One Minute Madness

PLMW 2014

Alexander Bakst

Data Structure Verification via Refinement Types

```
function append(x1, x2) {
  if (x1 != null){
      var n = x1.next;
      x1.next = append(n, x2);
      return x1;
  } else {
      return x2;
  }
}
```

Alexander Bakst
University of California, San Diego

Andrew Bedford

Non-interference

1. What is non-interference?
2. Our approach
   a. Flow sensitive
   b. Uses both static and dynamic analysis
      i. Less false positives
      ii. Less overhead

```
send publicValue to publicChannel;
recv_name file from user;
recv_content line from file;
send line to internet
```

3. Current work
   a. Concurrent programs
   b. Termination sensitivity

4. Future work
   a. Declassification
   b. Quantification
   c. Java implementation

```
if privateValue > 0 then
  while true do skip end
end;
send publicValue to publicChannel
```

Example where private information influences the termination of a program.

Andrew Bedford

UNIVERSITÉ LAVAL

Benjamin Greenman

Conditional Inheritance

```
class List<T> extends Eq<List<T>> given T extends Eq<T>
```

[0]Ben Greenman, Cornell University

Cole Schlesinger

tomorrow at

3:15pm

come see

CAROLYN ANDERSON

present

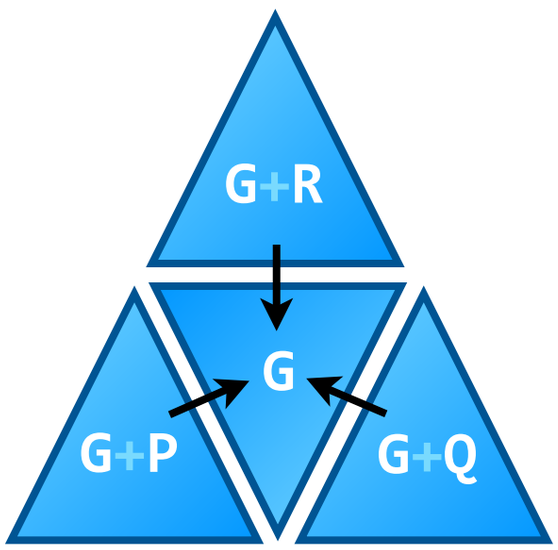
NetKAT

an algebraic presentation of network packet processing and verification

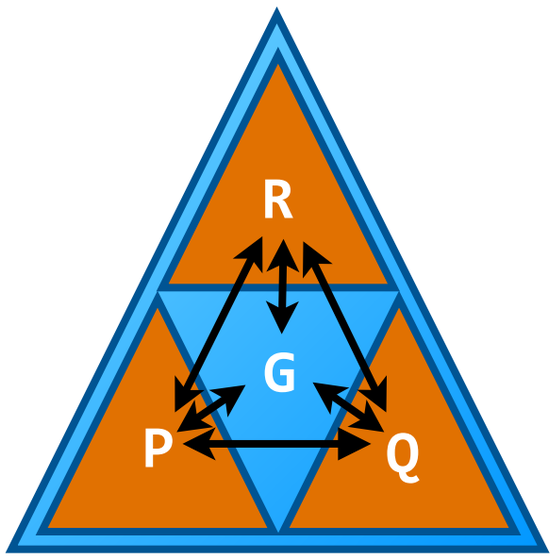
Cyrus Omar

Type-Oriented Foundations for
Safely Extensible Programming Systems

Cyrus Omar, CMU



(a) Separate Languages



(b) Extensible Language

client compatibility

G, P, Q, R
features
(syntax, type system, implementation strategy, editor service)

language

library

Denis Bogdanas

Executable Java semantics in $\mathbb{K}$ Framework

Things done

- Complete Java 1.4
- Underlying formalism:
  - $\mathbb{K}$ Framework

The problem

- Hard to explain
- Hard to use

The solution

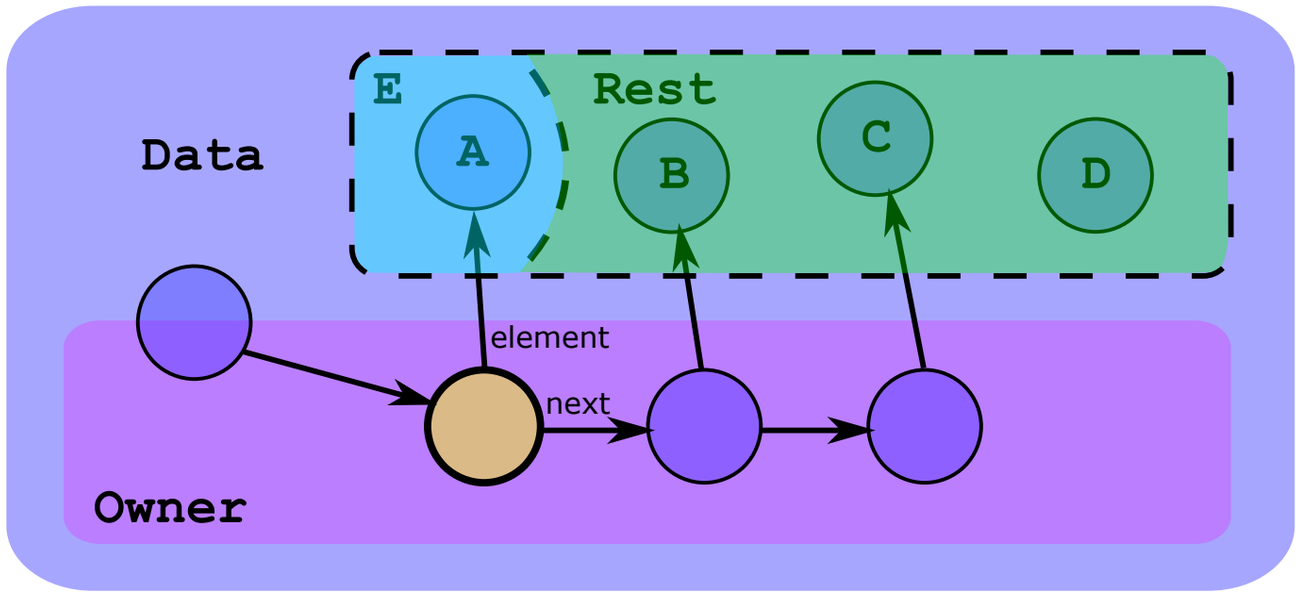
- Split the semantics into:
  - The preprocessing semantics
  - The execution semantics
- Valid java programs as preprocessing result

Diego Gomez Ajhuacho

Elias Castegren

Race-free Parallelism using Refined Ownership Types with Effects

elias.castegren@it.uu.se




```
class Link[Data = E + Rest]{
  E:Object element;
  Owner:Link[Rest] next;

  void map(Function f) writes Data{
    par{
      f.call(element); // writes E
      next.map(f);     // writes Rest
    }
  }
}
```

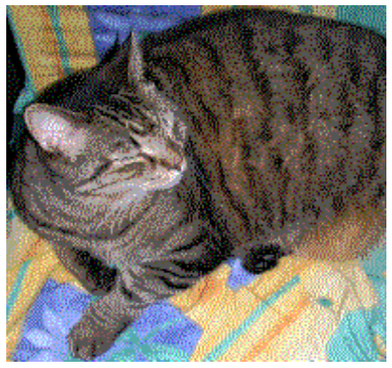
Elizabeth Davis

Modeling Steganography with Linear Epistemic Logic

Elizabeth Davis
Advisor: Frank Pfenning

Innocuous cover message

Bit ops + normalization



Embedded steganographic message

Epistemic logic : reason about information gained from extracting encoded message

Linear logic : reason about consumption and generation of resources in changing state

Linear Epistemic logic : reason about actions based on changing information state
(DeYoung & Pfenning, 2009)

$\langle K \rangle A$ – ***K*** *says* ***A*** : linear affirmation

$[K]A$ – ***K*** *has* ***A*** : linear knowledge

$[[K]]A$ – ***K*** *knows* ***A*** : persistent knowledge

Carnegie Mellon University

Eric Mullen

Formally Verifying a Superoptimizer

Eric Mullen

Eric Seidel

Refinement Types with LiquidHaskell

```
data Text = Text
  { arr :: Array
  , off :: {v:Nat | v           <= alen arr}
  , len :: {v:Nat | v + off <= alen arr}
  }

type Nat = {v:Int | v >= 0}

measure alen :: Array -> Nat
```

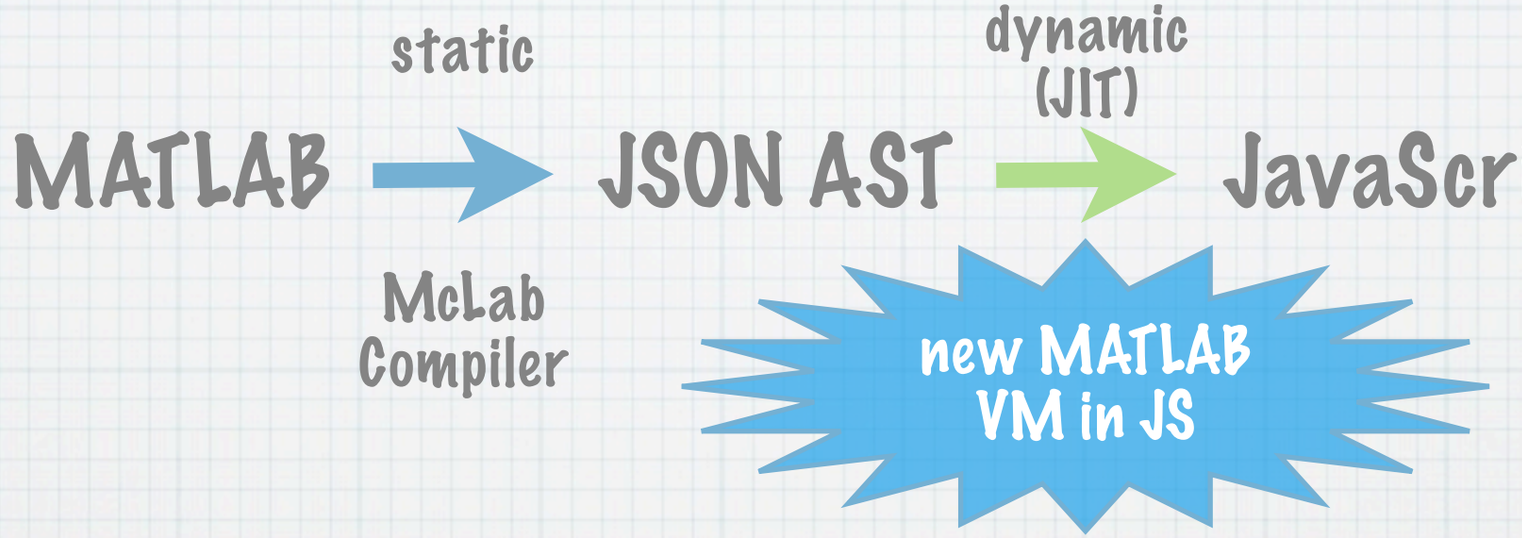
Eric Seidel -- UC San Diego

Erick Lavoie

Erick Lavoie, PhD @ McGill University

Why? Allow scientists and engineers to run MATLAB code in the browser, fast!

How?



Impact?

- Find the fastest JS subset for num. computation
- Design techniques for dynamic compilation to JS
- Characterize the performance behaviour of complex VMs
- Understand optimization across multiple VM layers

Ethel Bardsley

- Verify kernels
- Atomics make this harder
- Refined abstraction

Evgeny Roubinchtein

(Re)Implementing MetaOCaml: a multi-stage programming system

Multi-stage programming?

- A generalization of RE.compile()
- A (typed) eval()
- A partial evaluation you can repeat again (and again, and again. . .)

Is this stuff new?

- Native compiler has been done for OCaml 3
- Bytecode compiler has been done for OCaml 4
- (As far as I know) native compiler hasn't yet been done for OCaml 4

Me

Evgeny Roubinchtein, University of British Columbia, advised by Ron Garcia

Ezgi Cicek

A Type System for Incremental Computational Complexity

Ezgi Cicek, MPI-SWS

Fabian Muehlboeck

Fabian Muehlboeck

Gradual Typing for OO languages

```
List<T> SnocS<T>(IEnumerable<T> startS, T endS) {
     var elemsS = startS.ToList();
     elemsS.Add(endS);
     return elemsS;
}

dynamic SnocD(dynamic startD, dynamic endS) {
     var elemsD = startD.ToList();
     elemsD.Add(endD);
     return elemsD;
}
```

Felipe Banados

Gradual Type-and-Effect Systems

Felipe Bañados – University of Chile

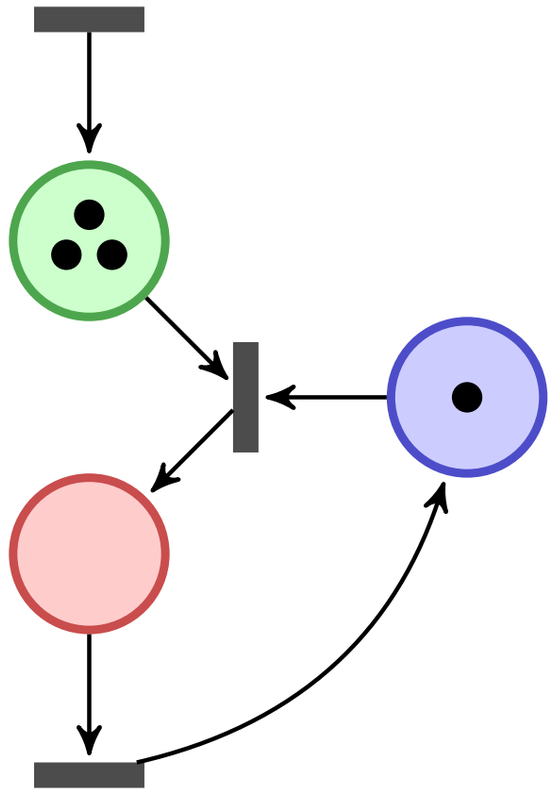
Filip Niksic

FILIP NIKSIC

MPI-SWS

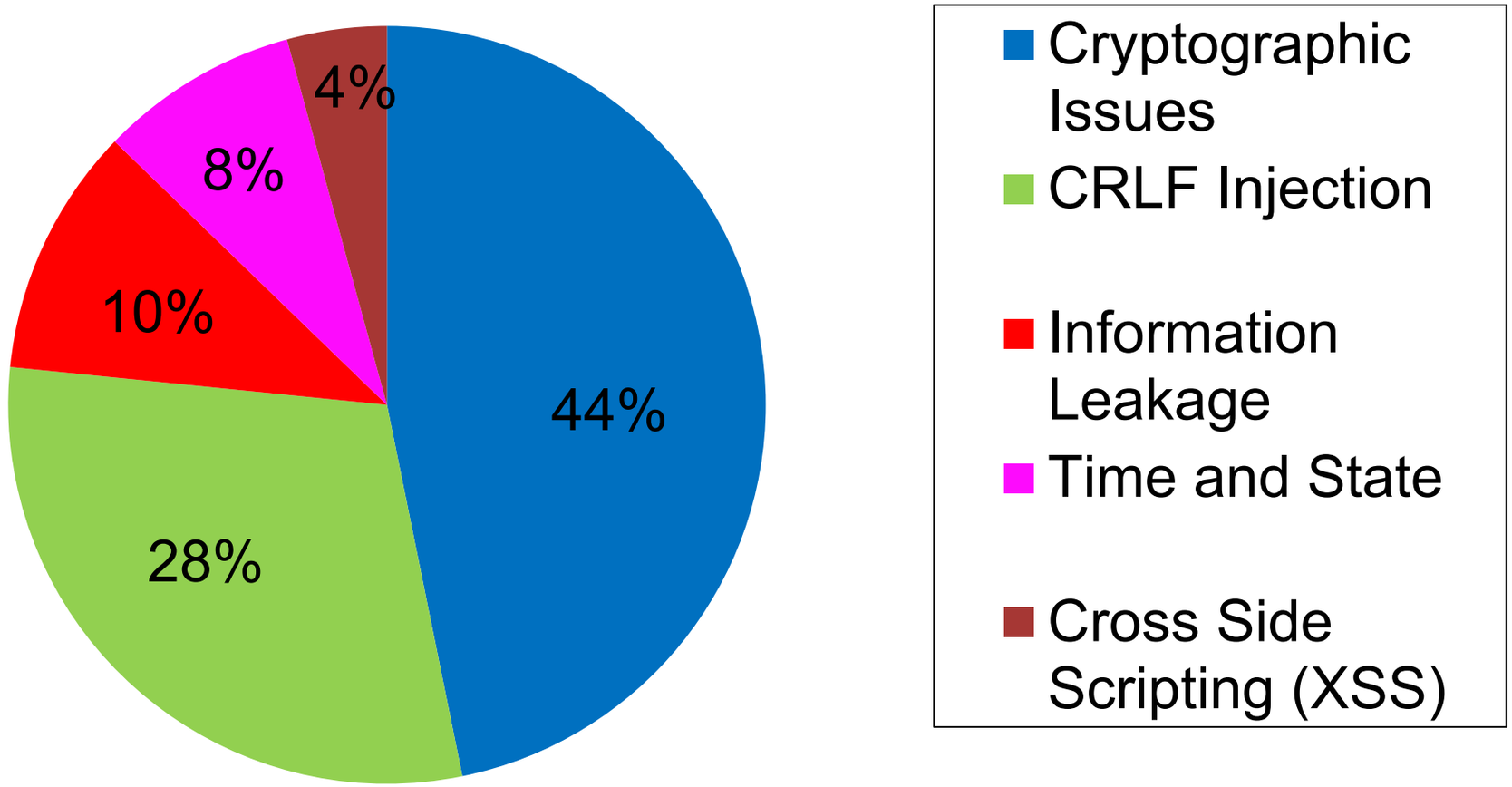
```
void thread() {
    // Non-critical section

    synchronized(lock)
    {
        // Critical section
    }
}
```



Hannah Gommerstadt

Vulnerability Distribution on the Android (~10,000 analyzed applications)



Heather Miller

Language and compiler support for DISTRIBUTED PROGRAMMING for Scala

Generally,

TYPE-SYSTEMS
GENERIC PROGRAMMING

Concretely,

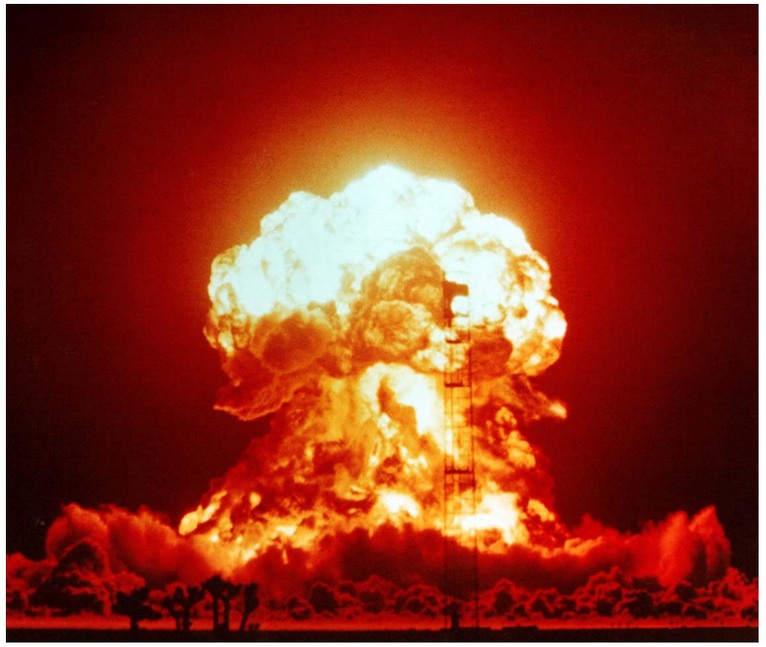
DISTRIBUTABLE FUNCTIONS
EFFICIENT & EXTENSIBLE SERIALIZATION
CONCURRENT/DATAFLOW ABSTRACTIONS

Heather Miller
heather.miller@epfl.ch
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

James Wilcox

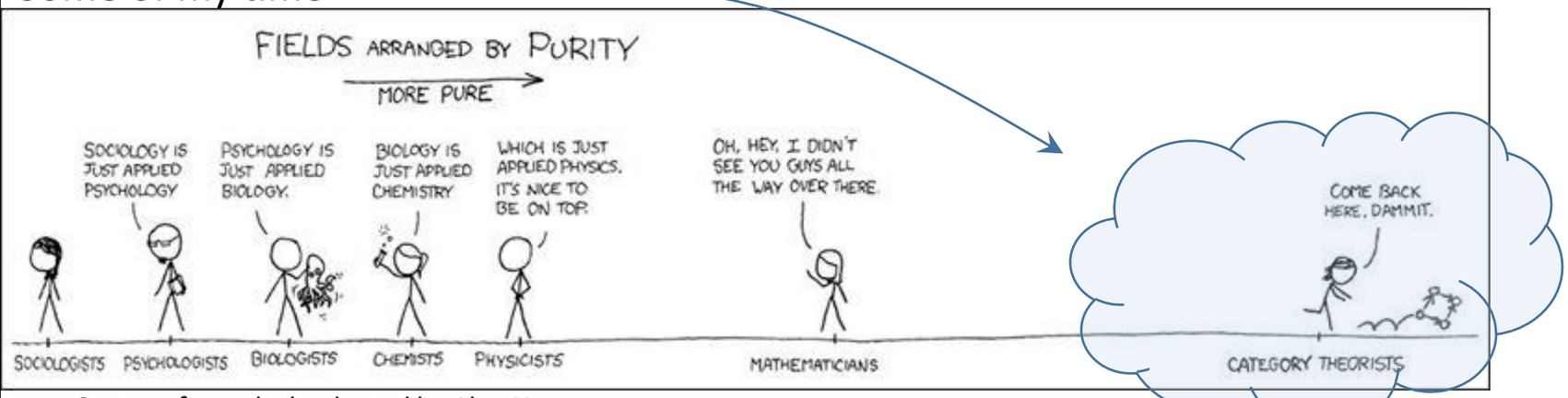
James Wilcox

University of Washington



Jason Gross

Some of my time



Cartoon from xkcd, adapted by Alan Huang

The rest of my time

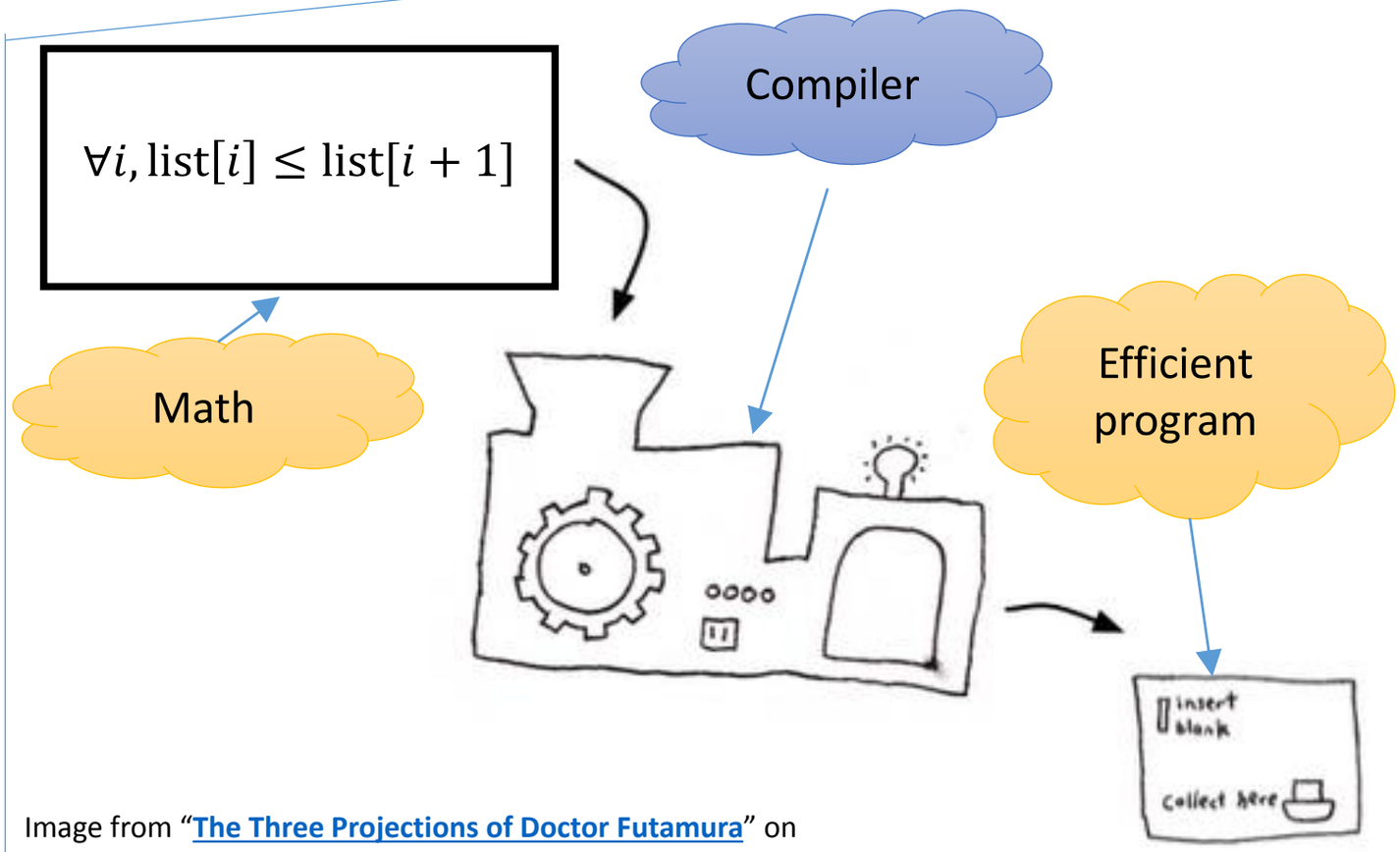
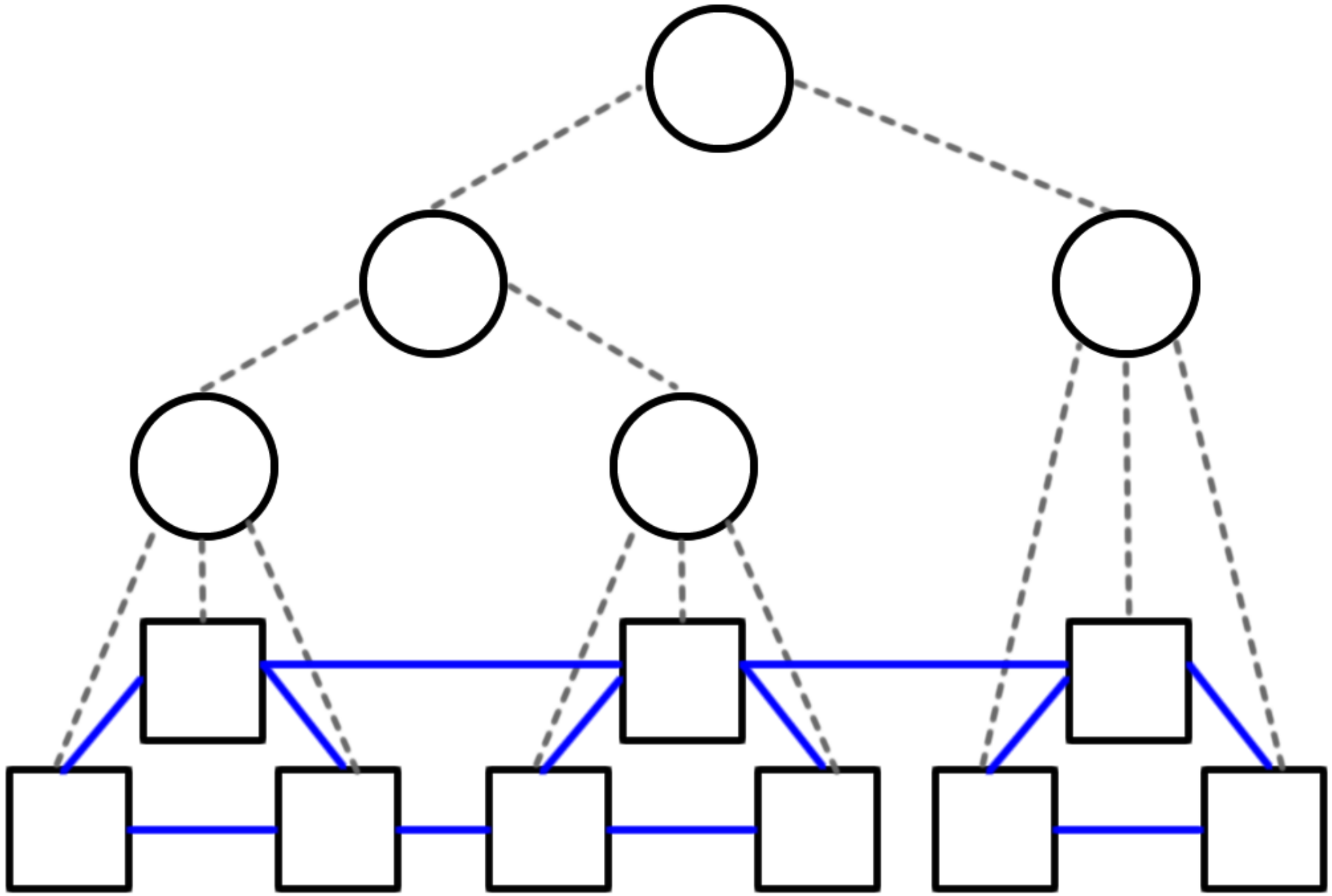


Image from "[The Three Projections of Doctor Futamura]" on [A Neighborhood of Infinity]

Jonathan Frankle



Laure Thompson

Using Epistemic Temporal Logic for Dynamic Security Policies

- **Security**: An observing agent learns no more than the combination of their previous knowledge and the knowledge allowed by the current policy
- **Knowledge**: Is formed from the agent's trace of past *remembered* observations
- **Policy**: The set of initial values an agent may learn
- **Connection:** Knowledge ≡ set of initial states that the agent *knows* to be possible
  - This possibilistic view corresponds to epistemic logic:

    $L\varphi$:= In *some* possible world an agent knows φ

Luke Maurer

- Common CBV and CBN π-calculus encodings are closely related to CPS.

Term $(\lambda x.\, xx)(\lambda y.\, y)$

CBV π $\boldsymbol{\nu} k\,(\boldsymbol{\nu} f\,(\overline{k}\langle f\rangle \mid !f(x,h).\,\overline{x}\langle x,h\rangle) \mid k(v).$
$\boldsymbol{\nu} k\,(\boldsymbol{\nu} g\,(\overline{k}\langle g\rangle \mid !g(y,h).\,\overline{h}\langle y\rangle) \mid k(w).\,\overline{v}\langle w,k_0\rangle))$

CBV CPS $(\lambda k.\, k(\lambda(x,h).\, x(x,h)))\,(\lambda v.$
$(\lambda k.\, k(\lambda(y,h).\, hy))(\lambda w.\, v(w,k_0)))$

- Have π-calculus encoding for call-by-need:

CBNd π $\boldsymbol{\nu} k\,(\boldsymbol{\nu} f\,(\overline{k}\langle f\rangle \mid !f(x,h).\,\boldsymbol{\nu} k'\,(\overline{x}\langle k'\rangle \mid k'(v).\,\overline{v}\langle x,h\rangle)) \mid k(v).$
$\boldsymbol{\nu} x\,(\overline{v}\langle x,k_0\rangle \mid x(k').\,\boldsymbol{\nu} k\,(\boldsymbol{\nu} g\,(\overline{k}\langle g\rangle \mid !g(y,h).\,\overline{y}\langle h\rangle) \mid$
$k(w).\,(!x(k).\,\overline{k}\langle w\rangle \mid \overline{k'}\langle w\rangle))))$

- We introduce a corresponding CPS transform:

CBNd CPS $(\lambda k.\, k(\lambda(x,h).\, x(\lambda_1 v.\, v(x,h))))\,(\lambda_1 v.$
$\nu x.\, x :=_1 (\lambda_1 k'.\,(\lambda k.\, k(\lambda(y,h).\, yh))(\lambda_1 w.\, x := \lambda k.\, kw\ \mathbf{in}\ k'w))$
$\mathbf{in}\ v(x,k_0))$

Mario Alvarez

Mario Alvarez (Princeton) – Senior thesis

- VST (Appel et al, Princeton)
- MirrorShard (G. Malecha, Harvard)
  - Reflective solver for separation-logic entailments
  - Can solve entailments we currently can't automatically
- Goal: improve automation in VST
  - Make system easier/practical for more applications, ultimately
  - Learn about integrating external solvers with VST (eventually, maybe SMT or others)

Matt Le

Deterministic Parallel Programming

- Nondeterminism makes parallel programming difficult
- Purely functional parallel languages are inherently deterministic
- Unfortunately, mutable state can improve efficiency
- Add restricted forms of mutable state
  - Extend runtime system to preserve determinism.
  - Formalize semantics of the language with these new features and prove determinism.

[1] `http://www.cs.rit.edu/~mtf/manticore/index.html`

Matthew Loring

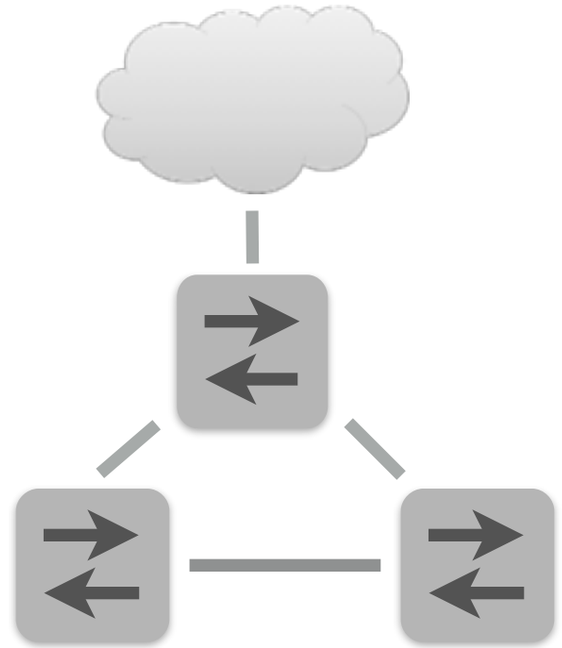
Security-Typed Programming Languages

- Goal: Develop languages that help programmers reason about the security of their programs
- Jif extends Java with direct support from the type system for programming with the Decentralized Label Model
  - Every type is annotated with labels specifying integrity and confidentiality policies

Matthew Milano

Network Verification

Matthew Milano
Cornell University



Goal: Verify formal properties of network configurations automatically

Idea: Encode NetKAT semantics using an SMT solver, and automatically answer queries about path properties

Challenge: Obvious direct encodings generate infinite models, which aren't handled well by current solvers

f ::= switch | inport ...
v ::= 0 | 1 | 2 | ...
a,b,c ::= true
| false
| f = v
| a_1 and a_2
| a_1 or a_2
| not a
p,q,r ::= filter a
| f := v
| p_1 | p_2
| p_1 ; p_2
| p^*
| dup

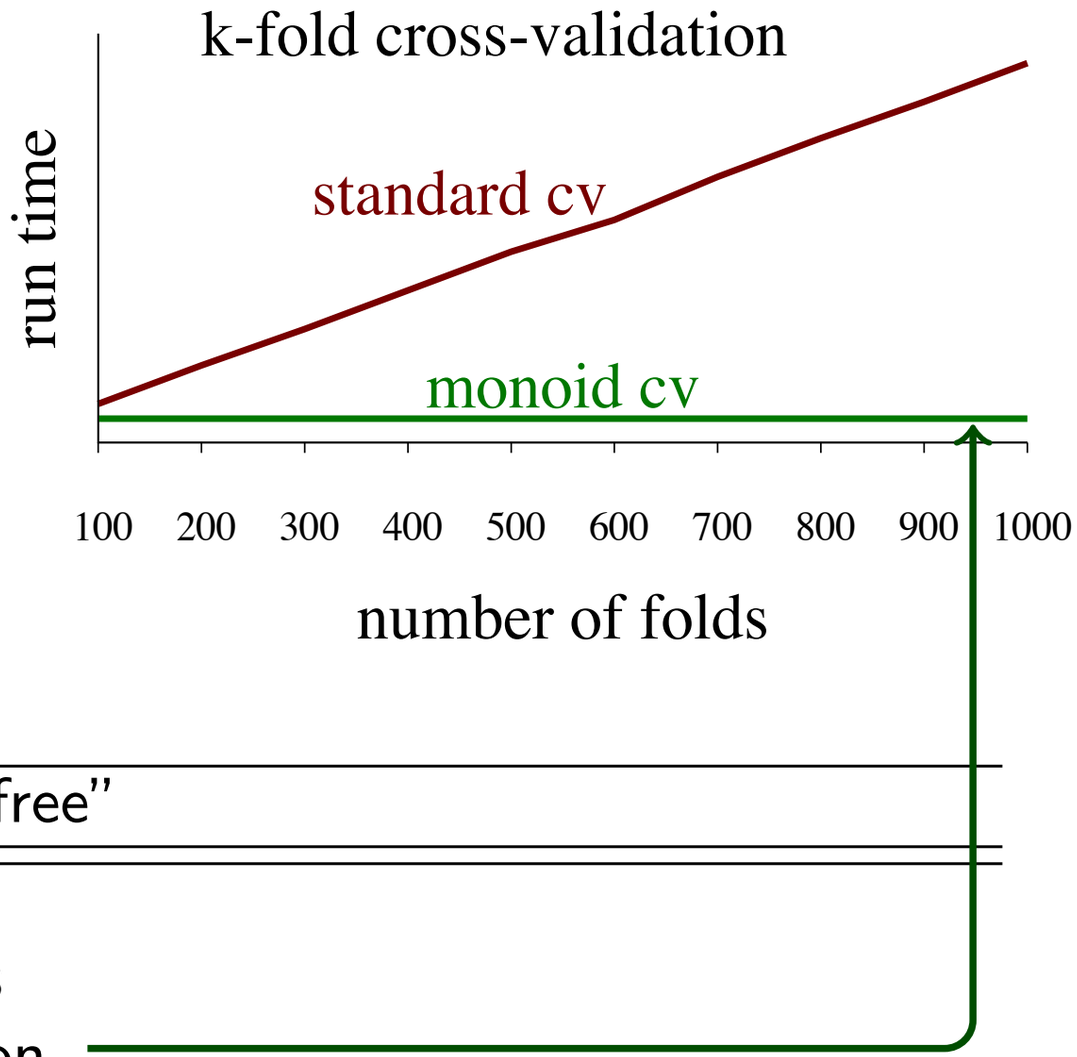
Mike Izbicki

mikeizbicki / **HLearn** Unwatch 27 Star 184 Fork 19

`http://github.com/mikeizbicki/hlearn`

Features

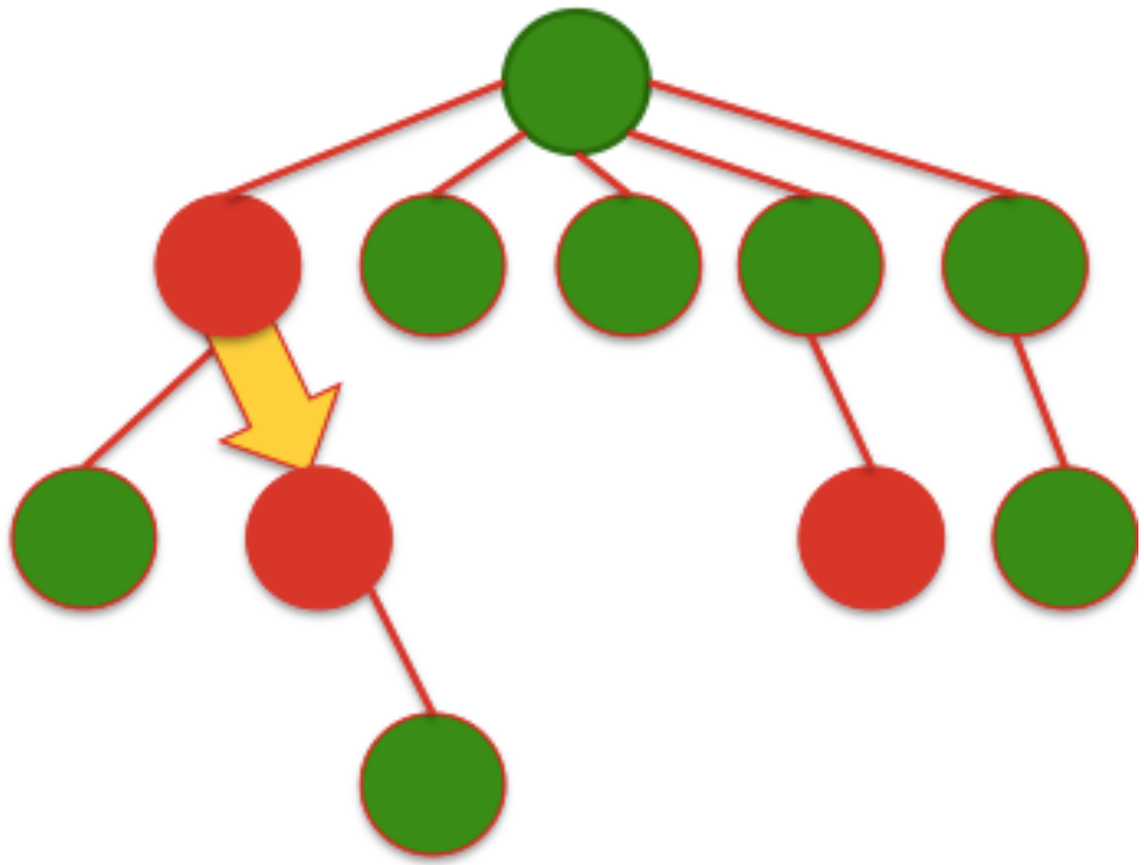
- purely functional (Haskell)
- very fast
- parameters at the type level
- design based on abstract algebra



algebraic structure	what we get "for free"
monoid	online algorithms
monoid	parallel algorithms
monoid	fast cross-validation
abelian group	subtract data points
abelian group	more fast cross-validation
R-module	data points weighted by the ring R
functor	(simple) transfer learning via fast data pre-processing
monad	(complex) transfer learning via fast pre-processing

Mike Shah

Java Critical Sections



Critical Section =

Figure 1. Call Graph

Michael.Shah@tufts.edu

Mohsen Lesani

Nathalie Oostvogels

Type Systems

for JavaScript web applications

for distributed web applications

Nathalie Oostvogels - Vrije Universiteit Brussel (Belgium)

Nicholas Braga

Target Testing:

Automated test case discovery technique using inferred models

Nicholas Braga Yuriy Brun

University of Massachusetts Amherst

Log:

```
1  74.15.155.103 [06/Jan/2011:07:24:13] "GET HTTP/1.1 /check-out.php"
2  13.15.232.201 [06/Jan/2011:07:24:19] "GET HTTP/1.1 /check-out.php"
3  13.15.232.201 [06/Jan/2011:07:25:33] "GET HTTP/1.1 /invalid-coupon.php"
4  74.15.155.103 [06/Jan/2011:07:27:05] "GET HTTP/1.1 /valid-coupon.php"
5  74.15.155.199 [06/Jan/2011:07:28:43] "GET HTTP/1.1 /check-out.php"
6  74.15.155.103 [06/Jan/2011:07:28:14] "GET HTTP/1.1 /reduce-price.php"
7  74.15.155.199 [06/Jan/2011:07:29:02] "GET HTTP/1.1 /get-credit-card.php"
8  13.15.232.201 [06/Jan/2011:07:30:22] "GET HTTP/1.1 /reduce-price.php"
9  74.15.155.103 [06/Jan/2011:07:30:55] "GET HTTP/1.1 /check-out.php"
10 13.15.232.201 [06/Jan/2011:07:31:17] "GET HTTP/1.1 /check-out.php"
11 13.15.232.201 [06/Jan/2011:07:31:20] "GET HTTP/1.1 /get-credit-card.php"
12 74.15.155.103 [06/Jan/2011:07:31:44] "GET HTTP/1.1 /get-credit-card.php"
```

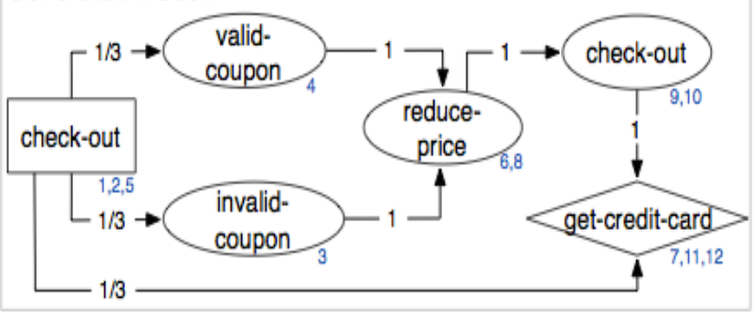
Regular Expressions:

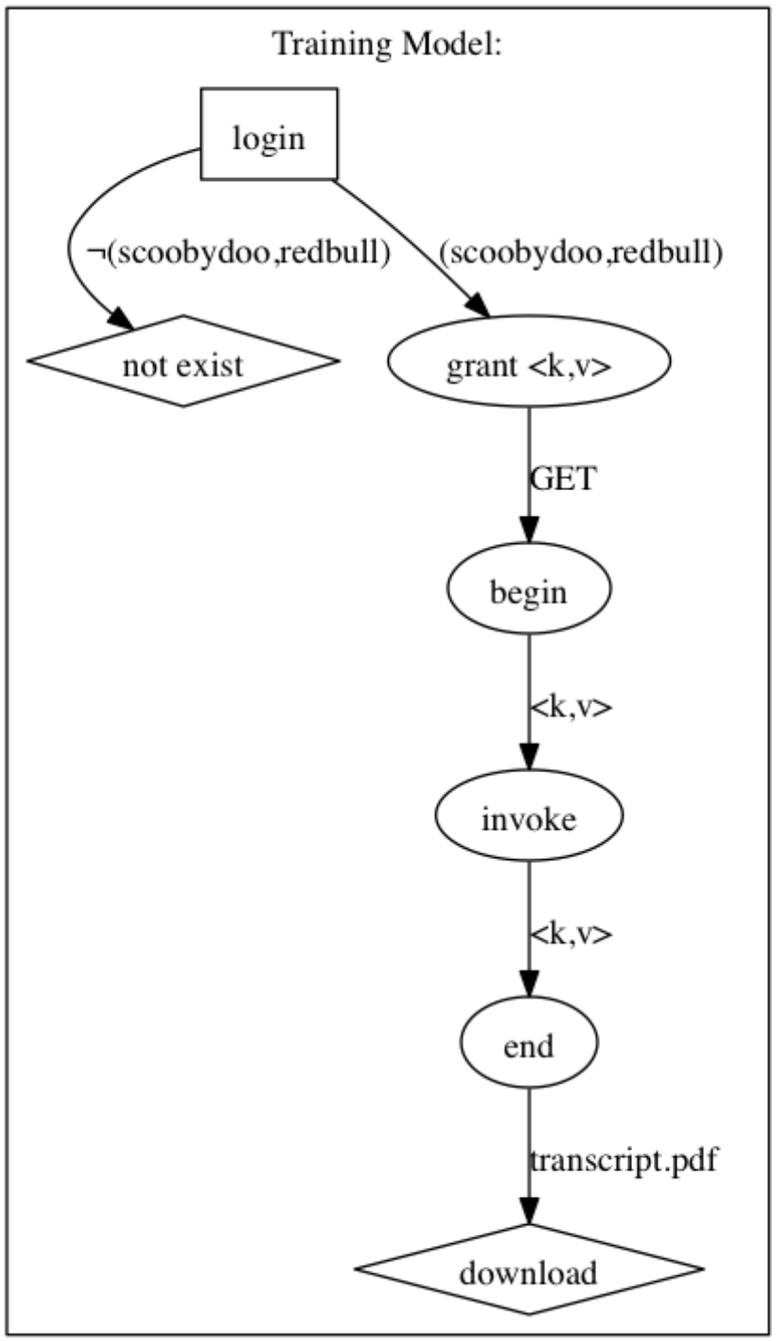
Line parsing: (?<ip>) .+ /(?<TYPE>.+).php"

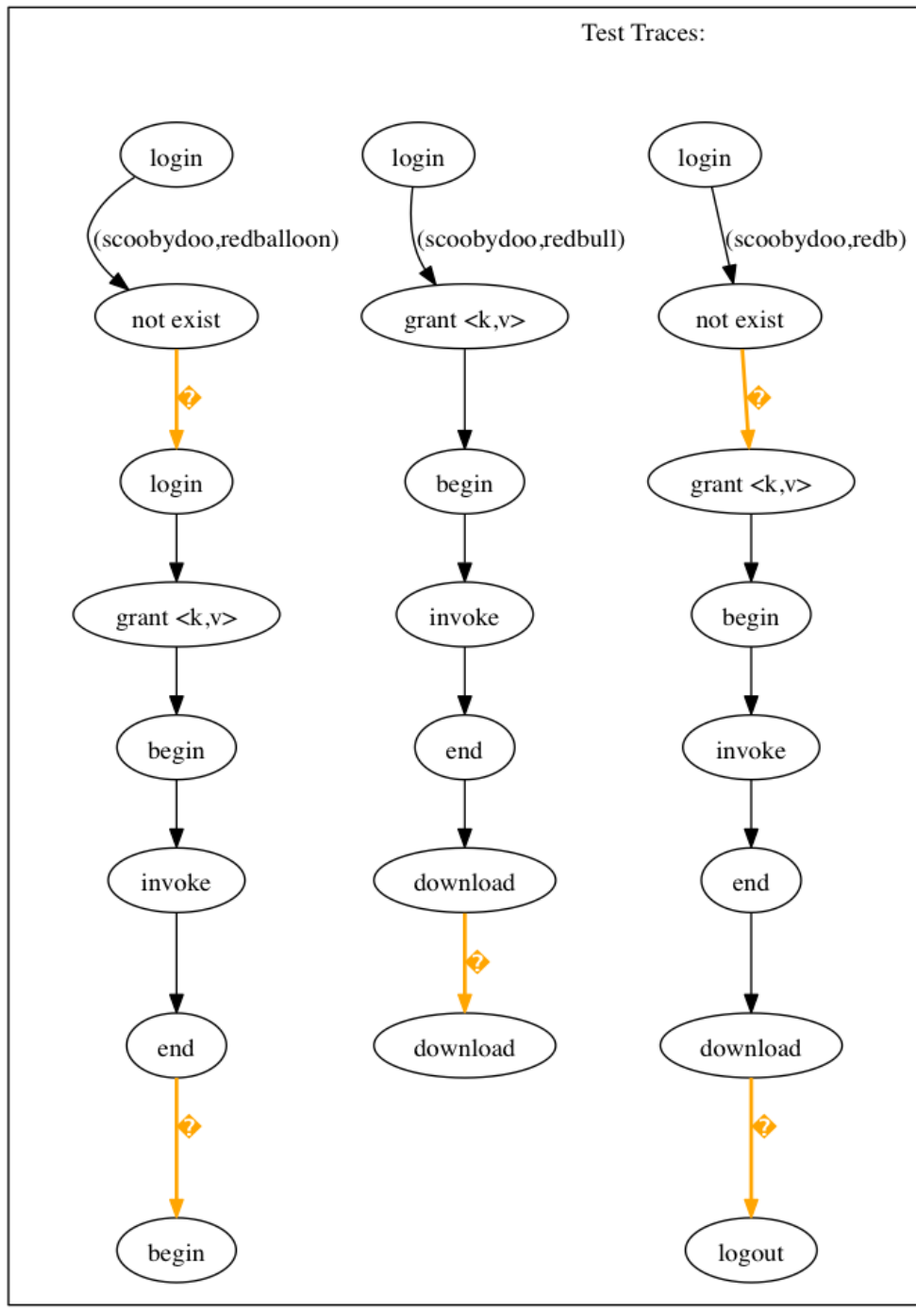
Execution mapping: \k<ip>

Synoptic

Generated Model:

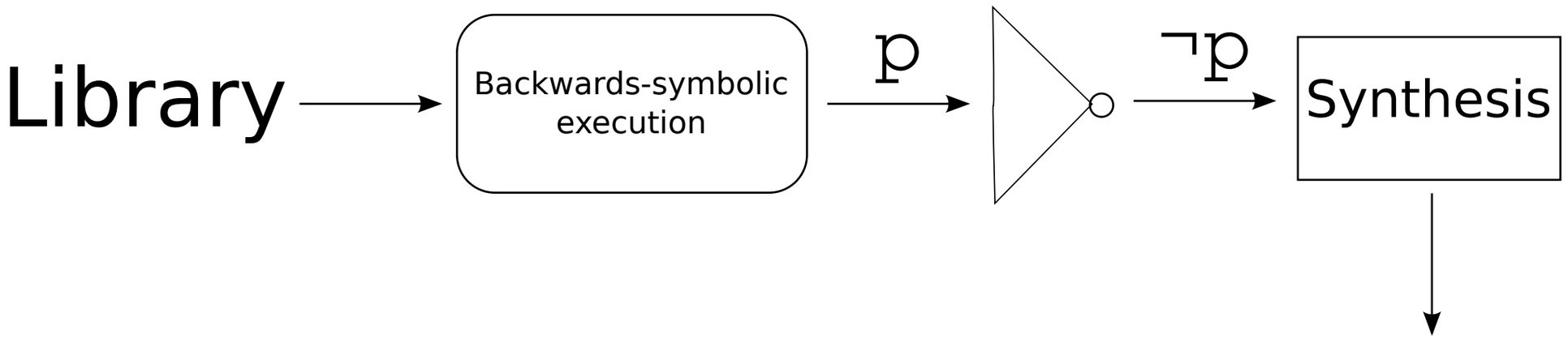






Nicholas Vanderweit

Leveraging Symbolic Execution to Violate Object Invariants



Nicholas Vanderweit
University of Colorado Boulder

Pavel Panchekha

Pavel Panchekha

University of Washington

Peng Wang

Compositional Compilation

- (Horizontally) Can link with other modules
- (Vertically) Can compose compiler passes
- (Higher-order) Coder pointers
- Mechanized proof

Peng (Perry) Wang
MIT

Phil Nguyen

- Contracts
  - enforce invariants dynamically
  - delay error discovery
  - introduce overheads
    - are hard to verify due to expressiveness
- Verification
  - uses abstract reduction semantics
  - scales to many language features

Philip Johnson-Freyd

Extending the Classical Sequent Calculus

Syntax

Commands: $c ::= \langle v || e \rangle$

Terms: $v ::= \mu\alpha.c \mid x \mid \textbf{case}[\texttt{call}(x, \alpha).c] \mid \texttt{pair}(v, v)$

Co-Terms: $e ::= \tilde{\mu}x.c \mid \alpha \mid \texttt{call}(v, e) \mid \textbf{case}[\texttt{pair}(x, y).c]$

Semantics

$$\langle \mu\alpha.c || E \rangle = c\{E/\alpha\} \qquad \langle V || \tilde{\mu}x.c \rangle = c\{V/x\}$$

$$\langle \textbf{case}[\texttt{call}(x, \alpha).c] || \texttt{call}(v, e) \rangle = \langle v || \tilde{\mu}x.\, \langle \mu\alpha.c || e \rangle \rangle$$

Plus rule for pairs (and what ever other types we add), η, ... V and E determine evaluation order

Simple Types

$$\frac{}{\Gamma, x : A \vdash x : A | \Delta} \qquad \frac{}{\Gamma | \alpha : A \vdash \alpha : A, \Delta}$$

$$\frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A | \Delta} \qquad \frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma | \tilde{\mu}x.c : A \vdash \Delta}$$

$$\frac{\Gamma \vdash v : A | \Delta \qquad \Gamma | e : A \vdash \Delta}{\langle v || e \rangle : (\Gamma \vdash \Delta)}$$

$$\frac{c : (\Gamma, x : A \vdash \alpha : B, \Delta)}{\Gamma \vdash \textbf{case}[\texttt{call}(x, \alpha).c] : A \to B | \Delta}$$

$$\frac{\Gamma \vdash v : A | \Delta \qquad \Gamma | e : B \vdash \Delta}{\Gamma | \texttt{call}(v, e) : A \to B \vdash \Delta}$$

$$\frac{\Gamma \vdash v_1 : A | \Delta \qquad \Gamma \vdash v_2 : B \Delta}{\Gamma \vdash \texttt{pair}(v_1, v_2) : A \otimes B | \Delta}$$

$$\frac{c : (\Gamma, x : A, y : B \vdash \Delta)}{\Gamma | \textbf{case}[\texttt{pair}(x, y).c] : A \otimes B \vdash \Delta}$$

Problem with Polymorphism

"Natural" second order quantification rules = not type safe when implicit

$$\langle \mu\alpha.\, \langle ((\lambda x.x), (\lambda f.\mu\_.\, \langle (f, \lambda\_.())||\alpha\rangle))||\alpha\rangle || \textbf{case}[(f, g).\, \langle f || \text{“}bam\text{”} \cdot \tilde{\mu}\_.\, \langle g || (\lambda x.x + 1) \cdot \beta \rangle \rangle] \rangle$$

ML value restriction doesn't dualize well

Shayak Sen

We can prove protocols secure.

… And why that may not good enough

Proofs of Crypto Protocols are hard.

PL Techniques have helped produce formal proofs of protocols
Spi-Calculus, ProVerif, PCL, CryptoVerif, F7

- But how secure is your secure crypto protocol?
  - Of course someone could break it in a hundred years.
    - Could they break it in a month?
      - How often?

- Want to formally prove exact probabilistic bounds on protocols.
  - Metatheory requires a probabilistic language with precise cost semantics
  - Concurrency and adversarial scheduling to reason about multiple sessions
    - The challenge is efficient bounds

Shayak Sen
Work in progress: with Anupam Datta, Joe Halpern, John Mitchell and Arnab Roy

Carnegie Mellon University

Shuying Liang

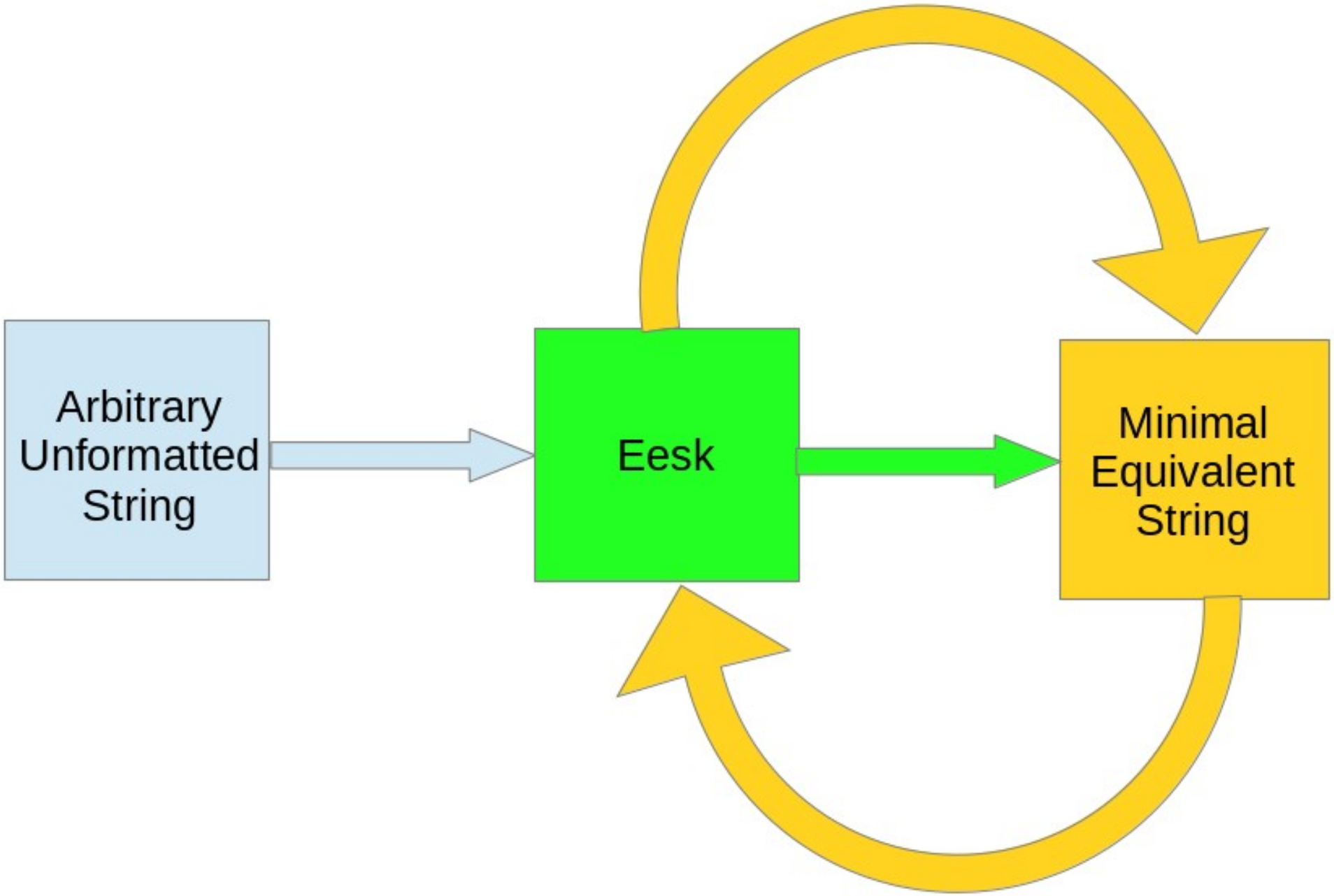
Pruning, Pushdown-Exception Flow Analysis

University of Utah

Shuying Liang

Theron Rabe

Eesk



Theron Rabe: trabe09@winona.edu

Thomas Wood

Thomas Wood

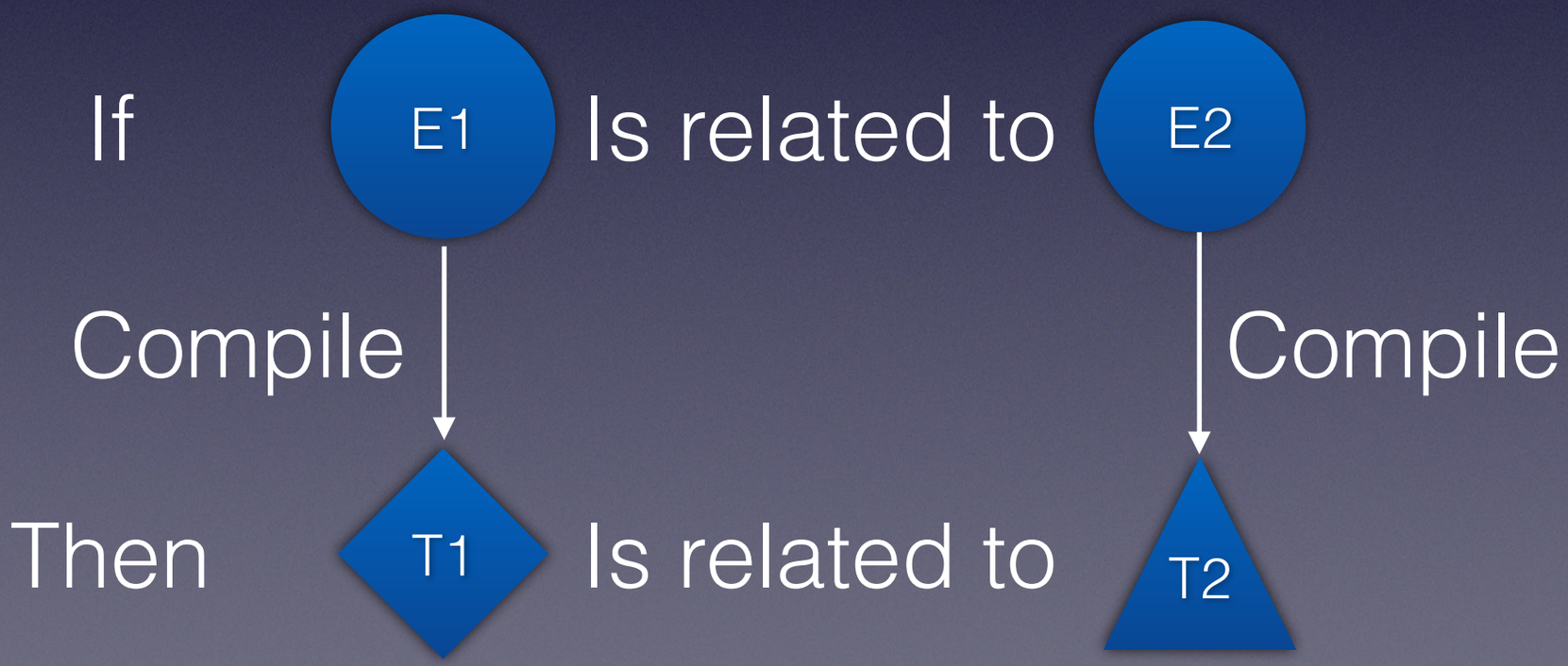
1st Year PhD student, Imperial College London

- Verifying Secure ECMAScript programs
  - SES: allows functions across sandbox boundaries
  - Do we unintentionally leak data or functionality?
  - Program logic for security properties
- Testing JS interpreters
  - JS test suite: Notoriously incomplete
  - JSCert: Executable formal semantics for JavaScript
  - Can we automatically do better?

William J. Bowman

Security Preserving Compilation

William J. Bowman
Northeastern University

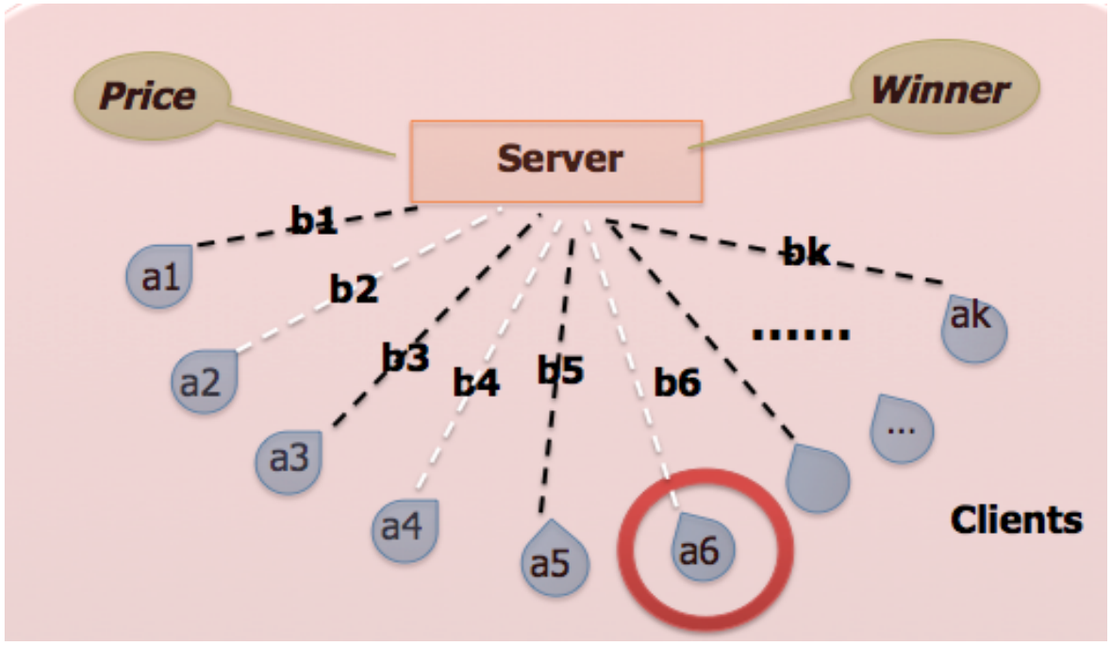


Ye Fang

Computer-aided mechanism design

Ye Fang, Swarat Chuaduri, Moshi Vardi @ Rice Univerisy

Come to poster session if you are interested!



Example:
Second price mechanism
->Vickrey auction

Generalized second price mechanism
-> Google online ads auction

...

precondition on inputs

```
a_i=
  real bid;
 real value;
 utility_function(outcome) =
    if(a_i is winner)
         ut_i = value – price;
    else
         ut_i =0;
```

postcondition on output

```
M(list of bids)=
    winner = ?;
    price = ?;
  return (winner, price)
```

$$\exists M, \forall inputs,\ precondition(inputs) \implies postcondition(output)$$

SAT

SMT solver

UNSAT

No

Yi Li

Symbolic Optimization with SMT Solvers

Yi Li – University of Toronto

- SMT solvers are used everywhere
- Extending SMT solvers to do optimization

- Applications in PL:
  - Numerical invariant generation
  - Program synthesis, counterexample generation
  - Constraint programming
  - Many others …
- Check out our poster and paper at POPL

Yuki Ishii

Type Debugging

- Embedded Type Debugger [Tsushima, ‘12]
  - interactive
  - OCaml
- Analyze the error logs of novice students
- How can we design “novice-friendly” error messages or languages?

```
let rec fold f init lst = match lst with
    [] -> init
  | first :: rest ->
    f first (fold f init rest)

let length lst = fold (fun left _ _ right -> left + right + 1) 0 lst
```

The 2nd argument of this application has wrong type.

- “argument”? “application”?
- “this” … which?
- why “wrong”?
- higher-order function…