" ] trailing ]
nospclclfcl =  %x01-09 / %x0B-0C / %x0E-1F / %x21-39 / %x3B-FF
               ; any octet except NUL, CR, LF, " " and "\:\"
middle     =  nospclclfcl *( ":\" / nospclclfcl )
trailing   =  *( ":\" / "\" / nospclclfcl )
SPACE      =  %x20        ; space character
crlf       =  %x0D %x0A   ; "carriage return "linefeed"

message    =  target
msgtarget  =  msgto *( ",\" msgto )
msgto      =  channel / ( user [ "\%" host ] "@" servername )
               / ( user "\%" host ) / targetmask
msgto      =  nickname / ( nickname ":\" user "@" host )
channel    =  ( "}\" / "+\" / ( ":\" channelid ) / ":\" ) chanstring
               [ ":\" chanstring ]
servername =  hostname
host       =  hostname / hostaddr
hostname   =  shortname *( ".\" shortname )
shortname  =  ( letter / digit ) *( letter / digit / ":\" )
               *( letter / digit )
               ; as specified in RFC 1123 [HNAME]
hostaddr   =  ip4addr / ip6addr
ip4addr    =  1*3digit ":\" 1*3digit ":\" 1*3digit ":\" 1*3digit
ip6addr    =  1*hexdigit 7( ":\" 1*hexdigit )
ip6addr    =  / "0:0:0:0:0:0:0:0:0:0\" ( "0\" / "FFFF\" ) ":\" ip4addr
nickname   =  ( letter / special ) *8( letter / digit / special / ":\" )
targetmask =  ( "$\" / ":\" ) mask
               ; see details on allowed masks in section 3.3.1
chanstring =  %x01-07 / %x0B-09 / %x0B-0C / %x0E-1F / %x21-2B
chanstring =  / %x2D-39 / %x3B-FF
               ; any octet except NUL, BELL, CR, LF, "\", ",\", and "\:\"
channelid  =  5( %x41-5A / digit ) 5( A-Z / 0-9 )
user       =  1* ( %x01-09 / %x0B-0C / %x0E-1F / %x21-3F / %x41-FF )
               ; any octet except NUL, CR, LF, ":\" and "\:\"
key        =  1*23( %x01-05 / %x07-09 / %x0C / %x0E-1F / %x21-7F )
               ; any 7-bit US ASCII character,
               ; except NUL, CR, LF, FF, h/v TABs, and ":\"
letter     =  %x41-5A / %x61-7A ; A-Z / a-z
digit      =  %x30-39 ; 0-9
hexdigit   =  digit / "A" / "B" / "C" / "D" / "E" / "F"
special    =  %x5B-60 / %x7B-7D
               ; [",",",",",",",",",",",",","","]"]
```
Efficient parsing techniques exist.
LALR(k)  Earley
LL(k)      Operator precedence
GLR        CYK
LR(k)       Combinators
SLR
PEG
packrat
Parsing tools abound.
State of the art?
*buf++
Apache
2,179 lines of C
lighttpd
1,211 lines of C
freenode IRCD
> 2000 lines of C
Courier IMAP
2,633 lines of C
Result?
IRCnet IRCD Buffer Overflow

---

**Synopsis:**
Apache 1.3
Version: 1.3.37 (latest version)

**Solution**
Vendor Patch

**Product**
Apache htpasswd util

**Software**
IRCnet IRCD 2.x

**Where**
Local system

---

**Imposed:**
A buffer overflow vulnerability has been found, it is dangerous only on environment where the binary is suid root.

---

**Target:**

**Details:**
Incorrect validation on the size of user input allows to copy a string, via strcpy, to a fixed size buffer.

File: htpasswd.c, Line 421.

---

**Keyword:**
FixedInTrunk, PatchAvailable

---

**Vulnerable systems:**
- mIRC version 6.1 and prior

**Immune systems:**
- mIRC version 6.11

When mIRC is installed, it registers its own handler for URL of the type "irc". Calling "irc://irc.hackme.com" from our web browser causes mIRC irc.hackme.com server. By inputting an overly long string to the "irc" protocon instruction pointer, thus controls the program's execution.

**Example:**
irc://[buffer]...... where's buffer >998 bytes

An attacker would be able to gain access to the target system if he was able to gain code execution under the current user's privilege.

---

**Release date:**
16 Oct 2003

**Last Change:**
20 Oct 2003

---

**Secunia ID:**
SA9999

---

**CVE-ID:**
CVE-2003-0864

---

**Date:**
2 Jan 2007
Why!??
Yacc blocks on read().
Yacc needs continuations.
The continuation of a parser is its derivative.
For more, google:
“Yacc is Dead”
The future is...
The future is...

...safe, correct
The future is...
...safe, correct
...domain-specific
The future is...
...safe, correct
...domain-specific
...deep analysis
Thanks!

- POPL 2006: Analysis of environments & stacks
- ICFP 2006: Abstract garbage collection
- PLDI 2006: Enabling coroutine fusion
- POPL 2007: Logic-flow analysis (for arrays)
- PLDI 2010: Featherweight Java analysis
- ICFP 2010: Deriving small-step analyzers
- SFP 2010: Pushdown small-step analysis
- POPL 2011: Small-step analysis on the GPU
Application: Dependence analysis
Dependence analysis
Dependence analysis
Dependence analysis
Dependence analysis

What resources are written?

What resources are read?

Which calling contexts are live on stack?
Context-sensitive dependence graphs
Context-sensitive dependence graphs

Resources → $v$ bound in $k$ → $f$ called in $k'$ → Calls
f()
g()
f() \parallel g()
Advanced technique: Abstract garbage collection
Abstract objects can die too.
Effects of abstract GC
Effects of abstract GC
Effects of abstract GC
Vicious cycle

Merging ($\cup$)

Forking ($\in$)
Vicious cycle

Merging ($\sqcup$)

Forking ($\in$)
Virtuous cycle

Un-merging

No forking
Orders of magnitude