What Program Analysis Can and Cannot Do for You

David Van Horn

with support from NSF, CRA, Google.

A Formulae-as-Types Notion of Control

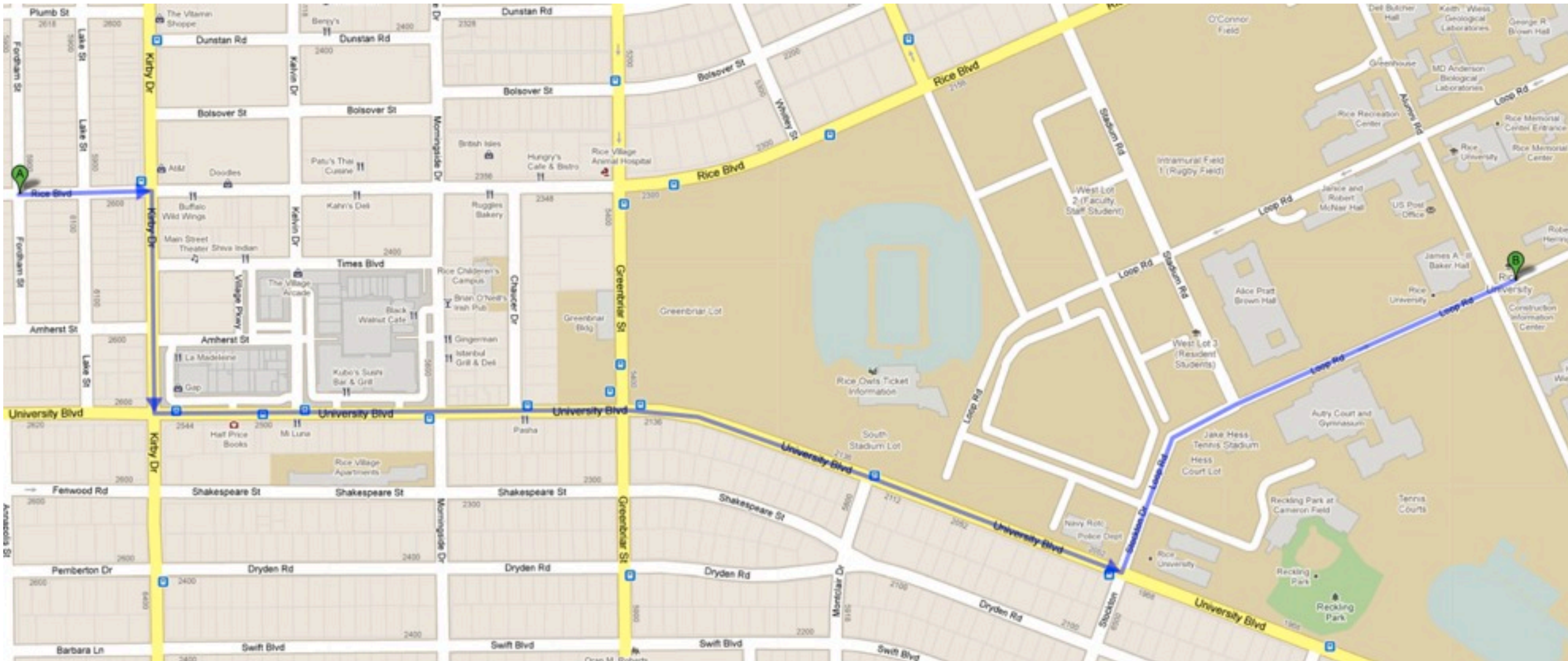
Timothy G. Griffin*
Department of Computer Science
Rice University
Houston, TX 77251-1892

Abstract

The programming language Scheme contains the control construct `call/cc` that allows access to the current continuation (the current control context). This, in effect, provides Scheme with first-class labels and jumps. We show that the well-known formulae-as-types correspondence, which relates a constructive proof of a formula α to a program of type α, can be extended to a typed Idealized Scheme. What is surprising about this correspondence is that it relates *classical* proofs to typed programs. The existence of computationally interesting "classical programs" — programs of type α, where α holds classically, but not constructively — is illustrated by the definition of conjunctive, disjunctive, and existential types using standard classical definitions. We also prove that

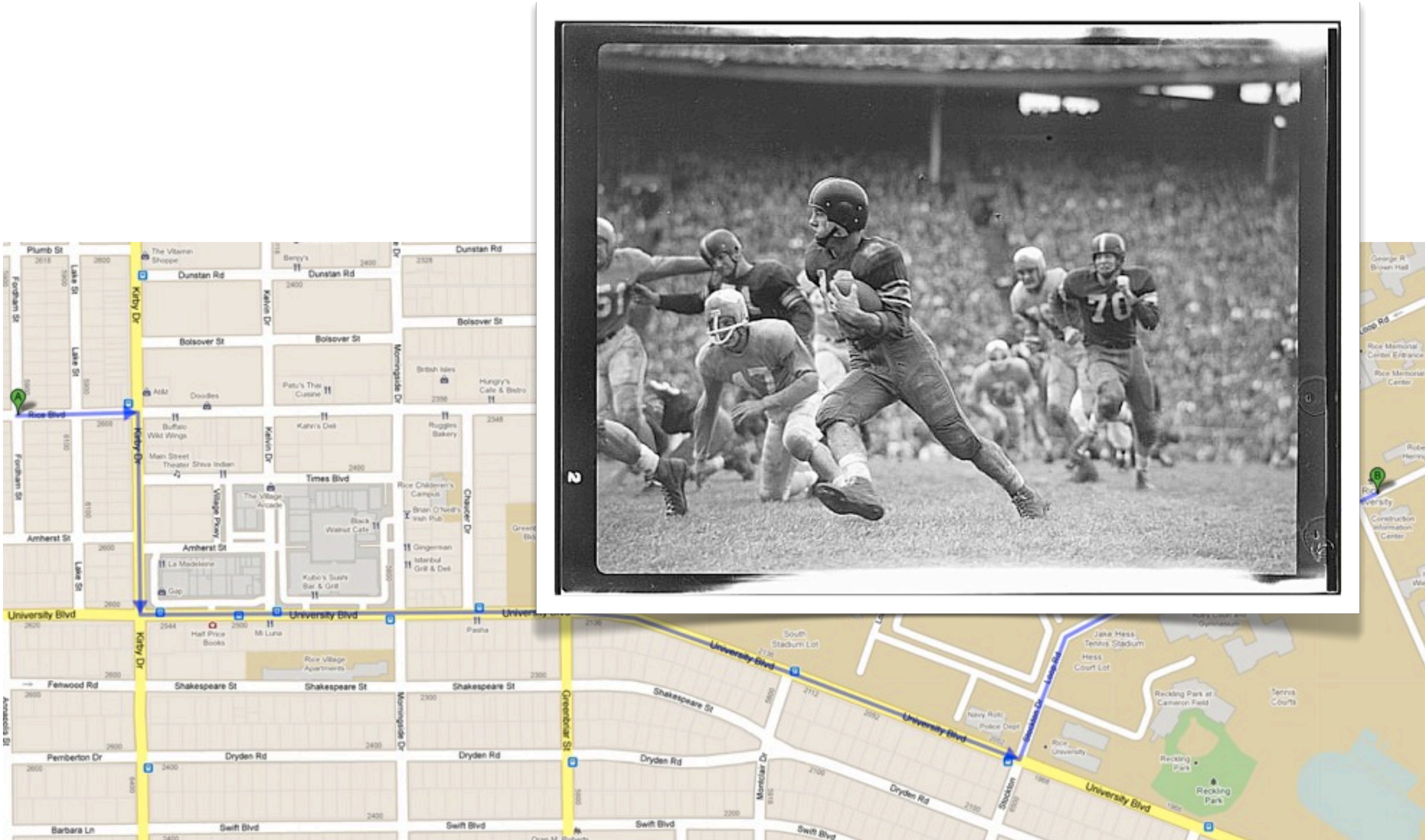
in general, classical proofs lack computational content. This paper shows, however, that the formulae-as-types correspondence *can* be extended to classical logic in a computationally interesting way. It is shown that classical proofs posses computational content when the notion of computation is extended to include explicit access to the current control context.

This notion of computation is found in the programming language Scheme [16], which contains the control construct `call/cc`[1] that provides access to the current continuation (the current control context). This, in effect, provides Scheme with first-class labels and jumps, and allows for programs that are more efficient than purely functional programs. The formulae-as-types correspondence presented in this paper is based on a typed version of *Idealized Scheme* — a typed ISWIM containing an operator



classical proofs to typed programs. The existence of computationally interesting “classical programs” — programs of type α, where α holds classically, but not constructively — is illustrated by the definition of conjunctive, disjunctive, and existential types using standard classical definitions. We also prove that

text). This, in effect, provides Scheme with first-class labels and jumps, and allows for programs that are more efficient than purely functional programs. The formulae-as-types correspondence presented in this paper is based on a typed version of *Idealized Scheme* — a typed ISWIM containing an operator



classical proofs to typed programs. The existence of computationally interesting “classical programs” — programs of type α, where α holds classically, but not constructively — is illustrated by the definition of conjunctive, disjunctive, and existential types using standard classical definitions. We also prove that

text). This, in effect, provides Scheme with first-class labels and jumps, and allows for programs that are more efficient than purely functional programs. The formulae-as-types correspondence presented in this paper is based on a typed version of *Idealized Scheme* — a typed ISWIM containing an operator

What Program Analysis Can and Cannot Do for You

David Van Horn

with support from NSF, CRA, Google.

Higher-order Program Analysis is Dead.

(I should know, I killed it.)

it's hard to write
it's slow
it's imprecise
it's awful

Higher-order Program Analysis is Alive and Well.

(I have a way forward.)

it’s easy to write
it’s fast
it’s precise
it’s great

So what?

Modern software is higher-order.

We need reasonable software.

So you should care.

Modern software is higher-order.

Modern software is higher-order.

Q: What are higher-order languages?

Modern software is higher-order.

Q: What are higher-order languages?

A: Languages in which computations are values.

Modern software is higher-order.

Python

```
# (ℝ → ℝ) → (ℝ → ℝ)
def deriv(f):
    def fp(x):
        return ((f(x+e) - f(x-e)) / (2*e));
    return fp
```

Modern software is higher-order.

Java

```
// (ℝ → ℝ) → (ℝ → ℝ)
public Func<Double, Double>
deriv(final Func<Double, Double> f) {
    return new Func<Double,Double>() {
        public Double apply(Double x) {
            return ((f.apply(x+ε) - f.apply(x-ε)) / (2*ε));
        }
    };
}
```

Modern software is higher-order.

C#

```
// (ℝ → ℝ) → (ℝ → ℝ)
static Func<double, double>
deriv(Func<double, double> f) {
  return (x)
    => (f(x+ε) - f(x-ε)) / (2*ε);
}
```

Modern software is higher-order.

JavaScript

```
// (ℝ → ℝ) → (ℝ → ℝ)
function deriv(f) {
  return function (x) {
    return (f(x+ε) - f(x-ε)) / (2*ε);
  };
}
```

Modern software is higher-order.

X10

```
// (ℝ → ℝ) → (ℝ → ℝ)
def deriv(f: (Double) => Double) {
  val fp = (x: Double) =>
    (f(x+ε) - f(x-ε)) / (2*ε);
  return fp;
}
```

An Introduction To Programming With X10

DRAFT

Jonathan Brezin, brezin@us.ibm.com
Stephen J. Fink, sjfink@us.ibm.com
with
Bard Bloom, bardb@us.ibm.com
Cal Swart, cals@us.ibm.com

Please send comments to brezin@us.ibm.com.

December 2, 2010

```
public class IntRange {
   val low: Int;
   var high: Int;
   public def this(low: Int, high: Int) {
      this.low = low; this.high = high;
   }
   public def includes(n:Int) = low <= n && n <= high;
   public static def isDigitFcn() {
      val digit = new IntRange(0,9);
      return (n: Int) => digit.includes(n);
   }
   public def inMeTester() {
      return (n: Int) => low <= n && n <= high;
   }
}
```

An Introduction To Programming With X10

DRAFT

Jonathan Brezin, brezin@us.ibm.com
Stephen J. Fink, sjfink@us.ibm.com
with
Bard Bloom, bardb@us.ibm.com
Cal Swart, cals@us.ibm.com

Please send comments to brezin@us.ibm.com.

December 2, 2010

```
public class IntRange {
   val low: Int;
   var high: Int;
   public def this(low: Int, high: Int) {
      this.low = low; this.high = high;
   }
   public def includes(n:Int) = low <= n && n <= high;
   public static def isDigitFcn() {
      val digit = new IntRange(0,9);
      return (n: Int) => digit.includes(n);
   }
   public def inMeTester() {
      return (n: Int) => low <= n && n <= high;
   }
}
```

window.setTimeout

WATCH EDIT

« Gecko DOM Reference

Summary

Executes a code snippet or a function after specified delay.

Syntax

```
var timeoutID = window.setTimeout(func, delay, [param1, param2, ...]);
var timeoutID = window.setTimeout(code, delay);
```

where

- `timeoutID` is the ID of the timeout, which can be used later with window.clearTimeout.
- `func` is the function you want to execute after `delay` milliseconds.
- `code` in the alternate syntax, is a string of code you want to execute after `delay` milliseconds. (Using this syntax is not recommended for the same reasons as using eval())
- `delay` is the number of milliseconds (thousandths of a second) that the function call should be delayed by. Note that the actual delay may be longer, see Notes below.

TABLE OF CONTENTS

TAGS FILES

Page Notifications Off

MDN Explore MDN

Log in | Become an MDN member

Search MDN

powered by Google

MDC › Doc Center › DOM › window.setTimeout

Languages ▾ This page ▾ Site tools ▾

window.setTimeout

WATCH EDIT

« Gecko DOM Reference

Summary

Executes a code snippet or a function after specified delay.

Syntax

```
var timeoutID = window.setTimeout(func, delay, [param1, param2, ...]);
var timeoutID = window.setTimeout(code, delay);
```

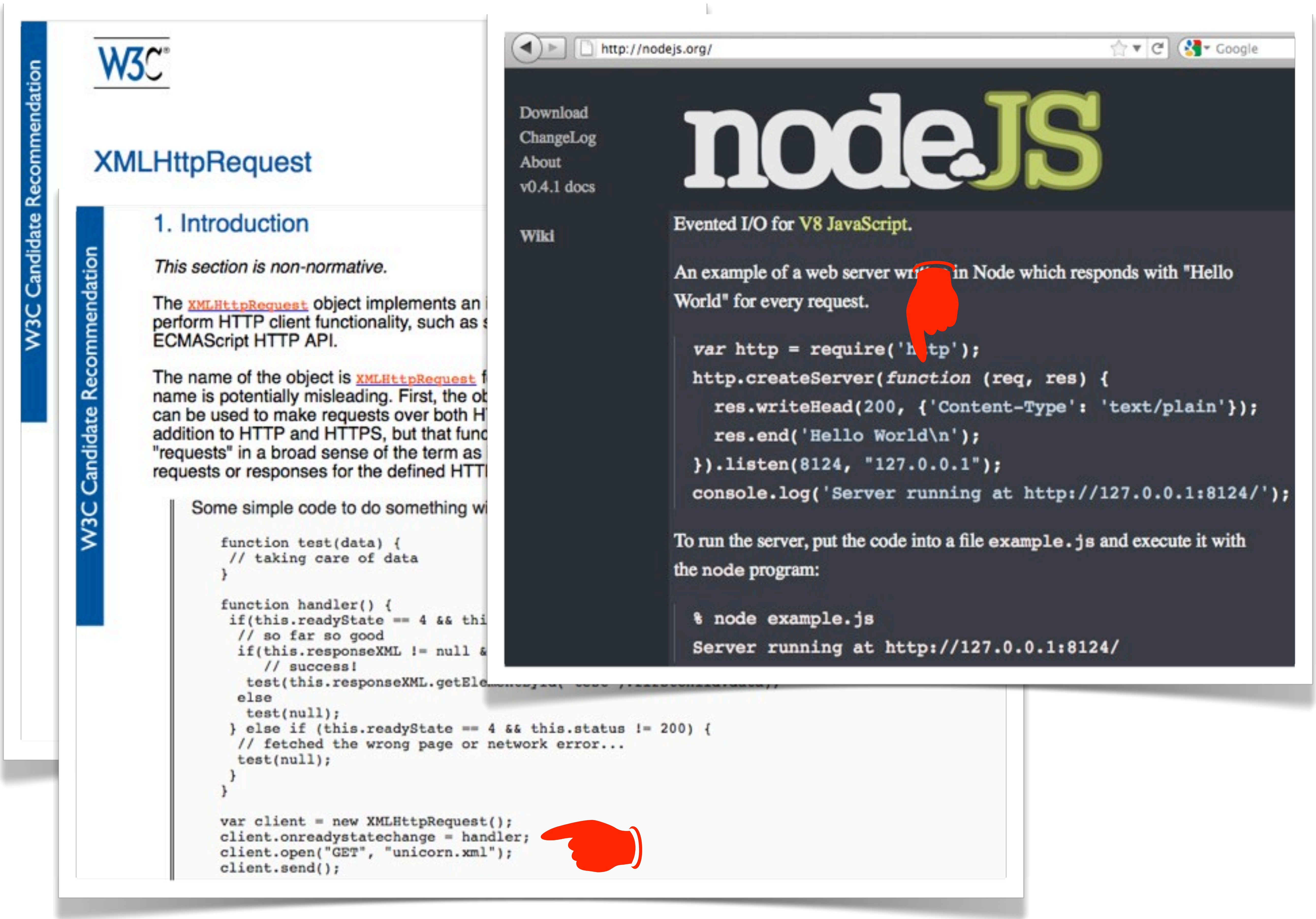
where

- `timeoutID` is the ID of the timeout, which can be used later with window.clearTimeout.
- `func` is the function you want to execute after `delay` milliseconds.
- `code` in the alternate syntax, is a string of code you want to execute after `delay` milliseconds. (Using this syntax is not recommended for the same reasons as using eval())
- `delay` is the number of milliseconds (thousandths of a second) that the function call should be delayed by. Note that the actual delay may be longer, see Notes below.

TABLE OF CONTENTS

TAGS FILES

Page Notifications Off


W3C Candidate Recommendation
XMLHttpRequest
W3C Candidate Recommendation
1. Introduction
This section is non-normative.
The XMLHttpRequest object implements an
perform HTTP client functionality, such as
ECMAScript HTTP API.
The name of the object is XMLHttpRequest f
name is potentially misleading. First, the ob
can be used to make requests over both H
addition to HTTP and HTTPS, but that func
"requests" in a broad sense of the term as
requests or responses for the defined HTT
Some simple code to do something wi
function test(data) {
// taking care of data
}
function handler() {
if(this.readyState == 4 && thi
// so far so good
if(this.responseXML != null &
// success!
test(this.responseXML.getEle
else
test(null);
} else if (this.readyState == 4 && this.status != 200) {
// fetched the wrong page or network error...
test(null);
}
}
var client = new XMLHttpRequest();
client.onreadystatechange = handler;
client.open("GET", "unicorn.xml");
client.send();
http://nodejs.org/
Google
Download
ChangeLog
About
v0.4.1 docs
Wiki
nodeJS
Evented I/O for V8 JavaScript.
An example of a web server wr in Node which responds with "Hello World" for every request.
var http = require('h tp');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/plain'});
res.end('Hello World\n');
}).listen(8124, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8124/');
To run the server, put the code into a file example.js and execute it with the node program:
% node example.js
Server running at http://127.0.0.1:8124/

java.util

Class Observable

java.lang.Object
  └ **java.util.Observable**

`public class` **`Observable`**
`extends Object`

This class represents an observable object, or "data" in the model-view paradigm. It can be subclassed to represent an object that the application wants to have observed.

An observable object can have one or more observers. An observer may be any object that implements interface `Observer`. After an observable instance changes, an application calling the `Observable`'s `notifyObservers` method causes all of its observers to be notified of the change by a call to their `update` method.

The order in which notifications will be delivered is unspecified. The default implementation provided in the Observable class will notify Observers in the order in which they registered interest, but subclasses may change this order, use no guaranteed order, deliver notifications on separate threads, or may guarantee that their subclass follows this order, as they choose.

Note that this notification mechanism is has nothing to do with threads and is completely separate from the `wait` and `notify` mechanism of class `Object`.

When an observable object is newly created, its set of observers is empty. Two observers are considered the same if and only if the `equals` method returns true for them.

Since:
  JDK1.0

See Also:
  `notifyObservers()`, `notifyObservers(java.lang.Object)`, `Observer`, `Observer.update(java.util.Observable, java.lang.Object)`

java.util

Class Observable

java.lang.Object
  └java.util.Observable

```
public class Observable
extends Object
```

This class represents an observable object, or "data" in the model-view paradigm. that the application wants to have observed.

An observable object can have one or more observers. An observer may be any ob After an observable instance changes, an application calling the Observable's not

Constructor Summary

Observable() Construct an Observable with zero Observers.	

Method Summary

void	**addObserver**(Observer o) Adds an observer to the set o ers for this obj already in the set.
protected void	**clearChanged**() Indicates that this object has no longer chang most recent change, so that the hasChanged method w
int	**countObservers**() Returns the number of observers of this Observabl
void	**deleteObserver**(Observer o) Deletes an observer from the set of observers of this
void	**deleteObservers**() Clears the observer list so that this object no longer has any obs
boolean	**hasChanged**() Tests if this object has changed.
void	**notifyObservers**() If this object has changed, as indicated by the hasChanged meth then call the clearChanged method to indicate that this object has no
void	**notifyObservers**(Object arg) If this object has changed, as indicated by the hasChanged meth then call the clearChanged method to indicate that this object has no
protected void	**setChanged**() Marks this Observable object as having been changed; the has

Programming Ruby

The Pragmatic Programmer's Guide

Previous < Contents ^ Next >

Object-Oriented Design Libraries

One of the interesting things about Ruby is the way it blurs the distinction between design and implementation. Ideas that have to be expressed at the design level in other languages can be implemented directly in Ruby.

To help in this process, Ruby has support for some design-level strategies.

- **The Visitor pattern** (Design Patterns,) is a way of traversing a collection without having to know the internal organization of that collection.
- **Delegation** is a omposing classes more flexibly and dynamically than can be done g standard inheritance.
- **The Singleton pattern** is a way of ensuring that only one instantiation of a particular class exists at a time.
- **The Observer pattern** implements a protocol allowing one object to notify a

Normall implem and trar

Library: observer

The Observer pattern, also known as Publish/Subscribe, provides a simple mechanism for one object to inform a set of interested third-party objects w state changes.

In the Ruby implementation, the notifying class mixes in the Observable mo which provides the methods for managing the associated observer objects

add_observer(*obj*)	*obj* as an observer on this object. *obj* will no receive notifications.
delete_observer(*obj*)	Delete *obj* as an observer on this object. It will n longer receive notifications.

Modern software is higher-order.

C++ | Java | JavaScript

Python | Scheme

C# | X10 / Habanero | OCaml

...and many more

Modern software is higher-order.

C++ | Java | JavaScript

Python | Scheme

C# | X10 / Habanero | OCaml

...and many more

Higher order

Modern software is higher-order.

C++ (360)
Java (201,211,215,310,402)
JavaScript (327)
Python (140,182,327)
Scheme (211,311)
C# (160,402,410,460)
X10 / Habanero (322)
OCaml (311,411)

...and many more

Higher order

We need reasonable software.

We need reasonable software.

Q: What does it mean to reason about software?

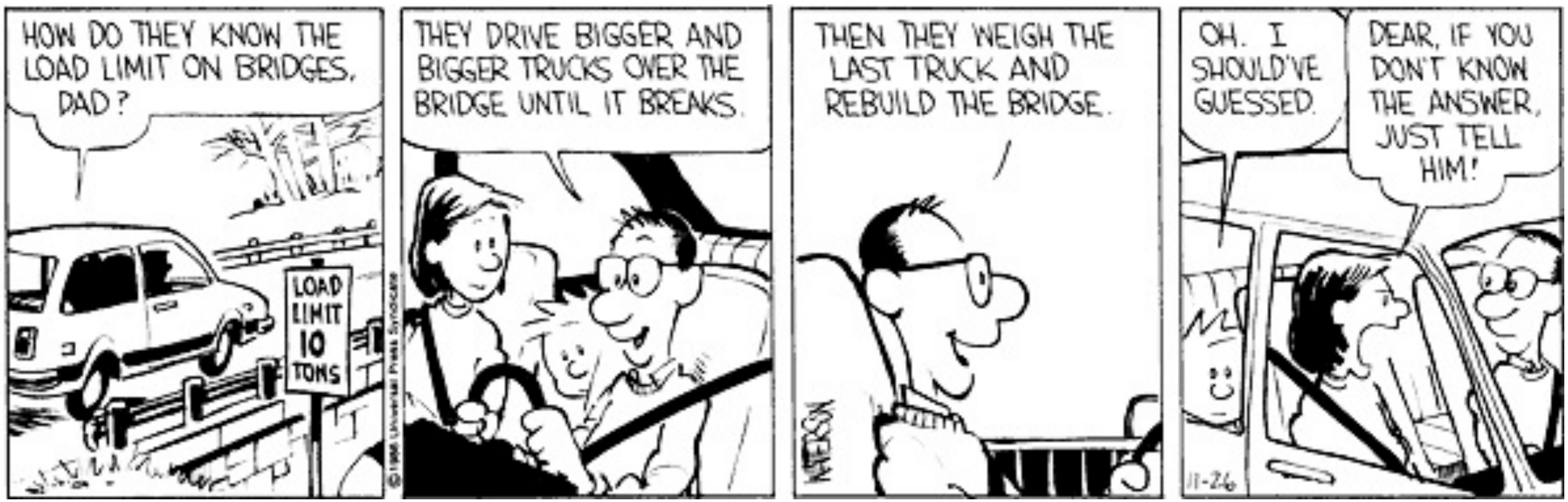
We need reasonable software.

Q: What does it mean to reason about software?

A: It means predicting the future.

We need reasonable software.

We need reasonable software.



We need reasonable software.

```
public void f(XYZ x) {
    x.m();
}
```

Optimizing Java compiler:
prove `x` is always an `X`, inline method definition.

We need reasonable software.

```
first(x)
```

Puzzled ML programmer:
  prove `x` is always a non-empty list: no problem.

We need reasonable software.

```
first(x)
```

Puzzled ML programmer:
prove `x` is *may* be the empty list: fix.

We need reasonable software.

```
checkPrivilege(R);
```

Security analyzer:
prove enable(R) is on the stack.

We need reasonable software.

Optimizing compilers

Parallelizing compilers

Software construction

Security analysis

Program understanding

Static debugging

Termination analysis

Model checking

...and many more

We need reasonable software.

Optimizing compilers
Parallelizing compilers
Software construction
Security analysis
Program understanding
Static debugging
Termination analysis
Model checking

...and many more

Program analysis

Higher-order program analysis

Scalability $\begin{cases} \text{Complexity} \\ \text{Maintenance} \\ \text{Verification} \\ \text{Expressivity} \\ \text{Modularity} \end{cases}$

Q: What is program analysis?

Q: What is program analysis?

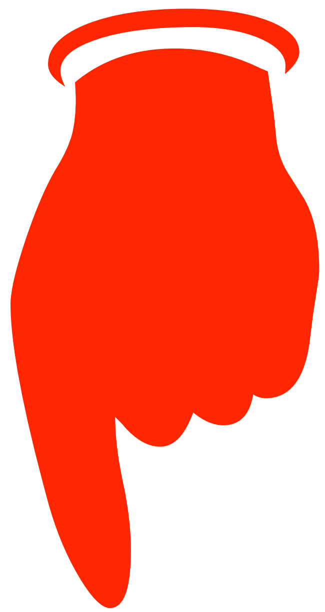
A: Prediction of which values show up at which program sites.

C#

`f(x+ε)`

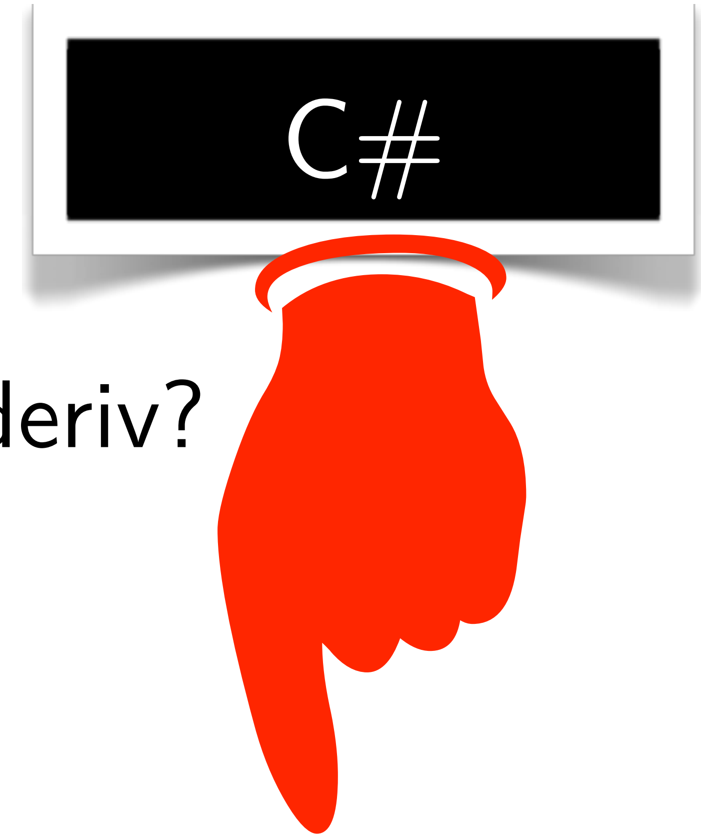
Where does data go to?

C#



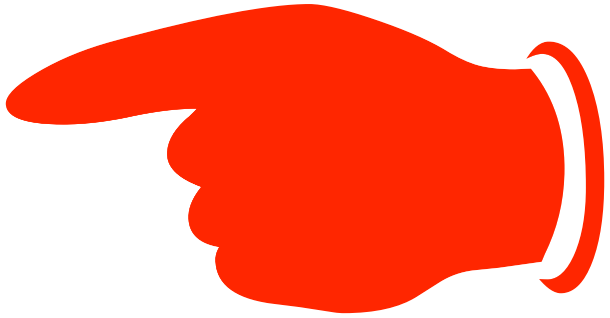
Where does control go to?

`f(x+ε)`



Who calls deriv?

```
deriv(Func<double, double> f) {
  return (x)
    => (f(x+ε) - f(x-ε)) / (2*ε);
}
```



To do control-flow analysis,
you need data-flow analysis

To do data-flow analysis,
you need control-flow analysis

Why so tangled up?

Values include Computations

Computation is Code plus Data

Why so tangled up?

Values include Computations

Computation is Code plus Data

Why so tangled up?

Values include Computations

Computation is Code plus Data

Computable predictions about run-time behavior

Why so tangled up?

Values include Computations

Computation is Code plus Data

Computable predictions about run-time behavior

So what's their complexity?

Existing analyses and their complexity

0CFA

```
function app(f,x) { return f(x); };
```

```
app(sqr,4);      app(dbl,5);
```

0CFA

```
function app(f,x) { return f(x); };

  app(sqr,4);        app(dbl,5);
```

0CFA

```
function app(f,x) { return f(x); };
```

```
app(sqr,4);        app(dbl,5);
```

0CFA

```
function app(f,x) { return f(x); };
```

{sqr}

```
app(sqr,4);        app(dbl,5);
```

0CFA

```
function app(f,x) { return f(x); };
```

{sqr} {4}

```
app(sqr,4);        app(dbl,5);
```

0CFA

```
function app(f,x) { return f(x); };
```

{sqr} {4}

```
app(sqr,4);        app(dbl,5);
```

0CFA

```
function app(f,x) { return f(x); };
```

{sqr}   {4}

```
app(sqr,4);      app(dbl,5);
```

{sqr(4)}

0CFA

```
function app(f,x) { return f(x); };
```

{sqr}   {4}

```
app(sqr,4);      app(dbl,5);
```

{sqr(4)}

0CFA

```
function app(f,x) { return f(x); };
```

{sqr} {4}

```
app(sqr,4);        app(dbl,5);
```

{sqr(4)}

0CFA

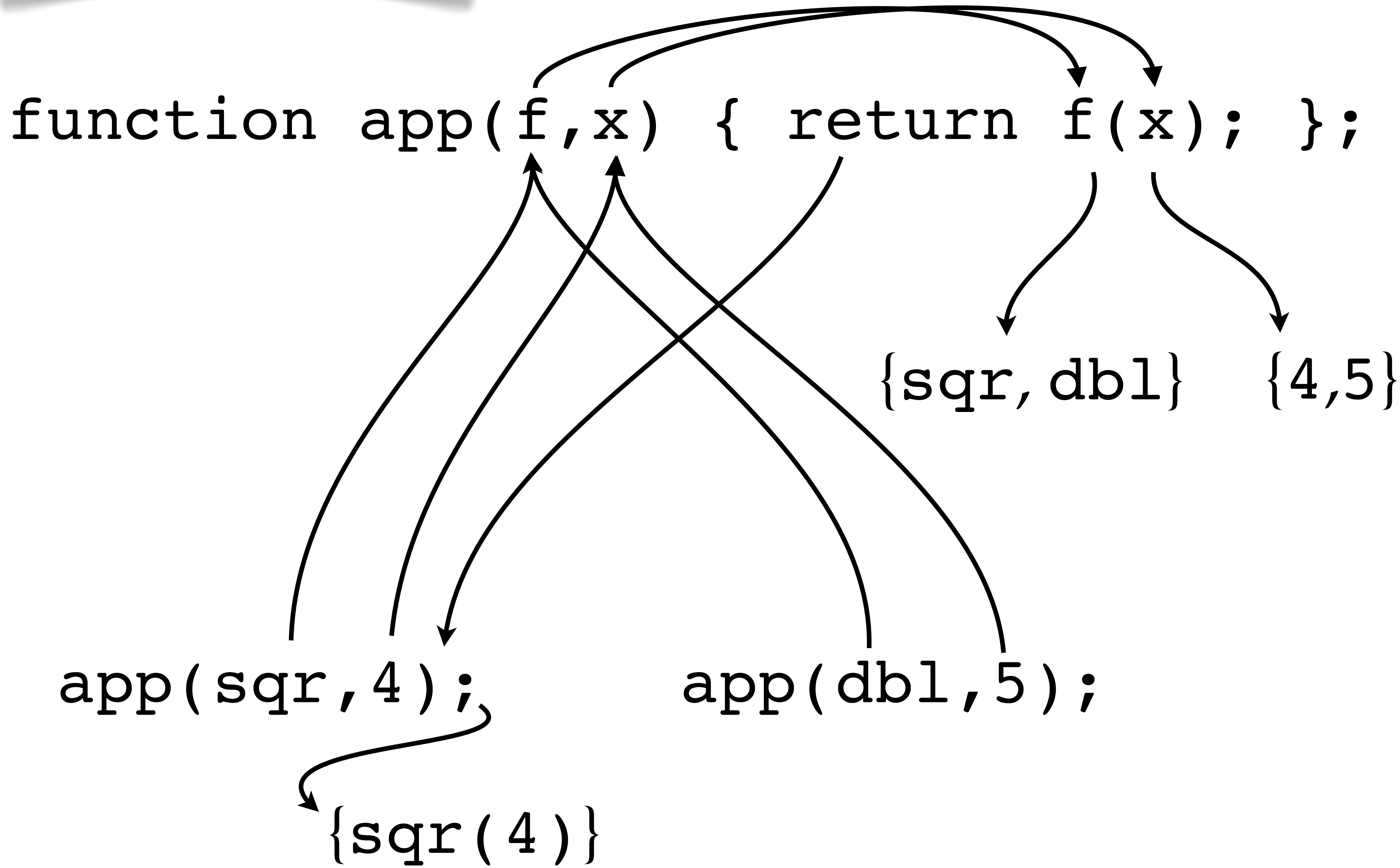
```
function app(f,x) { return f(x); };
```

{sqr, dbl} {4}

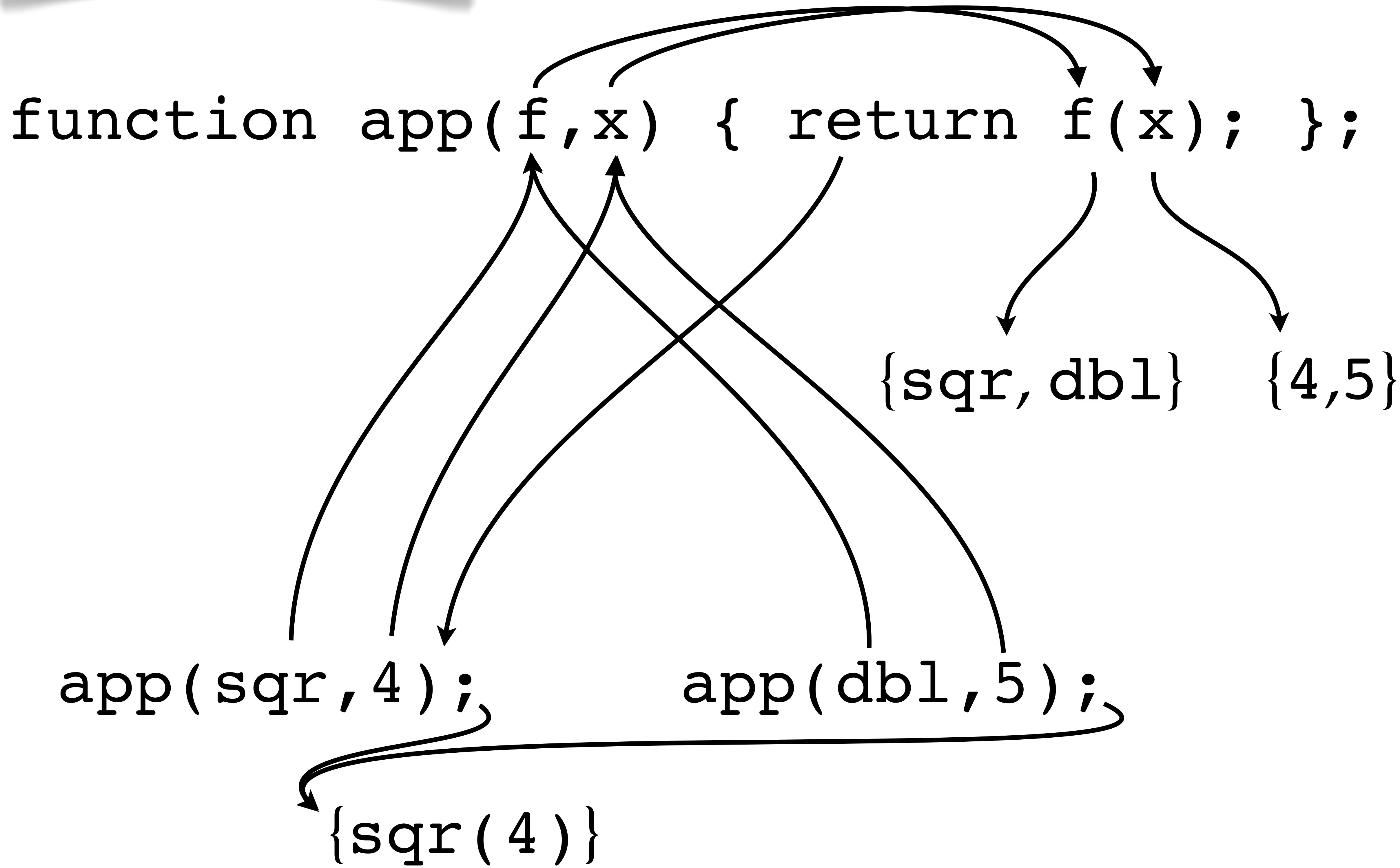
```
app(sqr,4);        app(dbl,5);
```

{sqr(4)}

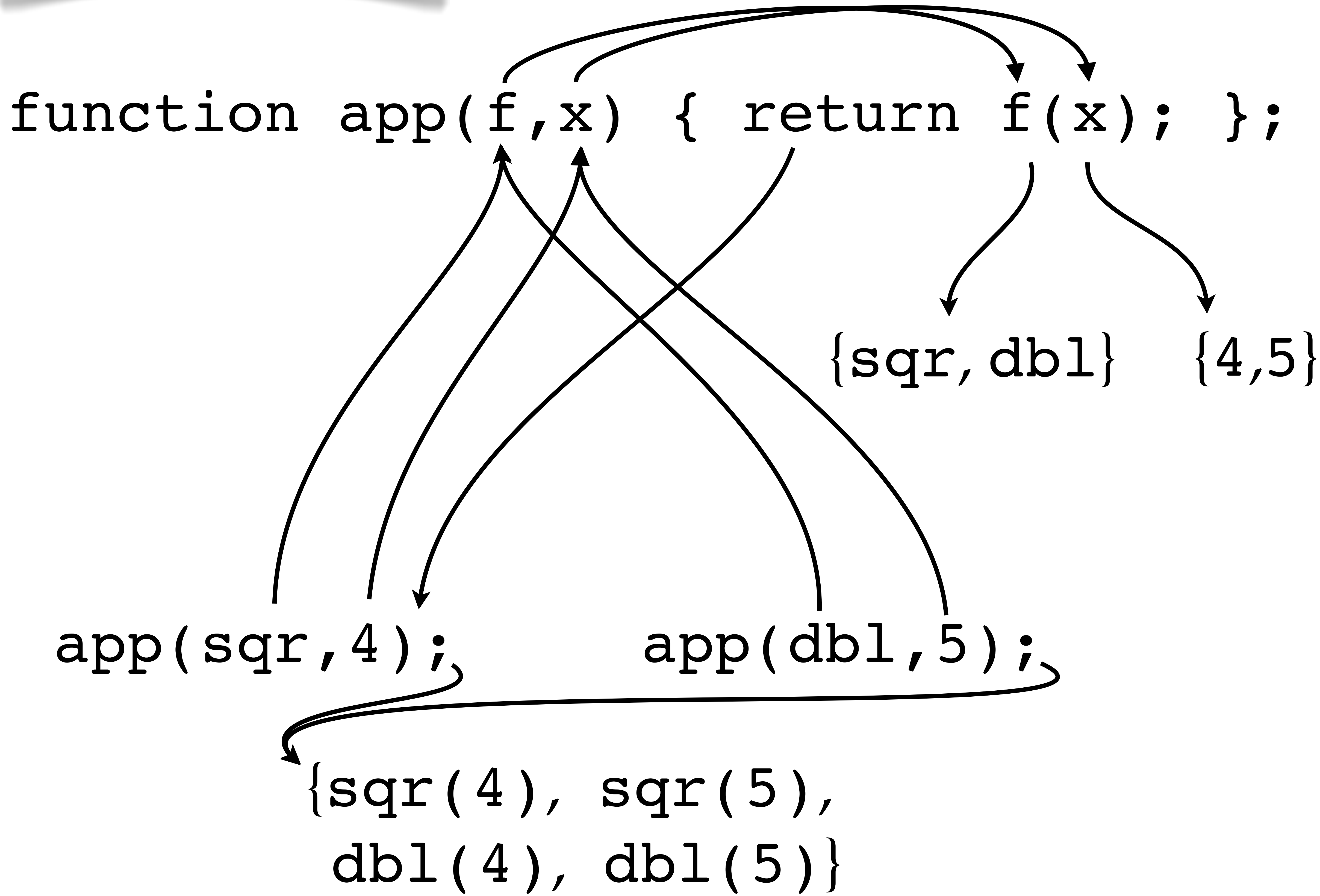
0CFA



0CFA



0CFA


function app(f,x) { return f(x); };
{sqr, dbl}
{4,5}
app(sqr,4);
app(dbl,5);
{sqr(4), sqr(5),
dbl(4), dbl(5)}

0CFA

```
function app(f,x) { return f(x); };
```

{sqr,dbl} {4,5}

```
app(sqr,4);        app(dbl,5);
```

{sqr(4), sqr(5),
dbl(4), dbl(5)}

0CFA

ICFP'07

```
function app(f,x) { return f(x); };
```

{4,5}

Theorem: 0CFA is complete for PTIME.

```
app(sqr,4);          app(dbl,5);
```

```
{sqr(4), sqr(5),
 dbl(4), dbl(5)}
```

Precision

0CFA

Precision

0CFA
Simple closure

Simple closure

```
function app(f,x) { return f(x); };
```

{sqr, dbl} {4,5}

```
app(sqr,4);     app(dbl,5);
```

{sqr(4), sqr(5),
 dbl(4), dbl(5)}

Simple closure

```
function app(f,x) { return f(x); };
```

{sqr,dbl} {4,5}

```
app(sqr,4);      app(dbl,5);
```

{sqr(4), sqr(5),
dbl(4), dbl(5)}

Simple closure

```
function app(f,x) { return f(x); };
```

{sqr, dbl}   {4,5}

```
app(sqr,4);      app(dbl,5);
```

{sqr(4), sqr(5),
dbl(4), dbl(5)}

Simple closure

SAS'08

```
function app(f,x) { return f(x); };
```

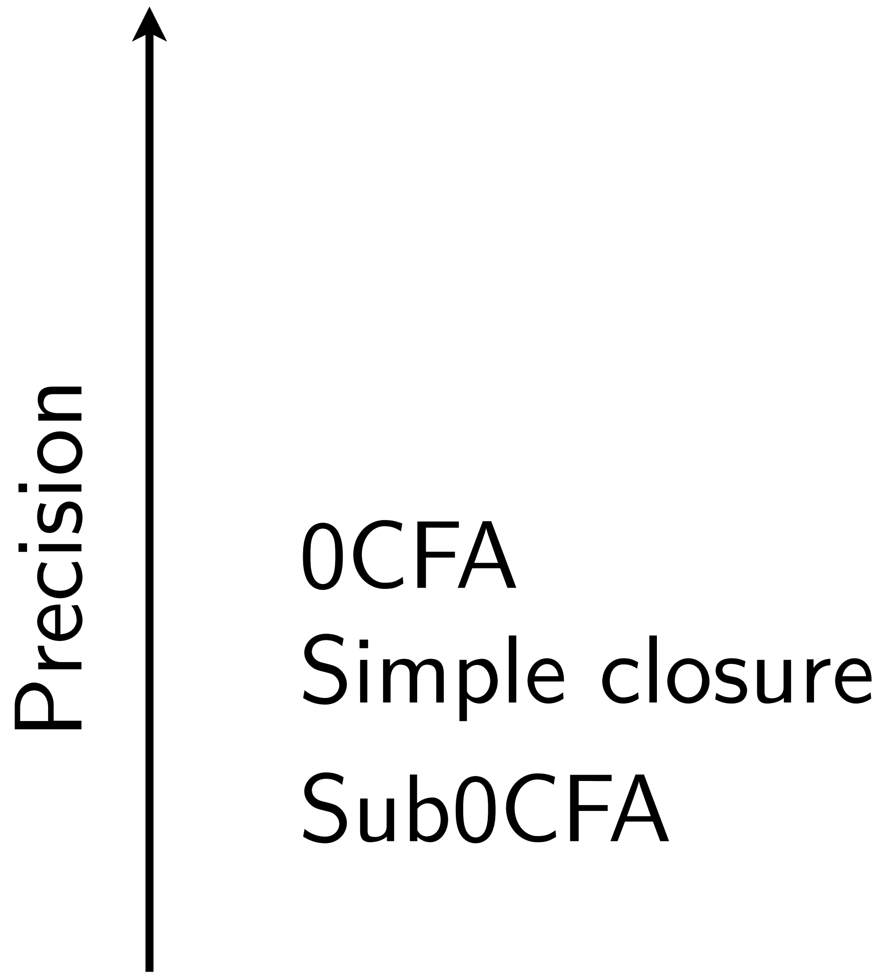
Theorem: Simple closure is complete for PTIME.

```
app(sqr,4);        app(dbl,5);
```

```
{sqr(4), sqr(5),
 dbl(4), dbl(5)}
```

Precision

0CFA
Simple closure


Precision
0CFA
Simple closure
Sub0CFA

Sub0CFA

```
function app(f,x) { return f(x); };
```

{sqr} {4}

```
app(sqr,4);        app(dbl,5);
```

{sqr(4)}

Sub0CFA

```
function app(f,x) { return f(x); };
```

{sqr} {4}

```
app(sqr,4);     app(dbl,5);
```

{sqr(4)}

Sub0CFA

```
function app(f,x) { return f(x); };
```

? {4}

```
app(sqr,4);      app(dbl,5);
```

{sqr(4)}

Sub0CFA

```
function app(f,x) { return f(x); };
```

? ?

```
app(sqr,4);        app(dbl,5);
```

{sqr(4)}

Sub0CFA

```
function app(f,x) { return f(x); };
```

? ?

```
app(sqr,4);      app(dbl,5);
```

{sqr(4)}

Sub0CFA

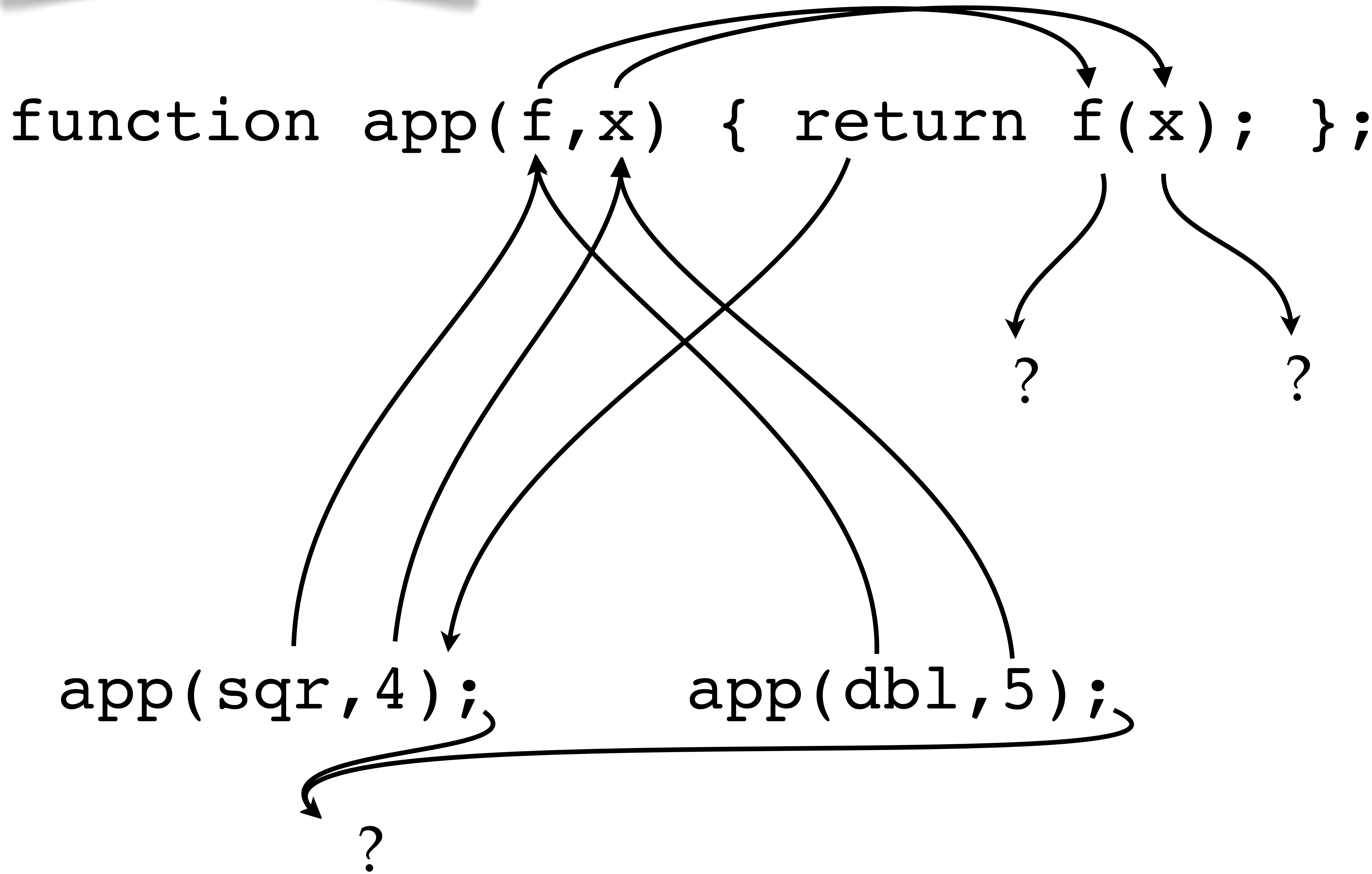
```
function app(f,x) { return f(x); };
```

? ?

```
app(sqr,4);     app(dbl,5);
```

?

Sub0CFA



Sub0CFA

SAS'08

```
function app(f,x) { return f(x); };
```

Theorem: Sub0CFA is complete for PTIME.

```
app(sqr,4);        app(dbl,5);
```

?

Precision

0CFA

Simple closure

Sub0CFA

Precision

0CFA

Simple closure

Sub0CFA

⋮

Whatever

```
function app(f,x) { return f(x); };
```

? ?

```
app(sqr,4);      app(dbl,5);
```

?

Whatever

SAS'08

```
function app(f,x) { return f(x); };
```

Theorem: They're all complete for PTIME.

```
app(sqr,4);        app(dbl,5);
```

?

Precision

0CFA
Simple closure
Sub0CFA
⋮

Precision

1CFA

0CFA

Simple closure

Sub0CFA

⋮

1CFA

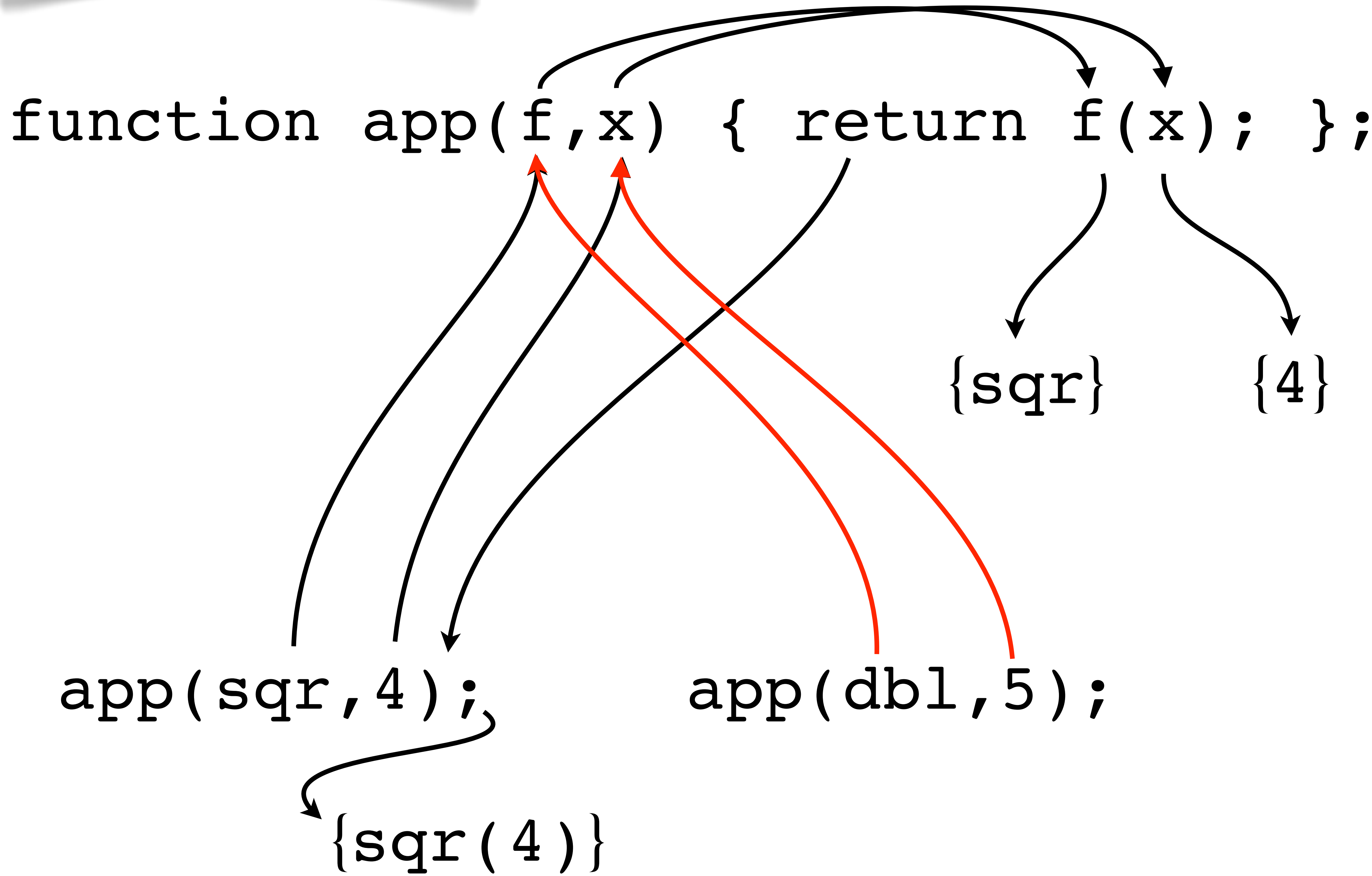
```
function app(f,x) { return f(x); };
```

{sqr} {4}

```
app(sqr,4);     app(dbl,5);
```

{sqr(4)}

1CFA



1CFA

```
function app(f,x) { return f(x); };
```

{sqr} {4}

{dbl} {5}

```
app(sqr,4);        app(dbl,5);
```

{sqr(4)}

1CFA

```
function app(f,x) { return f(x); };
```

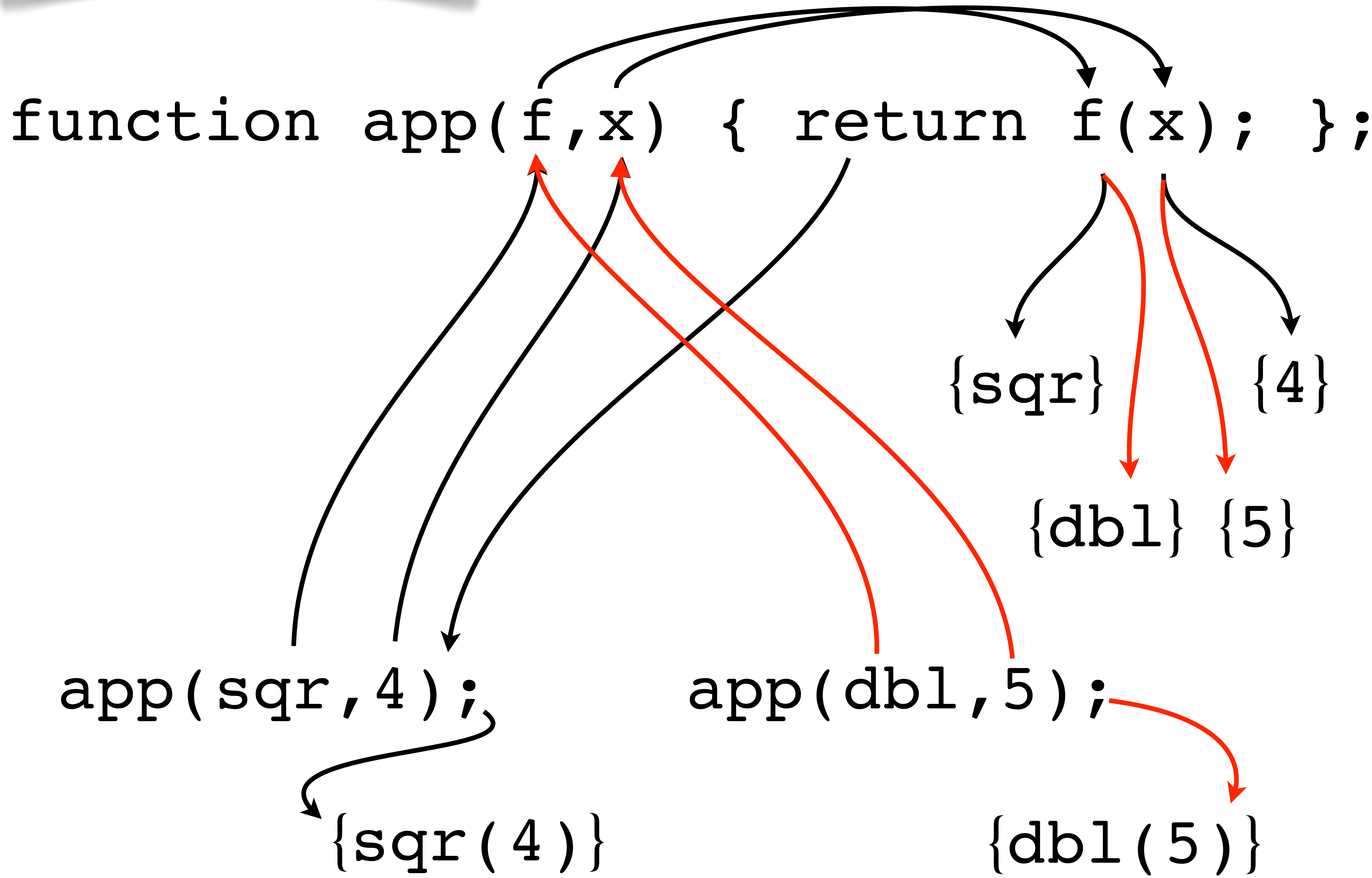
{sqr} {4}

{dbl} {5}

```
app(sqr,4);        app(dbl,5);
```

{sqr(4)} {dbl(5)}

1CFA


function app(f,x) { return f(x); };
{sqr}
{4}
{dbl} {5}
app(sqr,4);
app(dbl,5);
{sqr(4)}
{dbl(5)}

1CFA

ICFP'08

```
function app(f,x) { return f(x); };
```

Theorem: 1CFA is complete for EXPTIME.

```
app(sqr,4);          app(dbl,5);

    {sqr(4)}              {dbl(5)}
```

Precision

1CFA

0CFA

Simple closure

Sub0CFA

⋮

Precision

kCFA

$\vdots$

1CFA

0CFA

Simple closure

Sub0CFA

$\vdots$

*k*CFA

```
function app(f,x) { return f(x); };
```

{sqr} {4}

{dbl} {5}

```
app(sqr,4);        app(dbl,5);
```

{sqr(4)} {dbl(5)}

*k*CFA

ICFP'08

```
function app(f,x) { return f(x); };
```

Theorem: *k*CFA is complete for EXPTIME.

```
app(sqr,4);        app(dbl,5);
```

{`sqr(4)`}   {`dbl(5)`}

Precision

kCFA

$\vdots$

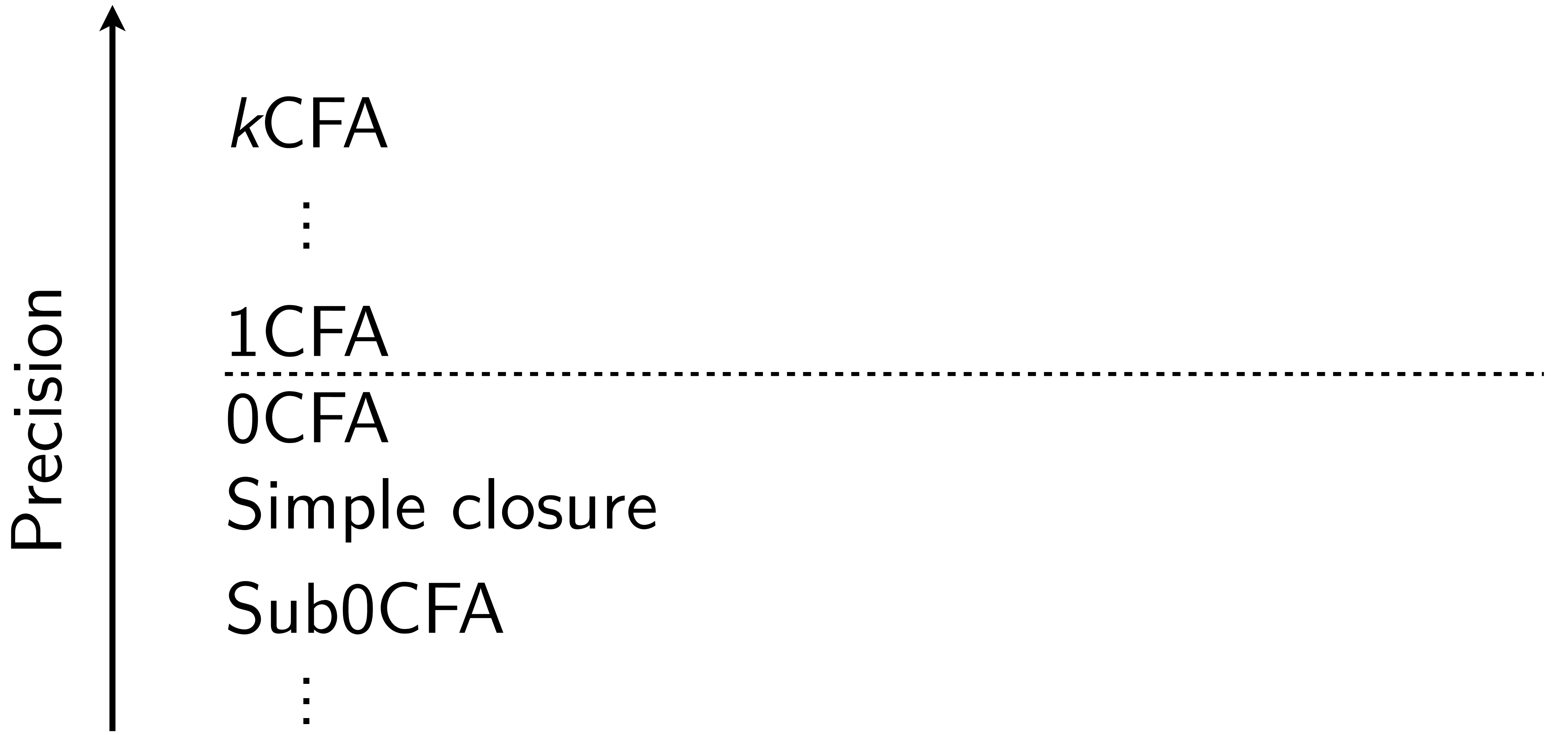
1CFA

0CFA

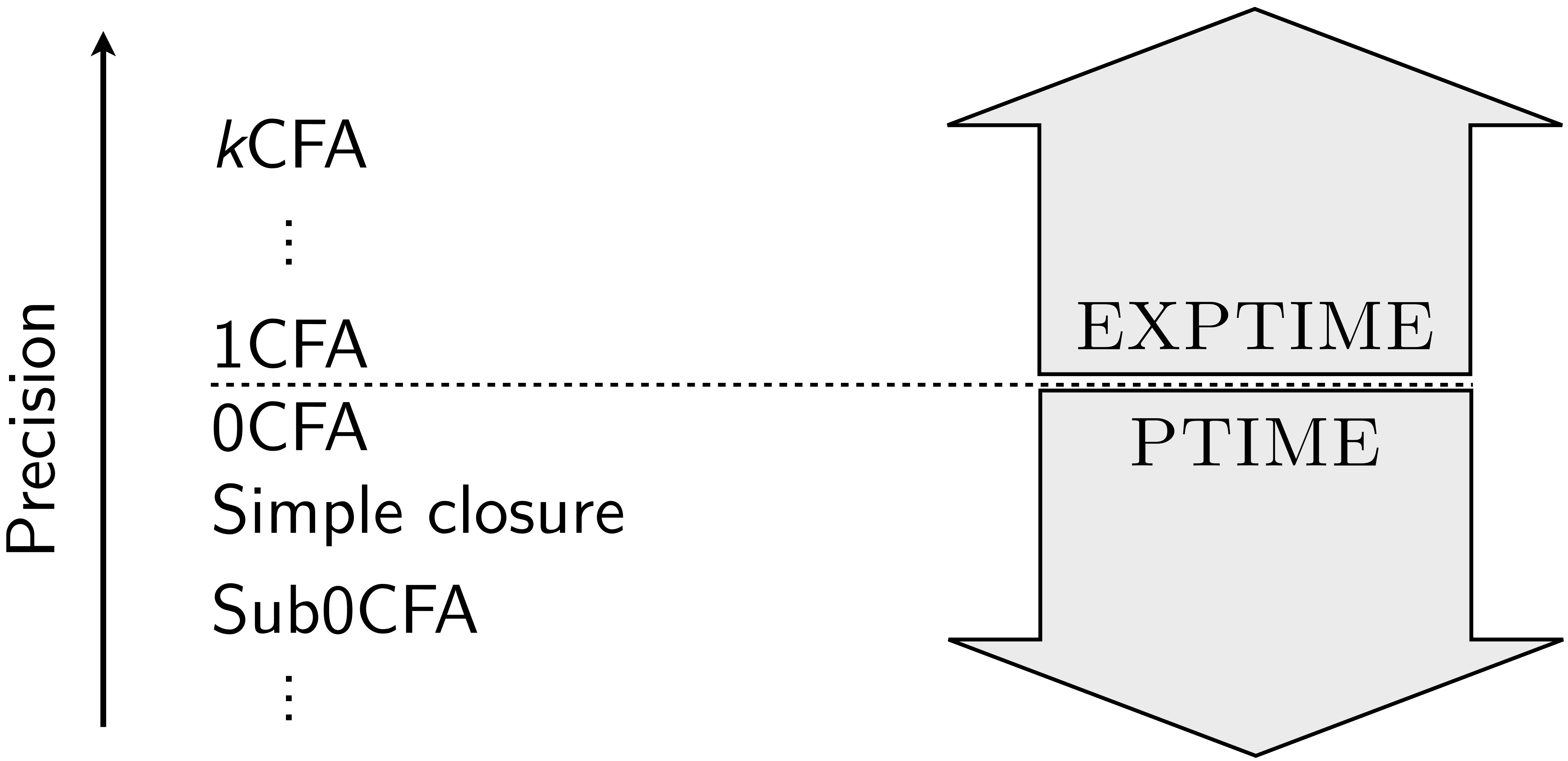
Simple closure

Sub0CFA

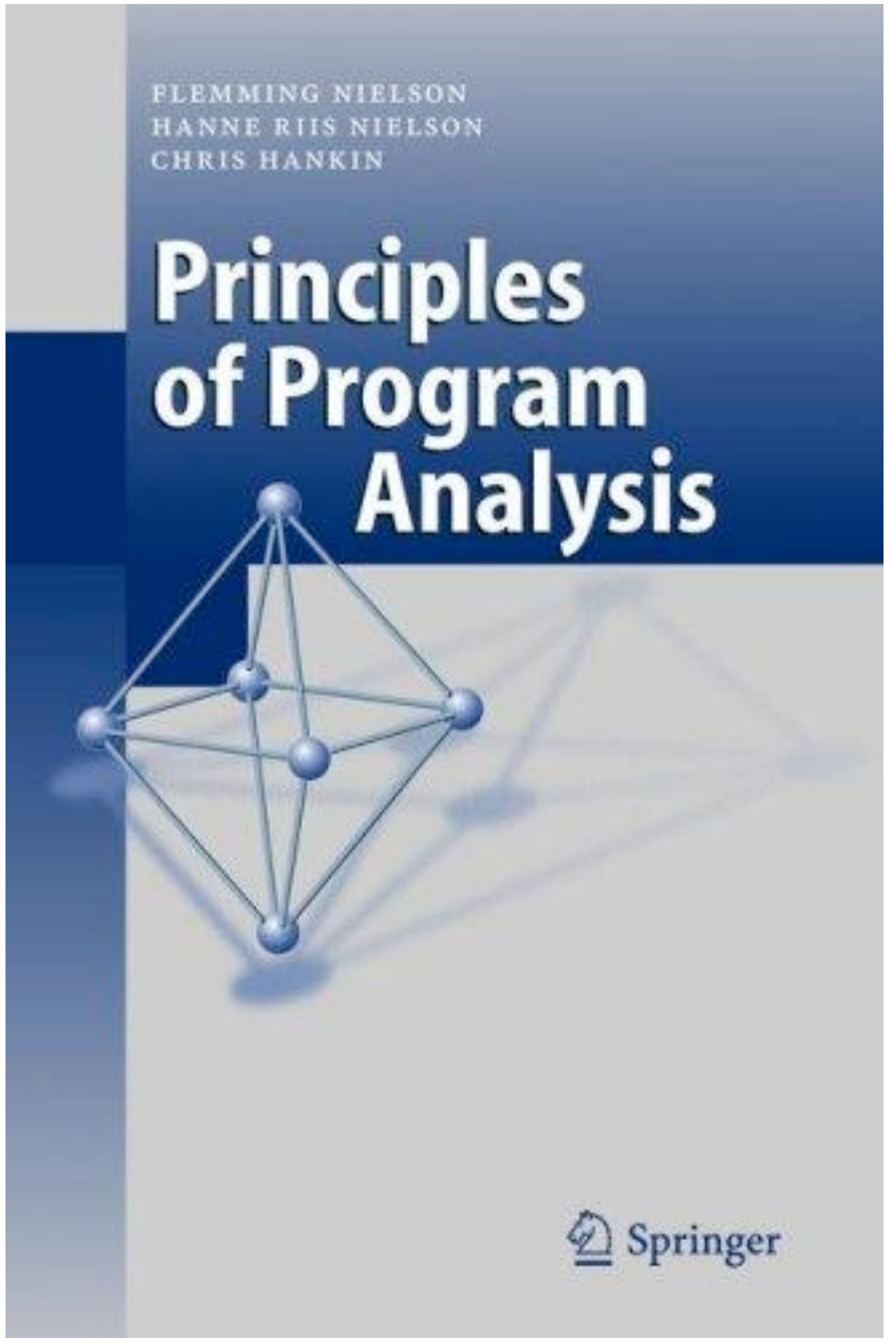
$\vdots$



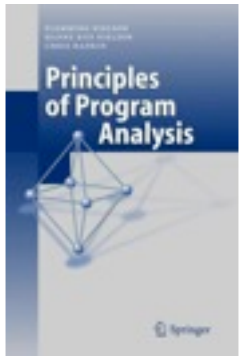
Precision
*k*CFA
⋮
1CFA
0CFA
Simple closure
Sub0CFA
⋮


Precision
kCFA
⋮
1CFA
0CFA
Simple closure
Sub0CFA
⋮
EXPTIME
PTIME

Rigor (mortis) of existing analyses


FLEMMING NIELSON
HANNE RIIS NIELSON
CHRIS HANKIN
Principles of Program Analysis
Springer

the Semantic Gap



[*con*] $(\widehat{\mathsf{C}},\widehat{\rho}) \models c^\ell$ always

[*var*] $(\widehat{\mathsf{C}},\widehat{\rho}) \models x^\ell$ iff $\widehat{\rho}(x) \subseteq \widehat{\mathsf{C}}(\ell)$

[*fn*] $(\widehat{\mathsf{C}},\widehat{\rho}) \models (\texttt{fn}\ x\ \texttt{=>}\ e_0)^\ell$ iff $\{\texttt{fn}\ x\ \texttt{=>}\ e_0\} \subseteq \widehat{\mathsf{C}}(\ell)$

[*fun*] $(\widehat{\mathsf{C}},\widehat{\rho}) \models (\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)^\ell$ iff $\{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq \widehat{\mathsf{C}}(\ell)$

[*app*]
$$
\begin{aligned}
&(\widehat{\mathsf{C}},\widehat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^\ell \\
&\quad \text{iff}\ \ (\widehat{\mathsf{C}},\widehat{\rho}) \models t_1^{\ell_1}\ \wedge\ (\widehat{\mathsf{C}},\widehat{\rho}) \models t_2^{\ell_2}\ \wedge \\
&\qquad\ \ (\forall (\texttt{fn}\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1) : \\
&\qquad\qquad (\widehat{\mathsf{C}},\widehat{\rho}) \models t_0^{\ell_0}\ \wedge \\
&\qquad\qquad \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\rho}(x)\ \wedge\ \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell))\ \wedge \\
&\qquad\ \ (\forall (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1) : \\
&\qquad\qquad (\widehat{\mathsf{C}},\widehat{\rho}) \models t_0^{\ell_0}\ \wedge \\
&\qquad\qquad \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\rho}(x)\ \wedge\ \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell)\ \wedge \\
&\qquad\qquad \{\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}\} \subseteq \widehat{\rho}(f))
\end{aligned}
$$

[*if*]
$$
\begin{aligned}
&(\widehat{\mathsf{C}},\widehat{\rho}) \models (\texttt{if}\ t_0^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell \\
&\quad \text{iff}\ \ (\widehat{\mathsf{C}},\widehat{\rho}) \models t_0^{\ell_0}\ \wedge \\
&\qquad\ \ (\widehat{\mathsf{C}},\widehat{\rho}) \models t_1^{\ell_1}\ \wedge\ (\widehat{\mathsf{C}},\widehat{\rho}) \models t_2^{\ell_2} \wedge \\
&\qquad\ \ \widehat{\mathsf{C}}(\ell_1) \subseteq \widehat{\mathsf{C}}(\ell)\ \wedge\ \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(\ell)
\end{aligned}
$$

[*let*]
$$
\begin{aligned}
&(\widehat{\mathsf{C}},\widehat{\rho}) \models (\texttt{let}\ x = t_1^{\ell_1}\ \texttt{in}\ t_2^{\ell_2})^\ell \\
&\quad \text{iff}\ \ (\widehat{\mathsf{C}},\widehat{\rho}) \models t_1^{\ell_1}\ \wedge\ (\widehat{\mathsf{C}},\widehat{\rho}) \models t_2^{\ell_2}\ \wedge \\
&\qquad\ \ \widehat{\mathsf{C}}(\ell_1) \subseteq \widehat{\rho}(x)\ \wedge\ \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(\ell)
\end{aligned}
$$

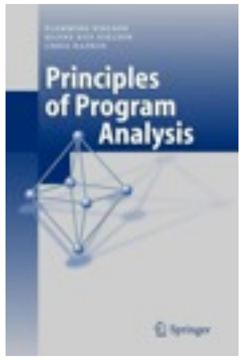
[*op*] $(\widehat{\mathsf{C}},\widehat{\rho}) \models (t_1^{\ell_1}\ op\ t_2^{\ell_2})^\ell$ iff $(\widehat{\mathsf{C}},\widehat{\rho}) \models t_1^{\ell_1}\ \wedge\ (\widehat{\mathsf{C}},\widehat{\rho}) \models t_2^{\ell_2}$

Table 3.1: Abstract Control Flow Analysis (Subsections 3.1.1 and 3.1.2).

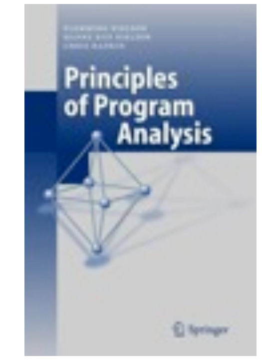
the Semantic Gap

$$
\begin{array}{ll}
[con] & (\hat{C},\hat{\rho}) \models c^{\ell} \text{ always} \\
[var] & (\hat{C},\hat{\rho}) \models x^{\ell} \text{ iff } \hat{\rho}(x) \subseteq \hat{C}(\ell) \\
[fn] & (\hat{C},\hat{\rho}) \models (\texttt{fn}\ x \texttt{ => } e_0)^{\ell} \text{ iff } \{\texttt{fn}\ x \texttt{ => } e_0\} \subseteq \hat{C}(\ell) \\
[fun] & (\hat{C},\hat{\rho}) \models (\texttt{fun}\ f\ x \texttt{ => } e_0)^{\ell} \text{ iff } \{\texttt{fun}\ f\ x \texttt{ => } e_0\} \subseteq \hat{C}(\ell) \\
[app] & (\hat{C},\hat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell} \\
 & \quad \text{iff } (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2} \wedge \\
 & \qquad (\forall (\texttt{fn}\ x \texttt{ => } t_0^{\ell_0}) \in \hat{C}(\ell_1) : \\
 & \qquad\qquad (\hat{C},\hat{\rho}) \models t_0^{\ell_0} \wedge \\
 & \qquad\qquad \hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell)) \wedge \\
 & \qquad (\forall (\texttt{fun}\ f\ x \texttt{ => } t_0^{\ell_0}) \in \hat{C}(\ell_1) : \\
 & \qquad\qquad (\hat{C},\hat{\rho}) \models t_0^{\ell_0} \wedge \\
 & \qquad\qquad \hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell) \wedge \\
 & \qquad\qquad \{\texttt{fun}\ f\ x \texttt{ => } t_0^{\ell_0}\} \subseteq \hat{\rho}(f)) \\
[if] & (\hat{C},\hat{\rho}) \models (\texttt{if}\ t_0^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^{\ell} \\
 & \quad \text{iff } (\hat{C},\hat{\rho}) \models t_0^{\ell_0} \wedge \\
 & \qquad (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2} \wedge \\
 & \qquad \hat{C}(\ell_1) \subseteq \hat{C}(\ell) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell) \\
[let] & (\hat{C},\hat{\rho}) \models (\texttt{let}\ x = t_1^{\ell_1} \texttt{ in } t_2^{\ell_2})^{\ell} \\
 & \quad \text{iff } (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2} \wedge \\
 & \qquad \hat{C}(\ell_1) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell) \\
[op] & (\hat{C},\hat{\rho}) \models (t_1^{\ell_1}\ op\ t_2^{\ell_2})^{\ell} \text{ iff } (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2}
\end{array}
$$

Table 3.1: Abstract Control Flow Analysis (Subsections 3.1.1 and 3.1.2).



the Semantic Gap



$$
\begin{array}{ll}
[con] & (\widehat{C},\widehat{\rho}) \models c^{\ell} \text{ always} \\
[var] & (\widehat{C},\widehat{\rho}) \models x^{\ell} \text{ iff } \widehat{\rho}(x) \subseteq \widehat{C}(\ell) \\
[fn] & (\widehat{C},\widehat{\rho}) \models (\texttt{fn } x \texttt{ => } e_0)^{\ell} \text{ iff } \{\texttt{fn } x \texttt{ => } e_0\} \subseteq \widehat{C}(\ell) \\
[fun] & (\widehat{C},\widehat{\rho}) \models (\texttt{fun } f\ x \texttt{ => } e_0)^{\ell} \text{ iff } \{\texttt{fun } f\ x \texttt{ => } e_0\} \subseteq \widehat{C}(\ell) \\
[app] & (\widehat{C},\widehat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell} \\
& \quad \text{iff } (\widehat{C},\widehat{\rho}) \models t_1^{\ell_1} \land (\widehat{C},\widehat{\rho}) \models t_2^{\ell_2} \land \\
& \qquad (\forall (\texttt{fn } x \texttt{ => } t_0^{\ell_0}) \in \widehat{C}(\ell_1): \\
& \qquad\qquad (\widehat{C},\widehat{\rho}) \models t_0^{\ell_0} \land \\
& \qquad\qquad \widehat{C}(\ell_2) \subseteq \widehat{\rho}(x) \land \widehat{C}(\ell_0) \subseteq \widehat{C}(\ell)) \land \\
& \qquad (\forall (\texttt{fun } f\ x \texttt{ => } t_0^{\ell_0}) \in \widehat{C}(\ell_1): \\
& \qquad\qquad (\widehat{C},\widehat{\rho}) \models t_0^{\ell_0} \land \\
& \qquad\qquad \widehat{C}(\ell_2) \subseteq \widehat{\rho}(x) \land \widehat{C}(\ell_0) \subseteq \widehat{C}(\ell) \land \\
& \qquad\qquad \{\texttt{fun } f\ x \texttt{ => } t_0^{\ell_0}\} \subseteq \widehat{\rho}(f)) \\
[if] & (\widehat{C},\widehat{\rho}) \models (\texttt{if } t_0^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^{\ell} \\
& \quad \text{iff } (\widehat{C},\widehat{\rho}) \models t_0^{\ell_0} \land \\
& \qquad (\widehat{C},\widehat{\rho}) \models t_1^{\ell_1} \land (\widehat{C},\widehat{\rho}) \models t_2^{\ell_2} \land \\
& \qquad \widehat{C}(\ell_1) \subseteq \widehat{C}(\ell) \land \widehat{C}(\ell_2) \subseteq \widehat{C}(\ell) \\
[let] & (\widehat{C},\widehat{\rho}) \models (\texttt{let } x = t_1^{\ell_1} \texttt{ in } t_2^{\ell_2})^{\ell} \\
& \quad \text{iff } (\widehat{C},\widehat{\rho}) \models t_1^{\ell_1} \land (\widehat{C},\widehat{\rho}) \models t_2^{\ell_2} \land \\
& \qquad \widehat{C}(\ell_1) \subseteq \widehat{\rho}(x) \land \widehat{C}(\ell_2) \subseteq \widehat{C}(\ell) \\
[op] & (\widehat{C},\widehat{\rho}) \models (t_1^{\ell_1}\ op\ t_2^{\ell_2})^{\ell} \text{ iff } (\widehat{C},\widehat{\rho}) \models t_1^{\ell_1} \land (\widehat{C},\widehat{\rho}) \models t_2^{\ell_2}
\end{array}
$$

Table 3.1: Abstract Control Flow Analysis (Subsections 3.1.1 and 3.1.2).

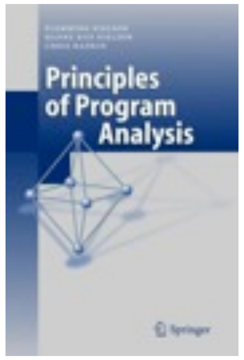
$$
\begin{array}{ll}
[var] & \rho \vdash x^{\ell} \to v^{\ell} \quad \text{if } x \in dom(\rho) \text{ and } v = \rho(x) \\
[fn] & \rho \vdash (\texttt{fn } x \texttt{ => } e_0)^{\ell} \to (\texttt{close } (\texttt{fn } x \texttt{ => } e_0) \texttt{ in } \rho_0)^{\ell} \\
& \qquad\qquad \text{where } \rho_0 = \rho \mid FV(\texttt{fn } x \texttt{ => } e_0) \\
[fun] & \rho \vdash (\texttt{fun } f\ x \texttt{ => } e_0)^{\ell} \to (\texttt{close } (\texttt{fun } f\ x \texttt{ => } e_0) \texttt{ in } \rho_0)^{\ell} \\
& \qquad\qquad \text{where } \rho_0 = \rho \mid FV(\texttt{fun } f\ x \texttt{ => } e_0) \\
[app_1] & \dfrac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (ie_1\ ie_2)^{\ell} \to (ie_1'\ ie_2)^{\ell}} \\
[app_2] & \dfrac{\rho \vdash ie_2 \to ie_2'}{\rho \vdash (v_1^{\ell_1}\ ie_2)^{\ell} \to (v_1^{\ell_1}\ ie_2')^{\ell}} \\
[app_{fn}] & \rho \vdash ((\texttt{close } (\texttt{fn } x \texttt{ => } e_1) \texttt{ in } \rho_1)^{\ell_1}\ v_2^{\ell_2})^{\ell} \to \\
& \qquad\qquad (\texttt{bind } \rho_1[x \mapsto v_2] \texttt{ in } e_1)^{\ell} \\
[app_{fun}] & \rho \vdash ((\texttt{close } (\texttt{fun } f\ x \texttt{ => } e_1) \texttt{ in } \rho_1)^{\ell_1}\ v_2^{\ell_2})^{\ell} \to \\
& \qquad\qquad (\texttt{bind } \rho_2[x \mapsto v_2] \texttt{ in } e_1)^{\ell} \\
& \qquad\qquad \text{where } \rho_2 = \rho_1[f \mapsto \texttt{close } (\texttt{fun } f\ x \texttt{ => } e_1) \texttt{ in } \rho_1] \\
[bind_1] & \dfrac{\rho_1 \vdash ie_1 \to ie_1'}{\rho \vdash (\texttt{bind } \rho_1 \texttt{ in } ie_1)^{\ell} \to (\texttt{bind } \rho_1 \texttt{ in } ie_1')^{\ell}} \\
[bind_2] & \rho \vdash (\texttt{bind } \rho_1 \texttt{ in } v_1^{\ell_1})^{\ell} \to v_1^{\ell}
\end{array}
$$

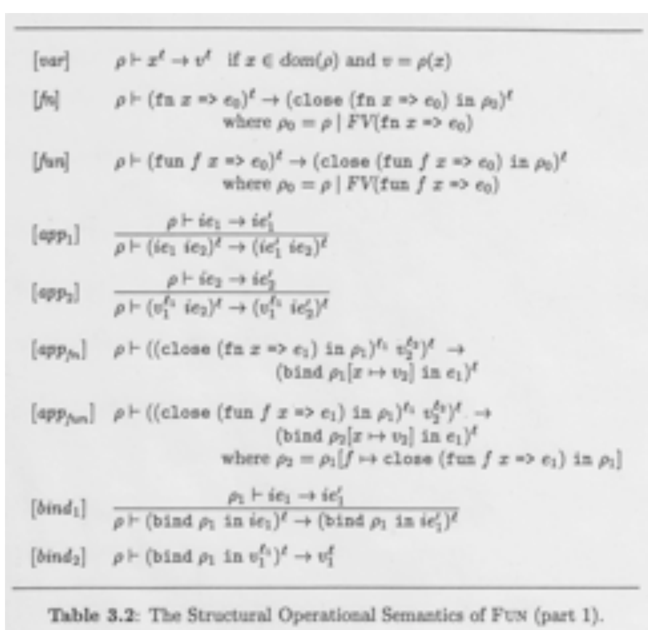
Table 3.2: The Structural Operational Semantics of FUN (part 1).

the Semantic Gap

[con]	$(\widehat{\mathsf{C}}, \widehat{\rho}) \models c^\ell$ always
[var]	$(\widehat{\mathsf{C}}, \widehat{\rho}) \models x^\ell$ iff $\widehat{\rho}(x) \subseteq \widehat{\mathsf{C}}(\ell)$
[fn]	$(\widehat{\mathsf{C}}, \widehat{\rho}) \models (\texttt{fn}\ x \texttt{ => } e_0)^\ell$ iff $\{\texttt{fn}\ x \texttt{ => } e_0\} \subseteq \widehat{\mathsf{C}}(\ell)$
[fun]	$(\widehat{\mathsf{C}}, \widehat{\rho}) \models (\texttt{fun}\ f\ x \texttt{ => } e_0)^\ell$ iff $\{\texttt{fun}\ f\ x \texttt{ => } e_0\} \subseteq \widehat{\mathsf{C}}(\ell)$
[app]	$(\widehat{\mathsf{C}}, \widehat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^\ell$ iff $(\widehat{\mathsf{C}}, \widehat{\rho}) \models t_1^{\ell_1} \wedge (\widehat{\mathsf{C}}, \widehat{\rho}) \models t_2^{\ell_2} \wedge$ $(\forall (\texttt{fn}\ x \texttt{ => } t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1):$ $(\widehat{\mathsf{C}}, \widehat{\rho}) \models t_0^{\ell_0} \wedge$ $\widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\rho}(x) \wedge \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell)) \wedge$ $(\forall (\texttt{fun}\ f\ x \texttt{ => } t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1):$ $(\widehat{\mathsf{C}}, \widehat{\rho}) \models t_0^{\ell_0} \wedge$ $\widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\rho}(x) \wedge \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell) \wedge$ $\{\texttt{fun}\ f\ x \texttt{ => } t_0^{\ell_0}\} \subseteq \widehat{\rho}(f))$
[if]	$(\widehat{\mathsf{C}}, \widehat{\rho}) \models (\texttt{if}\ t_0^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell$ iff $(\widehat{\mathsf{C}}, \widehat{\rho}) \models t_0^{\ell_0} \wedge$ $(\widehat{\mathsf{C}}, \widehat{\rho}) \models t_1^{\ell_1} \wedge (\widehat{\mathsf{C}}, \widehat{\rho}) \models t_2^{\ell_2} \wedge$ $\widehat{\mathsf{C}}(\ell_1) \subseteq \widehat{\mathsf{C}}(\ell) \wedge \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(\ell)$
[let]	$(\widehat{\mathsf{C}}, \widehat{\rho}) \models (\texttt{let}\ x = t_1^{\ell_1}\ \texttt{in}\ t_2^{\ell_2})^\ell$ iff $(\widehat{\mathsf{C}}, \widehat{\rho}) \models t_1^{\ell_1} \wedge (\widehat{\mathsf{C}}, \widehat{\rho}) \models t_2^{\ell_2} \wedge$ $\widehat{\mathsf{C}}(\ell_1) \subseteq \widehat{\rho}(x) \wedge \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(\ell)$
[op]	$(\widehat{\mathsf{C}}, \widehat{\rho}) \models (t_1^{\ell_1}\ op\ t_2^{\ell_2})^\ell$ iff $(\widehat{\mathsf{C}}, \widehat{\rho}) \models t_1^{\ell_1} \wedge (\widehat{\mathsf{C}}, \widehat{\rho}) \models t_2^{\ell_2}$

Table 3.1: Abstract Control Flow Analysis (Subsections 3.1.1 and 3.1.2).





[var]	$\rho \vdash x^\ell \to v^\ell$ if $x \in \mathrm{dom}(\rho)$ and $v = \rho(x)$
[fn]	$\rho \vdash (\texttt{fn}\ x \texttt{ => } e_0)^\ell \to (\texttt{close}\ (\texttt{fn}\ x \texttt{ => } e_0)\ \texttt{in}\ \rho_0)^\ell$ where $\rho_0 = \rho \mid FV(\texttt{fn}\ x \texttt{ => } e_0)$
[fun]	$\rho \vdash (\texttt{fun}\ f\ x \texttt{ => } e_0)^\ell \to (\texttt{close}\ (\texttt{fun}\ f\ x \texttt{ => } e_0)\ \texttt{in}\ \rho_0)^\ell$ where $\rho_0 = \rho \mid FV(\texttt{fun}\ f\ x \texttt{ => } e_0)$
[app_1]	$\dfrac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (ie_1\ ie_2)^\ell \to (ie_1'\ ie_2)^\ell}$
[app_2]	$\dfrac{\rho \vdash ie_2 \to ie_2'}{\rho \vdash (v_1^{\ell_1}\ ie_2)^\ell \to (v_1^{\ell_1}\ ie_2')^\ell}$
[app_{fn}]	$\rho \vdash ((\texttt{close}\ (\texttt{fn}\ x \texttt{ => } e_1)\ \texttt{in}\ \rho_1)^{\ell_1}\ v_2^{\ell_2})^\ell \to$ $(\texttt{bind}\ \rho_1[x \mapsto v_2]\ \texttt{in}\ e_1)^\ell$
[app_{fun}]	$\rho \vdash ((\texttt{close}\ (\texttt{fun}\ f\ x \texttt{ => } e_1)\ \texttt{in}\ \rho_1)^{\ell_1}\ v_2^{\ell_2})^\ell \to$ $(\texttt{bind}\ \rho_2[x \mapsto v_2]\ \texttt{in}\ e_1)^\ell$ where $\rho_2 = \rho_1[f \mapsto \texttt{close}\ (\texttt{fun}\ f\ x \texttt{ => } e_1)\ \texttt{in}\ \rho_1]$
[$bind_1$]	$\dfrac{\rho_1 \vdash ie_1 \to ie_1'}{\rho \vdash (\texttt{bind}\ \rho_1\ \texttt{in}\ ie_1)^\ell \to (\texttt{bind}\ \rho_1\ \texttt{in}\ ie_1')^\ell}$
[$bind_2$]	$\rho \vdash (\texttt{bind}\ \rho_1\ \texttt{in}\ v_1^{\ell_1})^\ell \to v_1^\ell$

Table 3.2: The Structural Operational Semantics of FUN (part 1).

the Semantic Gap

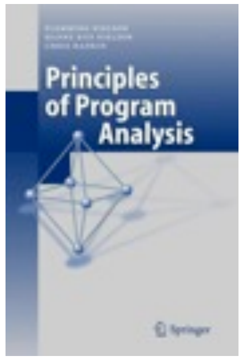


Table 3.1: Abstract Control Flow Analysis (Subsections 3.1.1 a

$$[if_1] \quad \frac{\rho \vdash ie_0 \rightarrow ie'_0}{\rho \vdash (\texttt{if}\ ie_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2)^\ell \rightarrow (\texttt{if}\ ie'_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2)^\ell}$$

$$[if_2] \quad \rho \vdash (\texttt{if}\ \texttt{true}^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell \rightarrow t_1^\ell$$

$$[if_3] \quad \rho \vdash (\texttt{if}\ \texttt{false}^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell \rightarrow t_2^\ell$$

$$[let_1] \quad \frac{\rho \vdash ie_1 \rightarrow ie'_1}{\rho \vdash (\texttt{let}\ x = ie_1\ \texttt{in}\ e_2)^\ell \rightarrow (\texttt{let}\ x = ie'_1\ \texttt{in}\ e_2)^\ell}$$

$$[let_2] \quad \rho \vdash (\texttt{let}\ x = v^{\ell_1}\ \texttt{in}\ e_2)^\ell \rightarrow (\texttt{bind}\ \rho_0[x \mapsto v]\ \texttt{in}\ e_2)^\ell \quad \text{where } \rho_0 = \rho \mid FV(e_2)$$

$$[op_1] \quad \frac{\rho \vdash ie_1 \rightarrow ie'_1}{\rho \vdash (ie_1\ op\ ie_2)^\ell \rightarrow (ie'_1\ op\ ie_2)^\ell}$$

$$[op_2] \quad \frac{\rho \vdash ie_2 \rightarrow ie'_2}{\rho \vdash (v_1^{\ell_1}\ op\ ie_2)^\ell \rightarrow (v_1^{\ell_1}\ op\ ie'_2)^\ell}$$

$$[op_3] \quad \rho \vdash (v_1^{\ell_1}\ op\ v_2^{\ell_2})^\ell \rightarrow v^\ell \qquad \text{if } v = v_1\ \mathbf{op}\ v_2$$

Table 3.3: The Structural Operational Semantics of FUN (part 2).

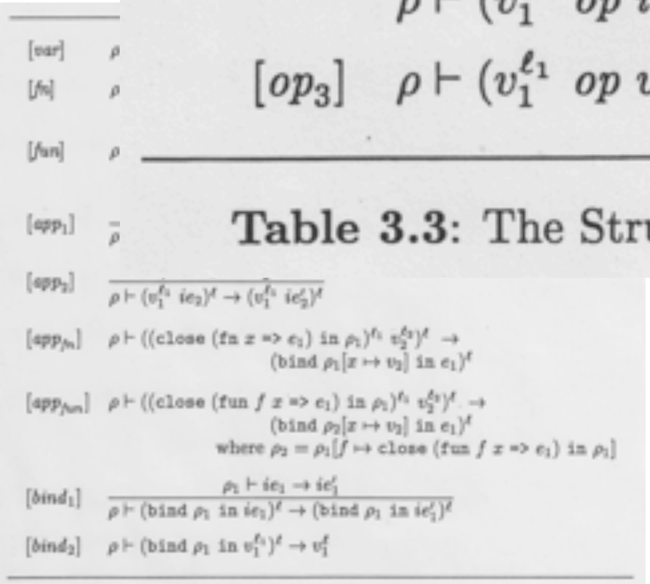
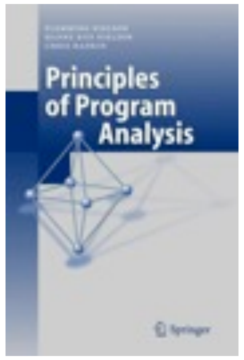


Table 3.2: The Structural Operational Semantics of FUN (part 1).

the Semantic Gap



$$[con]\quad (\hat{C},\hat{\rho}) \models c^\ell \text{ always}$$

$$[var]\quad (\hat{C},\hat{\rho}) \models x^\ell \text{ iff } \hat{\rho}(x) \subseteq \hat{C}(\ell)$$

$$[fn]\quad (\hat{C},\hat{\rho}) \models (\texttt{fn } x \texttt{ => } e_0)^\ell \text{ iff } \{\texttt{fn } x \texttt{ => } e_0\} \subseteq \hat{C}(\ell)$$

$$[fun]\quad (\hat{C},\hat{\rho}) \models (\texttt{fun } f\ x \texttt{ => } e_0)^\ell \text{ iff } \{\texttt{fun } f\ x \texttt{ => } e_0\} \subseteq \hat{C}(\ell)$$

$$\begin{array}{ll}[app] & (\hat{C},\hat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^\ell \\ & \quad \text{iff } (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2} \wedge \\ & \qquad (\forall (\texttt{fn } x \texttt{ => } t_0^{\ell_0}) \in \hat{C}(\ell_1): \\ & \qquad\quad (\hat{C},\hat{\rho}) \models t_0^{\ell_0} \wedge \\ & \qquad\quad \hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell)) \wedge \\ & \qquad (\forall (\texttt{fun } f\ x \texttt{ => } t_0^{\ell_0}) \in \hat{C}(\ell_1): \\ & \qquad\quad (\hat{C},\hat{\rho}) \models t_0^{\ell_0} \wedge \\ & \qquad\quad \hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell) \wedge \\ & \qquad\quad \{\texttt{fun } f\ x \texttt{ => } t_0^{\ell_0}\} \subseteq \hat{\rho}(f))\end{array}$$

$$\begin{array}{ll}[if] & (\hat{C},\hat{\rho}) \models (\texttt{if } t_0^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^\ell \\ & \quad \text{iff } (\hat{C},\hat{\rho}) \models t_0^{\ell_0} \wedge \\ & \qquad (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2} \wedge \\ & \qquad \hat{C}(\ell_1) \subseteq \hat{C}(\ell) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell)\end{array}$$

$$\begin{array}{ll}[let] & (\hat{C},\hat{\rho}) \models (\texttt{let } x \texttt{ = } t_1^{\ell_1} \texttt{ in } t_2^{\ell_2})^\ell \\ & \quad \text{iff } (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2} \wedge \\ & \qquad \hat{C}(\ell_1) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell)\end{array}$$

$$[op]\quad (\hat{C},\hat{\rho}) \models (t_1^{\ell_1}\ op\ t_2^{\ell_2})^\ell \text{ iff } (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2}$$

Table 3.1: Abstract Control Flow Analysis (Subsections 3.1.1 and 3.1.2).

$$[var]\quad \rho \vdash x^\ell \to v^\ell \quad \text{if } x \in dom(\rho) \text{ and } v = \rho(x)$$

$$[fn]\quad \rho \vdash (\texttt{fn } x \texttt{ => } e_0)^\ell \to (\texttt{close } (\texttt{fn } x \texttt{ => } e_0) \texttt{ in } \rho_0)^\ell \quad \text{where } \rho_0 = \rho \mid FV(\texttt{fn } x \texttt{ => } e_0)$$

$$[fun]\quad \rho \vdash (\texttt{fun } f\ x \texttt{ => } e_0)^\ell \to (\texttt{close } (\texttt{fun } f\ x \texttt{ => } e_0) \texttt{ in } \rho_0)^\ell \quad \text{where } \rho_0 = \rho \mid FV(\texttt{fun } f\ x \texttt{ => } e_0)$$

$$[app_1]\quad \frac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (ie_1\ ie_2)^\ell \to (ie_1'\ ie_2)^\ell}$$

$$[app_2]\quad \frac{\rho \vdash ie_2 \to ie_2'}{\rho \vdash (v_1^{\ell_1}\ ie_2)^\ell \to (v_1^{\ell_1}\ ie_2')^\ell}$$

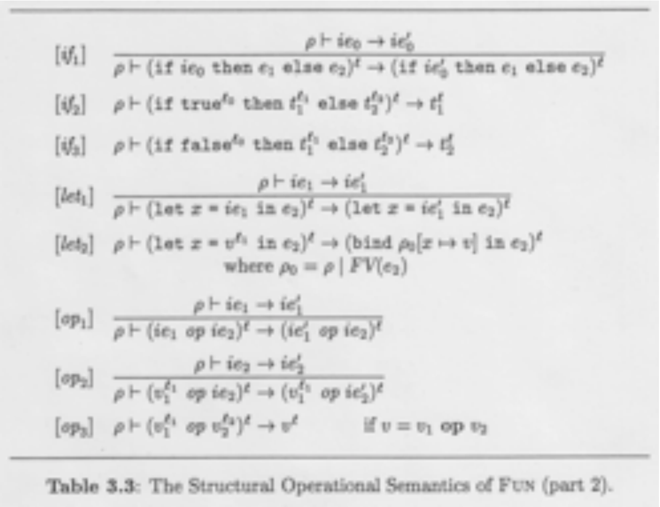
$$[app_{fn}]\quad \rho \vdash ((\texttt{close } (\texttt{fn } x \texttt{ => } e_1) \texttt{ in } \rho_1)^{\ell_1}\ v_2^{\ell_2})^\ell \to (\texttt{bind } \rho_1[x \mapsto v_2] \texttt{ in } e_1)^\ell$$

$$[app_{fun}]\quad \rho \vdash ((\texttt{close } (\texttt{fun } f\ x \texttt{ => } e_1) \texttt{ in } \rho_1)^{\ell_1}\ v_2^{\ell_2})^\ell \to (\texttt{bind } \rho_2[x \mapsto v_2] \texttt{ in } e_1)^\ell \quad \text{where } \rho_2 = \rho_1[f \mapsto \texttt{close } (\texttt{fun } f\ x \texttt{ => } e_1) \texttt{ in } \rho_1]$$

$$[bind_1]\quad \frac{\rho_1 \vdash ie_1 \to ie_1'}{\rho \vdash (\texttt{bind } \rho_1 \texttt{ in } ie_1)^\ell \to (\texttt{bind } \rho_1 \texttt{ in } ie_1')^\ell}$$

$$[bind_2]\quad \rho \vdash (\texttt{bind } \rho_1 \texttt{ in } v_1^{\ell_1})^\ell \to v_1^\ell$$

Table 3.2: The Structural Operational Semantics of FUN (part 1).



$$[if_1]\quad \frac{\rho \vdash ie_0 \to ie_0'}{\rho \vdash (\texttt{if } ie_0 \texttt{ then } e_1 \texttt{ else } e_2)^\ell \to (\texttt{if } ie_0' \texttt{ then } e_1 \texttt{ else } e_2)^\ell}$$

$$[if_2]\quad \rho \vdash (\texttt{if true}^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^\ell \to t_1^\ell$$

$$[if_3]\quad \rho \vdash (\texttt{if false}^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^\ell \to t_2^\ell$$

$$[let_1]\quad \frac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (\texttt{let } x \texttt{ = } ie_1 \texttt{ in } e_2)^\ell \to (\texttt{let } x \texttt{ = } ie_1' \texttt{ in } e_2)^\ell}$$

$$[let_2]\quad \rho \vdash (\texttt{let } x \texttt{ = } v^{\ell_1} \texttt{ in } e_2)^\ell \to (\texttt{bind } \rho_0[x \mapsto v] \texttt{ in } e_2)^\ell \quad \text{where } \rho_0 = \rho \mid FV(e_2)$$

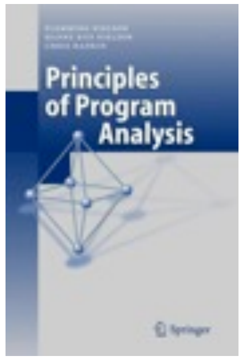
$$[op_1]\quad \frac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (ie_1\ op\ ie_2)^\ell \to (ie_1'\ op\ ie_2)^\ell}$$

$$[op_2]\quad \frac{\rho \vdash ie_2 \to ie_2'}{\rho \vdash (v_1^{\ell_1}\ op\ ie_2)^\ell \to (v_1^{\ell_1}\ op\ ie_2')^\ell}$$

$$[op_3]\quad \rho \vdash (v_1^{\ell_1}\ op\ v_2^{\ell_2})^\ell \to v^\ell \qquad \text{if } v = v_1 \text{ op } v_2$$

Table 3.3: The Structural Operational Semantics of FUN (part 2).

the Semantic Gap



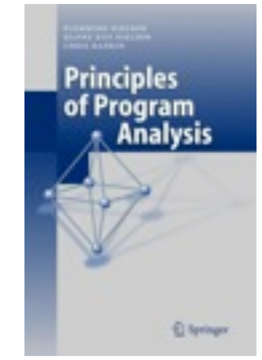
$$
\begin{array}{ll}
[con] & (\widehat{\mathsf{C}},\widehat{\rho}) \models_s c^{\ell} \text{ always} \\[1ex]
[var] & (\widehat{\mathsf{C}},\widehat{\rho}) \models_s x^{\ell} \text{ iff } \widehat{\rho}(x) \subseteq \widehat{\mathsf{C}}(\ell) \\[1ex]
[fn] & (\widehat{\mathsf{C}},\widehat{\rho}) \models_s (\texttt{fn}\ x\ \texttt{=>}\ e_0)^{\ell} \\
 & \quad \text{iff} \quad \{\texttt{fn}\ x\ \texttt{=>}\ e_0\} \subseteq \widehat{\mathsf{C}}(\ell) \wedge \\
 & \qquad\quad (\widehat{\mathsf{C}},\widehat{\rho}) \models_s e_0 \\[1ex]
[fun] & (\widehat{\mathsf{C}},\widehat{\rho}) \models_s (\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)^{\ell} \\
 & \quad \text{iff} \quad \{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq \widehat{\mathsf{C}}(\ell) \wedge \\
 & \qquad\quad (\widehat{\mathsf{C}},\widehat{\rho}) \models_s e_0 \ \wedge\ \{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq \widehat{\rho}(f) \\[1ex]
[app] & (\widehat{\mathsf{C}},\widehat{\rho}) \models_s (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell} \\
 & \quad \text{iff} \quad (\widehat{\mathsf{C}},\widehat{\rho}) \models_s t_1^{\ell_1} \ \wedge\ (\widehat{\mathsf{C}},\widehat{\rho}) \models_s t_2^{\ell_2} \ \wedge \\
 & \qquad\quad (\forall (\texttt{fn}\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1) : \\
 & \qquad\qquad\quad \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\rho}(x) \ \wedge\ \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell)) \ \wedge \\
 & \qquad\quad (\forall (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1) : \\
 & \qquad\qquad\quad \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\rho}(x) \ \wedge\ \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell)) \\[1ex]
[if] & (\widehat{\mathsf{C}},\widehat{\rho}) \models_s (\texttt{if}\ t_0^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^{\ell} \\
 & \quad \text{iff} \quad (\widehat{\mathsf{C}},\widehat{\rho}) \models_s t_0^{\ell_0} \ \wedge \\
 & \qquad\quad (\widehat{\mathsf{C}},\widehat{\rho}) \models_s t_1^{\ell_1} \ \wedge\ (\widehat{\mathsf{C}},\widehat{\rho}) \models_s t_2^{\ell_2} \ \wedge \\
 & \qquad\quad \widehat{\mathsf{C}}(\ell_1) \subseteq \widehat{\mathsf{C}}(\ell) \ \wedge\ \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(\ell) \\[1ex]
[let] & (\widehat{\mathsf{C}},\widehat{\rho}) \models_s (\texttt{let}\ x = t_1^{\ell_1}\ \texttt{in}\ t_2^{\ell_2})^{\ell} \\
 & \quad \text{iff} \quad (\widehat{\mathsf{C}},\widehat{\rho}) \models_s t_1^{\ell_1} \ \wedge\ (\widehat{\mathsf{C}},\widehat{\rho}) \models_s t_2^{\ell_2} \ \wedge \\
 & \qquad\quad \widehat{\mathsf{C}}(\ell_1) \subseteq \widehat{\rho}(x) \ \wedge\ \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(\ell) \\[1ex]
[op] & (\widehat{\mathsf{C}},\widehat{\rho}) \models_s (t_1^{\ell_1}\ op\ t_2^{\ell_2})^{\ell} \ \text{ iff } \ (\widehat{\mathsf{C}},\widehat{\rho}) \models_s t_1^{\ell_1} \ \wedge\ (\widehat{\mathsf{C}},\widehat{\rho}) \models_s t_2^{\ell_2}
\end{array}
$$

Table 3.5: Syntax directed Control Flow Analysis.

Table 3.1: Abstract Control Flow Analysis (Subsections 3.1.1 and 3.1.2).

Table 3.2: The Structural Operational Semantics of FUN (part 1).

the Semantic Gap



$[con]\quad (\widehat{C},\widehat{\rho}) \models c^\ell$ always

$[var]\quad (\widehat{C},\widehat{\rho}) \models x^\ell$ iff $\widehat{\rho}(x) \subseteq \widehat{C}(\ell)$

$[fn]\quad (\widehat{C},\widehat{\rho}) \models (\texttt{fn}\ x \texttt{=>} e_0)^\ell$ iff $\{\texttt{fn}\ x \texttt{=>} e_0\} \subseteq \widehat{C}(\ell)$

$[fun]\quad (\widehat{C},\widehat{\rho}) \models (\texttt{fun}\ f\ x \texttt{=>} e_0)^\ell$ iff $\{\texttt{fun}\ f\ x \texttt{=>} e_0\} \subseteq \widehat{C}(\ell)$

$$[app]\quad \begin{array}{l} (\widehat{C},\widehat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^\ell \\ \quad \text{iff}\ \ (\widehat{C},\widehat{\rho}) \models t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models t_2^{\ell_2} \wedge \\ \qquad (\forall (\texttt{fn}\ x \texttt{=>} t_0^{\ell_0}) \in \widehat{C}(\ell_1) : \\ \qquad\qquad (\widehat{C},\widehat{\rho}) \models t_0^{\ell_0} \wedge \\ \qquad\qquad \widehat{C}(\ell_2) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(\ell_0) \subseteq \widehat{C}(\ell)) \wedge \\ \qquad (\forall (\texttt{fun}\ f\ x \texttt{=>} t_0^{\ell_0}) \in \widehat{C}(\ell_1) : \\ \qquad\qquad (\widehat{C},\widehat{\rho}) \models t_0^{\ell_0} \wedge \\ \qquad\qquad \widehat{C}(\ell_2) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(\ell_0) \subseteq \widehat{C}(\ell) \wedge \\ \qquad\qquad \{\texttt{fun}\ f\ x \texttt{=>} t_0^{\ell_0}\} \subseteq \widehat{\rho}(f)) \end{array}$$

$$[if]\quad \begin{array}{l} (\widehat{C},\widehat{\rho}) \models (\texttt{if}\ t_0^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell \\ \quad \text{iff}\ \ (\widehat{C},\widehat{\rho}) \models t_0^{\ell_0} \wedge \\ \qquad (\widehat{C},\widehat{\rho}) \models t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models t_2^{\ell_2} \wedge \\ \qquad \widehat{C}(\ell_1) \subseteq \widehat{C}(\ell) \wedge \widehat{C}(\ell_2) \subseteq \widehat{C}(\ell) \end{array}$$

$$[let]\quad \begin{array}{l} (\widehat{C},\widehat{\rho}) \models (\texttt{let}\ x = t_1^{\ell_1}\ \texttt{in}\ t_2^{\ell_2})^\ell \\ \quad \text{iff}\ \ (\widehat{C},\widehat{\rho}) \models t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models t_2^{\ell_2} \wedge \\ \qquad \widehat{C}(\ell_1) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(\ell_2) \subseteq \widehat{C}(\ell) \end{array}$$

$[op]\quad (\widehat{C},\widehat{\rho}) \models (t_1^{\ell_1}\ op\ t_2^{\ell_2})^\ell$ iff $(\widehat{C},\widehat{\rho}) \models t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models t_2^{\ell_2}$

Table 3.1: Abstract Control Flow Analysis (Subsections 3.1.1 and 3.1.2).

$[con]\quad (\widehat{C},\widehat{\rho}) \models_s c^\ell$ always

$[var]\quad (\widehat{C},\widehat{\rho}) \models_s x^\ell$ iff $\widehat{\rho}(x) \subseteq \widehat{C}(\ell)$

$$[fn]\quad \begin{array}{l} (\widehat{C},\widehat{\rho}) \models_s (\texttt{fn}\ x \texttt{=>} e_0)^\ell \\ \quad \text{iff}\ \ \{\texttt{fn}\ x \texttt{=>} e_0\} \subseteq \widehat{C}(\ell) \wedge \\ \qquad (\widehat{C},\widehat{\rho}) \models_s e_0 \end{array}$$

$$[fun]\quad \begin{array}{l} (\widehat{C},\widehat{\rho}) \models_s (\texttt{fun}\ f\ x \texttt{=>} e_0)^\ell \\ \quad \text{iff}\ \ \{\texttt{fun}\ f\ x \texttt{=>} e_0\} \subseteq \widehat{C}(\ell) \wedge \\ \qquad (\widehat{C},\widehat{\rho}) \models_s e_0 \wedge \{\texttt{fun}\ f\ x \texttt{=>} e_0\} \subseteq \widehat{\rho}(f) \end{array}$$

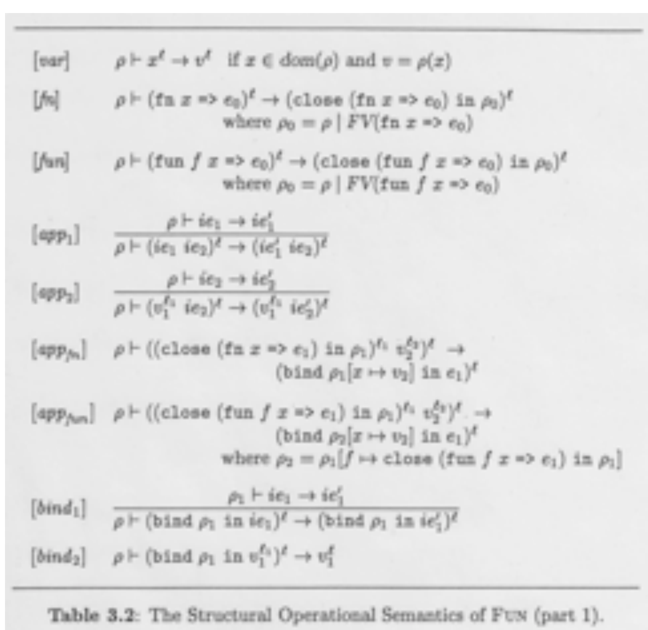
$$[app]\quad \begin{array}{l} (\widehat{C},\widehat{\rho}) \models_s (t_1^{\ell_1}\ t_2^{\ell_2})^\ell \\ \quad \text{iff}\ \ (\widehat{C},\widehat{\rho}) \models_s t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models_s t_2^{\ell_2} \wedge \\ \qquad (\forall (\texttt{fn}\ x \texttt{=>} t_0^{\ell_0}) \in \widehat{C}(\ell_1) : \\ \qquad\qquad \widehat{C}(\ell_2) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(\ell_0) \subseteq \widehat{C}(\ell)) \wedge \\ \qquad (\forall (\texttt{fun}\ f\ x \texttt{=>} t_0^{\ell_0}) \in \widehat{C}(\ell_1) : \\ \qquad\qquad \widehat{C}(\ell_2) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(\ell_0) \subseteq \widehat{C}(\ell)) \end{array}$$

$$[if]\quad \begin{array}{l} (\widehat{C},\widehat{\rho}) \models_s (\texttt{if}\ t_0^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell \\ \quad \text{iff}\ \ (\widehat{C},\widehat{\rho}) \models_s t_0^{\ell_0} \wedge \\ \qquad (\widehat{C},\widehat{\rho}) \models_s t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models_s t_2^{\ell_2} \wedge \\ \qquad \widehat{C}(\ell_1) \subseteq \widehat{C}(\ell) \wedge \widehat{C}(\ell_2) \subseteq \widehat{C}(\ell) \end{array}$$

$$[let]\quad \begin{array}{l} (\widehat{C},\widehat{\rho}) \models_s (\texttt{let}\ x = t_1^{\ell_1}\ \texttt{in}\ t_2^{\ell_2})^\ell \\ \quad \text{iff}\ \ (\widehat{C},\widehat{\rho}) \models_s t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models_s t_2^{\ell_2} \wedge \\ \qquad \widehat{C}(\ell_1) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(\ell_2) \subseteq \widehat{C}(\ell) \end{array}$$

$[op]\quad (\widehat{C},\widehat{\rho}) \models_s (t_1^{\ell_1}\ op\ t_2^{\ell_2})^\ell$ iff $(\widehat{C},\widehat{\rho}) \models_s t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models_s t_2^{\ell_2}$

Table 3.5: Syntax directed Control Flow Analysis.



$[var]\quad \rho \vdash x^\ell \rightarrow v^\ell$ if $x \in dom(\rho)$ and $v = \rho(x)$

$$[fn]\quad \begin{array}{l} \rho \vdash (\texttt{fn}\ x \texttt{=>} e_0)^\ell \rightarrow (\texttt{close}\ (\texttt{fn}\ x \texttt{=>} e_0)\ \texttt{in}\ \rho_0)^\ell \\ \qquad \text{where}\ \rho_0 = \rho \mid FV(\texttt{fn}\ x \texttt{=>} e_0) \end{array}$$

$$[fun]\quad \begin{array}{l} \rho \vdash (\texttt{fun}\ f\ x \texttt{=>} e_0)^\ell \rightarrow (\texttt{close}\ (\texttt{fun}\ f\ x \texttt{=>} e_0)\ \texttt{in}\ \rho_0)^\ell \\ \qquad \text{where}\ \rho_0 = \rho \mid FV(\texttt{fun}\ f\ x \texttt{=>} e_0) \end{array}$$

$$[app_1]\quad \frac{\rho \vdash ie_1 \rightarrow ie_1'}{\rho \vdash (ie_1\ ie_2)^\ell \rightarrow (ie_1'\ ie_2)^\ell}$$

$$[app_2]\quad \frac{\rho \vdash ie_2 \rightarrow ie_2'}{\rho \vdash (v_1^{\ell_1}\ ie_2)^\ell \rightarrow (v_1^{\ell_1}\ ie_2')^\ell}$$

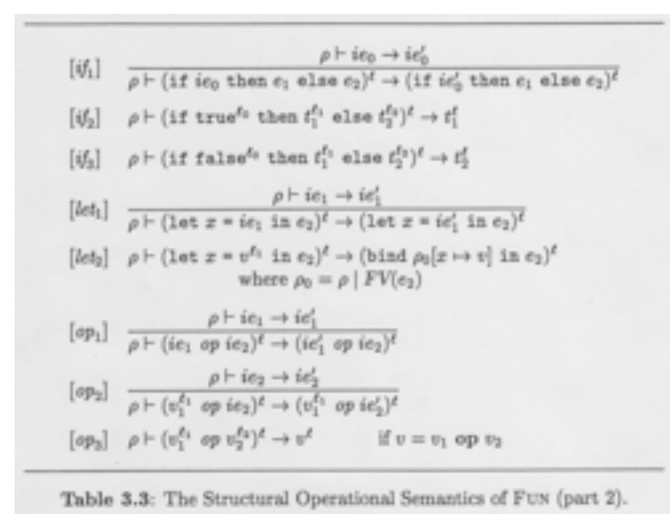
$$[app_{fn}]\quad \begin{array}{l} \rho \vdash ((\texttt{close}\ (\texttt{fn}\ x \texttt{=>} e_1)\ \texttt{in}\ \rho_1)^{\ell_1}\ v_2^{\ell_2})^\ell \rightarrow \\ \qquad (\texttt{bind}\ \rho_1[x \mapsto v_2]\ \texttt{in}\ e_1)^\ell \end{array}$$

$$[app_{fun}]\quad \begin{array}{l} \rho \vdash ((\texttt{close}\ (\texttt{fun}\ f\ x \texttt{=>} e_1)\ \texttt{in}\ \rho_1)^{\ell_1}\ v_2^{\ell_2})^\ell \rightarrow \\ \qquad (\texttt{bind}\ \rho_2[x \mapsto v_2]\ \texttt{in}\ e_1)^\ell \\ \qquad \text{where}\ \rho_2 = \rho_1[f \mapsto \texttt{close}\ (\texttt{fun}\ f\ x \texttt{=>} e_1)\ \texttt{in}\ \rho_1] \end{array}$$

$$[bind_1]\quad \frac{\rho_1 \vdash ie_1 \rightarrow ie_1'}{\rho \vdash (\texttt{bind}\ \rho_1\ \texttt{in}\ ie_1)^\ell \rightarrow (\texttt{bind}\ \rho_1\ \texttt{in}\ ie_1')^\ell}$$

$[bind_2]\quad \rho \vdash (\texttt{bind}\ \rho_1\ \texttt{in}\ v_1^{\ell_1})^\ell \rightarrow v_1^\ell$

Table 3.2: The Structural Operational Semantics of FUN (part 1).



$$[if_1]\quad \frac{\rho \vdash ie_0 \rightarrow ie_0'}{\rho \vdash (\texttt{if}\ ie_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2)^\ell \rightarrow (\texttt{if}\ ie_0'\ \texttt{then}\ e_1\ \texttt{else}\ e_2)^\ell}$$

$[if_2]\quad \rho \vdash (\texttt{if}\ \texttt{true}^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell \rightarrow t_1^\ell$

$[if_3]\quad \rho \vdash (\texttt{if}\ \texttt{false}^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell \rightarrow t_2^\ell$

$$[let_1]\quad \frac{\rho \vdash ie_1 \rightarrow ie_1'}{\rho \vdash (\texttt{let}\ x = ie_1\ \texttt{in}\ e_2)^\ell \rightarrow (\texttt{let}\ x = ie_1'\ \texttt{in}\ e_2)^\ell}$$

$$[let_2]\quad \begin{array}{l} \rho \vdash (\texttt{let}\ x = v^{\ell_1}\ \texttt{in}\ e_2)^\ell \rightarrow (\texttt{bind}\ \rho_0[x \mapsto v]\ \texttt{in}\ e_2)^\ell \\ \qquad \text{where}\ \rho_0 = \rho \mid FV(e_2) \end{array}$$

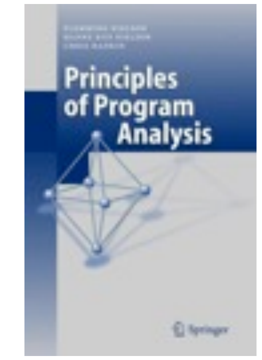
$$[op_1]\quad \frac{\rho \vdash ie_1 \rightarrow ie_1'}{\rho \vdash (ie_1\ op\ ie_2)^\ell \rightarrow (ie_1'\ op\ ie_2)^\ell}$$

$$[op_2]\quad \frac{\rho \vdash ie_2 \rightarrow ie_2'}{\rho \vdash (v_1^{\ell_1}\ op\ ie_2)^\ell \rightarrow (v_1^{\ell_1}\ op\ ie_2')^\ell}$$

$[op_3]\quad \rho \vdash (v_1^{\ell_1}\ op\ v_2^{\ell_2})^\ell \rightarrow v^\ell \qquad$ if $v = v_1\ \mathbf{op}\ v_2$

Table 3.3: The Structural Operational Semantics of FUN (part 2).

the Semantic Gap



$$
\begin{array}{ll}
[con] & (\hat{C},\hat{\rho}) \models c^{\ell} \text{ always} \\
[var] & (\hat{C},\hat{\rho}) \models x^{\ell} \text{ iff } \hat{\rho}(x) \subseteq \hat{C}(\ell) \\
[fn] & (\hat{C},\hat{\rho}) \models (\texttt{fn}\ x \texttt{ => } e_0)^{\ell} \text{ iff } \{\texttt{fn}\ x \texttt{ => } e_0\} \subseteq \hat{C}(\ell) \\
[fun] & (\hat{C},\hat{\rho}) \models (\texttt{fun}\ f\ x \texttt{ => } e_0)^{\ell} \text{ iff } \{\texttt{fun}\ f\ x \texttt{ => } e_0\} \subseteq \hat{C}(\ell) \\
[app] & (\hat{C},\hat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell} \\
& \quad \text{iff } (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2} \wedge \\
& \qquad (\forall (\texttt{fn}\ x \texttt{ => } t_0^{\ell_0}) \in \hat{C}(\ell_1) : \\
& \qquad\qquad (\hat{C},\hat{\rho}) \models t_0^{\ell_0} \wedge \\
& \qquad\qquad \hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell)) \wedge \\
& \qquad (\forall (\texttt{fun}\ f\ x \texttt{ => } t_0^{\ell_0}) \in \hat{C}(\ell_1) : \\
& \qquad\qquad (\hat{C},\hat{\rho}) \models t_0^{\ell_0} \wedge \\
& \qquad\qquad \hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell) \wedge \\
& \qquad\qquad \{\texttt{fun}\ f\ x \texttt{ => } t_0^{\ell_0}\} \subseteq \hat{\rho}(f)) \\
[if] & (\hat{C},\hat{\rho}) \models (\texttt{if}\ t_0^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^{\ell} \\
& \quad \text{iff } (\hat{C},\hat{\rho}) \models t_0^{\ell_0} \wedge \\
& \qquad (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2} \wedge \\
& \qquad \hat{C}(\ell_1) \subseteq \hat{C}(\ell) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell) \\
[let] & (\hat{C},\hat{\rho}) \models (\texttt{let}\ x = t_1^{\ell_1} \texttt{ in } t_2^{\ell_2})^{\ell} \\
& \quad \text{iff } (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2} \wedge \\
& \qquad \hat{C}(\ell_1) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell) \\
[op] & (\hat{C},\hat{\rho}) \models (t_1^{\ell_1}\ op\ t_2^{\ell_2})^{\ell} \text{ iff } (\hat{C},\hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models t_2^{\ell_2}
\end{array}
$$

Table 3.1: Abstract Control Flow Analysis (Subsections 3.1.1 and 3.1.2).

$$
\begin{array}{ll}
[con] & (\hat{C},\hat{\rho}) \models_s c^{\ell} \text{ always} \\
[var] & (\hat{C},\hat{\rho}) \models_s x^{\ell} \text{ iff } \hat{\rho}(x) \subseteq \hat{C}(\ell) \\
[fn] & (\hat{C},\hat{\rho}) \models_s (\texttt{fn}\ x \texttt{ => } e_0)^{\ell} \\
& \quad \text{iff } \{\texttt{fn}\ x \texttt{ => } e_0\} \subseteq \hat{C}(\ell) \wedge \\
& \qquad (\hat{C},\hat{\rho}) \models_s e_0 \\
[fun] & (\hat{C},\hat{\rho}) \models_s (\texttt{fun}\ f\ x \texttt{ => } e_0)^{\ell} \\
& \quad \text{iff } \{\texttt{fun}\ f\ x \texttt{ => } e_0\} \subseteq \hat{C}(\ell) \wedge \\
& \qquad (\hat{C},\hat{\rho}) \models_s e_0 \wedge \{\texttt{fun}\ f\ x \texttt{ => } e_0\} \subseteq \hat{\rho}(f) \\
[app] & (\hat{C},\hat{\rho}) \models_s (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell} \\
& \quad \text{iff } (\hat{C},\hat{\rho}) \models_s t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models_s t_2^{\ell_2} \wedge \\
& \qquad (\forall (\texttt{fn}\ x \texttt{ => } t_0^{\ell_0}) \in \hat{C}(\ell_1) : \\
& \qquad\qquad \hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell)) \wedge \\
& \qquad (\forall (\texttt{fun}\ f\ x \texttt{ => } t_0^{\ell_0}) \in \hat{C}(\ell_1) : \\
& \qquad\qquad \hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell)) \\
[if] & (\hat{C},\hat{\rho}) \models_s (\texttt{if}\ t_0^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^{\ell} \\
& \quad \text{iff } (\hat{C},\hat{\rho}) \models_s t_0^{\ell_0} \wedge \\
& \qquad (\hat{C},\hat{\rho}) \models_s t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models_s t_2^{\ell_2} \wedge \\
& \qquad \hat{C}(\ell_1) \subseteq \hat{C}(\ell) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell) \\
[let] & (\hat{C},\hat{\rho}) \models_s (\texttt{let}\ x = t_1^{\ell_1} \texttt{ in } t_2^{\ell_2})^{\ell} \\
& \quad \text{iff } (\hat{C},\hat{\rho}) \models_s t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models_s t_2^{\ell_2} \wedge \\
& \qquad \hat{C}(\ell_1) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell) \\
[op] & (\hat{C},\hat{\rho}) \models_s (t_1^{\ell_1}\ op\ t_2^{\ell_2})^{\ell} \text{ iff } (\hat{C},\hat{\rho}) \models_s t_1^{\ell_1} \wedge (\hat{C},\hat{\rho}) \models_s t_2^{\ell_2}
\end{array}
$$

Table 3.5: Syntax directed Control Flow Analysis.

$$
\begin{array}{lll}
[con] & \mathcal{C}_*[\![c^{\ell}]\!] = \emptyset & \\
[var] & \mathcal{C}_*[\![x^{\ell}]\!] = \{\mathsf{r}(x) \subseteq \mathsf{C}(\ell)\} & \\
[fn] & \mathcal{C}_*[\![(\texttt{fn}\ x \texttt{ => } e_0)^{\ell}]\!] & = \{\{\texttt{fn}\ x \texttt{ => } e_0\} \subseteq \mathsf{C}(\ell)\} \\
& & \cup\ \mathcal{C}_*[\![e_0]\!] \\
[fun] & \mathcal{C}_*[\![(\texttt{fun}\ f\ x \texttt{ => } e_0)^{\ell}]\!] & = \{\{\texttt{fun}\ f\ x \texttt{ => } e_0\} \subseteq \mathsf{C}(\ell)\} \\
& & \cup\ \mathcal{C}_*[\![e_0]\!] \cup \{\{\texttt{fun}\ f\ x \texttt{ => } e_0\} \subseteq \mathsf{r}(f)\} \\
[app] & \mathcal{C}_*[\![(t_1^{\ell_1}\ t_2^{\ell_2})^{\ell}]\!] & = \mathcal{C}_*[\![t_1^{\ell_1}]\!] \cup \mathcal{C}_*[\![t_2^{\ell_2}]\!] \\
& & \cup\ \{\{t\} \subseteq \mathsf{C}(\ell_1) \Rightarrow \mathsf{C}(\ell_2) \subseteq \mathsf{r}(x) \\
& & \qquad \mid t = (\texttt{fn}\ x \texttt{ => } t_0^{\ell_0}) \in \mathbf{Term}_*\} \\
& & \cup\ \{\{t\} \subseteq \mathsf{C}(\ell_1) \Rightarrow \mathsf{C}(\ell_0) \subseteq \mathsf{C}(\ell) \\
& & \qquad \mid t = (\texttt{fn}\ x \texttt{ => } t_0^{\ell_0}) \in \mathbf{Term}_*\} \\
& & \cup\ \{\{t\} \subseteq \mathsf{C}(\ell_1) \Rightarrow \mathsf{C}(\ell_2) \subseteq \mathsf{r}(x) \\
& & \qquad \mid t = (\texttt{fun}\ f\ x \texttt{ => } t_0^{\ell_0}) \in \mathbf{Term}_*\} \\
& & \cup\ \{\{t\} \subseteq \mathsf{C}(\ell_1) \Rightarrow \mathsf{C}(\ell_0) \subseteq \mathsf{C}(\ell) \\
& & \qquad \mid t = (\texttt{fun}\ f\ x \texttt{ => } t_0^{\ell_0}) \in \mathbf{Term}_*\} \\
[if] & \mathcal{C}_*[\![(\texttt{if}\ t_0^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^{\ell}]\!] & = \mathcal{C}_*[\![t_0^{\ell_0}]\!] \cup \mathcal{C}_*[\![t_1^{\ell_1}]\!] \cup \mathcal{C}_*[\![t_2^{\ell_2}]\!] \\
& & \cup\ \{\mathsf{C}(\ell_1) \subseteq \mathsf{C}(\ell)\} \\
& & \cup\ \{\mathsf{C}(\ell_2) \subseteq \mathsf{C}(\ell)\} \\
[let] & \mathcal{C}_*[\![(\texttt{let}\ x = t_1^{\ell_1} \texttt{ in } t_2^{\ell_2})^{\ell}]\!] & = \mathcal{C}_*[\![t_1^{\ell_1}]\!] \cup \mathcal{C}_*[\![t_2^{\ell_2}]\!] \\
& & \cup\ \{\mathsf{C}(\ell_1) \subseteq \mathsf{r}(x)\} \cup \{\mathsf{C}(\ell_2) \subseteq \mathsf{C}(\ell)\} \\
[op] & \mathcal{C}_*[\![(t_1^{\ell_1}\ op\ t_2^{\ell_2})^{\ell}]\!] & = \mathcal{C}_*[\![t_1^{\ell_1}]\!] \cup \mathcal{C}_*[\![t_2^{\ell_2}]\!]
\end{array}
$$

Table 3.6: Constraint based Control Flow Analysis.

$$
\begin{array}{ll}
[var] & \rho \vdash x^{\ell} \to v^{\ell} \quad \text{if } x \in dom(\rho) \text{ and } v = \rho(x) \\
[fn] & \rho \vdash (\texttt{fn}\ x \texttt{ => } e_0)^{\ell} \to (\texttt{close}\ (\texttt{fn}\ x \texttt{ => } e_0) \texttt{ in } \rho_0)^{\ell} \\
& \qquad \text{where } \rho_0 = \rho \mid FV(\texttt{fn}\ x \texttt{ => } e_0) \\
[fun] & \rho \vdash (\texttt{fun}\ f\ x \texttt{ => } e_0)^{\ell} \to (\texttt{close}\ (\texttt{fun}\ f\ x \texttt{ => } e_0) \texttt{ in } \rho_0)^{\ell} \\
& \qquad \text{where } \rho_0 = \rho \mid FV(\texttt{fun}\ f\ x \texttt{ => } e_0) \\
[app_1] & \dfrac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (ie_1\ ie_2)^{\ell} \to (ie_1'\ ie_2)^{\ell}} \\
[app_2] & \dfrac{\rho \vdash ie_2 \to ie_2'}{\rho \vdash (v_1^{\ell_1}\ ie_2)^{\ell} \to (v_1^{\ell_1}\ ie_2')^{\ell}} \\
[app_{fn}] & \rho \vdash ((\texttt{close}\ (\texttt{fn}\ x \texttt{ => } e_1) \texttt{ in } \rho_1)^{\ell_1}\ v_2^{\ell_2})^{\ell} \to \\
& \qquad (\texttt{bind}\ \rho_1[x \mapsto v_2] \texttt{ in } e_1)^{\ell} \\
[app_{fun}] & \rho \vdash ((\texttt{close}\ (\texttt{fun}\ f\ x \texttt{ => } e_1) \texttt{ in } \rho_1)^{\ell_1}\ v_2^{\ell_2})^{\ell} \to \\
& \qquad (\texttt{bind}\ \rho_2[x \mapsto v_2] \texttt{ in } e_1)^{\ell} \\
& \qquad \text{where } \rho_2 = \rho_1[f \mapsto \texttt{close}\ (\texttt{fun}\ f\ x \texttt{ => } e_1) \texttt{ in } \rho_1] \\
[bind_1] & \dfrac{\rho_1 \vdash ie_1 \to ie_1'}{\rho \vdash (\texttt{bind}\ \rho_1 \texttt{ in } ie_1)^{\ell} \to (\texttt{bind}\ \rho_1 \texttt{ in } ie_1')^{\ell}} \\
[bind_2] & \rho \vdash (\texttt{bind}\ \rho_1 \texttt{ in } v_1^{\ell_1})^{\ell} \to v_1^{\ell}
\end{array}
$$

Table 3.2: The Structural Operational Semantics of FUN (part 1).

$$
\begin{array}{ll}
[if_1] & \dfrac{\rho \vdash ie_0 \to ie_0'}{\rho \vdash (\texttt{if}\ ie_0 \texttt{ then } e_1 \texttt{ else } e_2)^{\ell} \to (\texttt{if}\ ie_0' \texttt{ then } e_1 \texttt{ else } e_2)^{\ell}} \\
[if_2] & \rho \vdash (\texttt{if}\ \texttt{true}^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^{\ell} \to t_1^{\ell} \\
[if_3] & \rho \vdash (\texttt{if}\ \texttt{false}^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^{\ell} \to t_2^{\ell} \\
[let_1] & \dfrac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (\texttt{let}\ x = ie_1 \texttt{ in } e_2)^{\ell} \to (\texttt{let}\ x = ie_1' \texttt{ in } e_2)^{\ell}} \\
[let_2] & \rho \vdash (\texttt{let}\ x = v^{\ell_1} \texttt{ in } e_2)^{\ell} \to (\texttt{bind}\ \rho_0[x \mapsto v] \texttt{ in } e_2)^{\ell} \\
& \qquad \text{where } \rho_0 = \rho \mid FV(e_2) \\
[op_1] & \dfrac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (ie_1\ op\ ie_2)^{\ell} \to (ie_1'\ op\ ie_2)^{\ell}} \\
[op_2] & \dfrac{\rho \vdash ie_2 \to ie_2'}{\rho \vdash (v_1^{\ell_1}\ op\ ie_2)^{\ell} \to (v_1^{\ell_1}\ op\ ie_2')^{\ell}} \\
[op_3] & \rho \vdash (v_1^{\ell_1}\ op\ v_2^{\ell_2})^{\ell} \to v^{\ell} \qquad \text{if } v = v_1 \mathbf{\ op\ } v_2
\end{array}
$$

Table 3.3: The Structural Operational Semantics of FUN (part 2).

My challenge to ICFP:

Develop a program analysis for reasoning about:

Space-consumption in a lazy language

State and control in a language with effects

Security in a language with stack inspection

Blame in a language with behavioral contracts

Safe parallelism in a language with futures

My ch
ICFP:

Develop a progr ut:

- Spac age
- Stat th effects
- Secu nspection
- Blam al contracts
- Safe futures

B-2 - Cody Enterprise - Monday, August 16, 2010

SPORTS/OUTDOORS

Rough ride

photos by SCOTT SALISBURY

Geoff Marolda of Houston, Texas, is thrown off a bronc as he comes out of the chute during bareback riding recently at the Cody Nite Rodeo.

Modularity of existing analyses

Three approaches:

Three approaches:

Do nothing (analyze whole programs only)

Three approaches:

Do nothing (analyze whole programs only)

Hemorrhage precision (black hole approach)

Three approaches:

Do nothing (analyze whole programs only)

Hemorrhage precision (black hole approach)

Do something really complicated

Modular Set-Based Analysis from Contracts

Philippe Meunier
College of Computer and Information Science, Northeastern University
meunier@ccs.neu.edu

Robert Bruce Findler
Department of Computer Science, University of Chicago
robby@cs.uchicago.edu

Matthias Felleisen
College of Computer and Information Science, Northeastern University
matthias@ccs.neu.edu

Abstract

In PLT Scheme, programs consist of modules with contracts. The latter describe the inputs and outputs of functions and objects via predicates. A run-time system enforces these predicates; if a predicate fails, the enforcer raises an exception that blames a specific module with an explanation of the fault.

In this paper, we show how to use such module contracts to turn set-based analysis into a fully modular parameterized analysis. Using this analysis, a static debugger can indicate for any given contract check whether the corresponding predicate is always satisfied, partially satisfied, or (potentially) completely violated. The static debugger can also predict the source of potential errors, i.e., it is sound with respect to the blame assignment of the contract system.

Categories and Subject Descriptors F.3.2 [*Semantics of Programming Languages*]: Program analysis; D.2.4 [*Software / Program Verification*]: Programming by contract

General Terms Languages, Reliability, Verification.

Keywords Static Debugging, Set-based Analysis, Modular Analysis, Runtime Contracts.

1. Modules, Contracts, and Static Debugging

A static debugger helps programmers find errors via program analyses. It uses the invariants of the programming language to analyze the program and determines whether the program may violate one of them during execution. For example, a static debugger can find expressions that may dereference null pointers. Some static debuggers use lightweight analyses, e.g., Flanagan et al.'s MrSpidey [11] relies on a variant of set-based analysis [10, 16, 21]; others use a deep abstract interpretation, e.g., Bourdoncle's Syntox [4]; and yet others employ theorem proving, e.g., Detlefs et al.'s ESC [7].

Experience with static debuggers shows that they work well for reasonably small programs. Using MrSpidey, we have routinely debugged or re-engineered programs of 2,000 to 5,000 lines of code in PLT Scheme. Flanagan has successfully analyzed the core of the interpreter, dubbed MrEd [13], a 40,000 line program. Existing static debuggers, however, suffer from a monolithic approach to program analysis. Because their analyses require the availability of the entire program, programmers cannot analyze their programs until they have everyone else's modules.

POPL'06 January 11–13, 2006, Charleston, South Carolina, USA.

Over the past few years, we have added a first-order module system to PLT Scheme [12] and have equipped the module system with a contract system [8]. A contract is roughly a predicate on the inputs and outputs of (exported) functions, including object methods and higher-order functions. The contract system monitors the contracts during program execution. If a module violates a contract, the contract system pinpoints the guilty party and issues an explanatory message.

This paper makes five contributions to static debugging and software contracts. *First*, it explains how to construct a modular static debugger for programs with contracts, using those contracts in a dual role: one as a source of abstract values and one as a sink for abstract values. *Second*, we prove that our contract-based, whole-program analysis computes its results in a modular manner. That is, our contract-aware set-based analysis produces the same predictions for a given point in the program regardless of whether it analyzes the whole program or just the surrounding module. *Third*, for any given contract check, the system indicates whether the corresponding predicate is always satisfied, partially satisfied, or completely violated. *Fourth*, the static debugger can also predict the source of potential errors, i.e., it is sound with respect to the blame assignment of the contract system. *Fifth*, the analysis is parameterized over both a predicate approximation relation and a predicate domain function.

2. Overview

The paper presents a model of a modular static debugger. The model consists of two parts: a runtime contract system and a set-based analysis for modules with contracts. A correctness theorem ties the two parts together. Figure 1 provides an overview of these three pieces in graphical form. The vertical column on the left represents the runtime contract system. A contract compiler translates a collection of modules and a main expression into a suitably annotated form. During execution, which we naturally model via a reduction system, the contract system keeps track of the contract obligations; if something goes wrong it blames a specific module.

The first horizontal row of Figure 1 depicts the analysis process, which consists of three stages. First, it partitions the program into module-like pieces by lifting expressions with contract annotations out of the main program. Second, the resulting collection of program pieces is analyzed with a parameterized set-based analysis. This step yields both sets of abstract values and sets of potential errors, including explanations that blame the guilty party; we call the latter *blame sets*. Third, the former are summarized as set-of-values descriptions, dubbed *types*.

The rest of the grid in Figure 1 explains our proof technique for the correctness theorem. Since each reduction step creates a complete program, the correctness proof can proceed via subject reduction. We re-apply the analysis after each reduction step. The proof then shows that the reductions preserve the types and the blame

M

Philippe Me
College of Computer a
Science, Northeaster
meunier@ccs.n

Abstract

In PLT Scheme, program
latter describe the inputs
via predicates. A run-tin
a predicate fails, the enf
specific module with an e
In this paper, we show
set-based analysis into a f
ing this analysis, a static
tract check whether the co
partially satisfied, or (pot
debugger can also predic
sound with respect to the

Categories and Subject
gramming Languages]: P
gram Verification]: Progr

General Terms Langua

Keywords Static Debug
ysis, Runtime Contracts.

1. Modules, Contr

A static debugger helps p
yses. It uses the invariants
the program and determin
of them during execution.
expressions that may dere
gers use lightweight analy
relies on a variant of set-
deep abstract interpretatio
others employ theorem pr
Experience with static
reasonably small progran
debugged or re-engineere
in PLT Scheme. Flanaga
the interpreter, dubbed M
static debuggers, howeve
program analysis. Becau
of the entire program, pro
until they have everyone

*Source**Sink*	$\mathsf{int}_h^{\ell_5^+\ell_5^-}$	$\langle \ldots e_5\ \mathsf{int}_h^{\ell_5^+\ell_5^-} \rangle_h^{\ell_6^+\ell_6^-}$	$\mathsf{any}_h^{\ell_5^+\ell_5^-}$	$\langle \ldots e_5\ \mathsf{any}_h^{\ell_5^+\ell_5^-} \rangle_h^{\ell_6^+\ell_6^-}$
$n_{e_1\ldots}^{\ell_n}$		$\left.\begin{matrix}\{\ell_n\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{matrix}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\left.\begin{matrix}\{\ell_n\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{matrix}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\mathsf{int}_f^{\ell_1^+\ell_1^-}$		$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle \ldots e_1\ \mathsf{int}_f^{\ell_1^+\ell_1^-} \rangle_f^{\ell_2^+\ell_2^-}$		$\left.\begin{matrix}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\sqsubseteq e_5\end{matrix}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\left.\begin{matrix}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\sqsubseteq e_5\end{matrix}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\mathsf{any}_f^{\ell_1^+\ell_1^-}$	$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$			$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle \ldots e_1\ \mathsf{any}_f^{\ell_1^+\ell_1^-} \rangle_f^{\ell_2^+\ell_2^-}$				$\left.\begin{matrix}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\sqsubseteq e_5\end{matrix}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$(\lambda x^\beta.e^\ell)_{e_1\ldots}^{\ell_\lambda}$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_5^+)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_5^-)$	$\left.\begin{matrix}\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{matrix}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$(c_g^{\ell_1^+\ell_1^-}\to c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$			$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle \ldots e_3\ (c_g^{\ell_1^+\ell_1^-}\to c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-} \rangle_f^{\ell_4^+\ell_4^-}$			$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_5^+)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_5^-)$	$\left.\begin{matrix}\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\\ e_3\not\sqsubseteq e_5\end{matrix}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$

*Source**Sink*	$(e^{\ell_5}\ e^{\ell_6})^{\ell_a}$	$(c_i^{\ell_7^+\ell_7^-}\to c_h^{\ell_8^+\ell_8^-})_h^{\ell_5^+\ell_5^-}$	$\langle \ldots e_5\ (c_i^{\ell_7^+\ell_7^-}\to c_h^{\ell_8^+\ell_8^-})_h^{\ell_5^+\ell_5^-} \rangle_h^{\ell_6^+\ell_6^-}$
$n_{e_1\ldots}^{\ell_n}$	$\{\ell_n\}\subseteq\varphi(\ell_5)\Rightarrow\{\langle\lambda,\mathcal{R}\rangle\}\subseteq\psi(\ell_a)$	$\{\ell_n\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$	
$\mathsf{int}_f^{\ell_1^+\ell_1^-}$	$\{\ell_1^+\}\subseteq\varphi(\ell_5)\Rightarrow\{\langle\lambda,\mathcal{R}\rangle\}\subseteq\psi(\ell_a)$	$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$	
$\langle \ldots e_1\ \mathsf{int}_f^{\ell_1^+\ell_1^-} \rangle_f^{\ell_2^+\ell_2^-}$			
$\mathsf{any}_f^{\ell_1^+\ell_1^-}$			
$\langle \ldots e_1\ \mathsf{any}_f^{\ell_1^+\ell_1^-} \rangle_f^{\ell_2^+\ell_2^-}$			
$(\lambda x^\beta.e^\ell)_{e_1\ldots}^{\ell_\lambda}$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_6)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_a)$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_7^+)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_8^-)$	$\left.\begin{matrix}\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{matrix}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$(c_g^{\ell_1^+\ell_1^-}\to c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}$	$\{\ell_3^+\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_6)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_a)$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_7^+)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_8^-)$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle \ldots e_3\ (c_g^{\ell_1^+\ell_1^-}\to c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-} \rangle_f^{\ell_4^+\ell_4^-}$			$\left.\begin{matrix}\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\\ e_3\not\sqsubseteq e_5\end{matrix}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$

Table 1. Constraints creation for source-sink pairs.

M

Philippe Me
College of Computer a
Science, Northeaster
meunier@ccs.n

Abstract

In PLT Scheme, program
latter describe the inputs
via predicates. A run-tin
a predicate fails, the enf
specific module with an e
In this paper, we show
set-based analysis into a f
ing this analysis, a static
tract check whether the co
partially satisfied, or (pot
debugger can also predic
sound with respect to the

Categories and Subject
gramming Languages]: P
gram Verification]: Progra

General Terms Langua

Keywords Static Debug
ysis, Runtime Contracts.

1. Modules, Contr

A static debugger helps p
yses. It uses the invariants
the program and determin
of them during execution.
expressions that may dere
gers use lightweight analy
relies on a variant of set-
deep abstract interpretatio
others employ theorem pr
Experience with static
reasonably small progran
debugged or re-engineere
in PLT Scheme. Flanaga
the interpreter, dubbed M
static debuggers, howeve
program analysis. Becau
of the entire program, pro
until they have everyone

*Source**Sink*	$\mathsf{int}_h^{\ell_5^+\ell_5^-}$	$\langle\ldots e_5\ \mathsf{int}_h^{\ell_5^+\ell_5^-}\rangle_h^{\ell_6^+\ell_6^-}$	$\mathsf{any}_h^{\ell_5^+\ell_5^-}$	$\langle\ldots e_5\ \mathsf{any}_h^{\ell_5^+\ell_5^-}\rangle_h^{\ell_6^+\ell_6^-}$
$n_{e_1\ldots}^{\ell_n}$		$\left.\begin{array}{l}\{\ell_n\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\left.\begin{array}{l}\{\ell_n\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\mathsf{int}_f^{\ell_1^+\ell_1^-}$		$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle\ldots e_1\ \mathsf{int}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$		$\left.\begin{array}{l}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\left.\begin{array}{l}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\mathsf{any}_f^{\ell_1^+\ell_1^-}$	$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$			$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle\ldots e_1\ \mathsf{any}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$				$\left.\begin{array}{l}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$(\lambda x^\beta.e^\ell)_{e_1\ldots}^{\ell_\lambda}$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_5^+)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_5^-)$	$\left.\begin{array}{l}\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$(c_g^{\ell_1^+\ell_1^-}\to c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_5^+)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_5^-)$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$ $\left.\begin{array}{l}\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\\ e_3\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle\ldots e_3\ (c_g^{\ell_1^+\ell_1^-}\to c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}\rangle_f^{\ell_4^+\ell_4^-}$				

*Source**Sink*	$(e^{\ell_5}\ e^{\ell_6})^{\ell_a}$	$(c_i^{\ell_7^+\ell_7^-}\to c_h^{\ell_8^+\ell_8^-})_h^{\ell_5^+\ell_5^-}$	$\langle\ldots e_5\ (c_i^{\ell_7^+\ell_7^-}\to c_h^{\ell_8^+\ell_8^-})_h^{\ell_5^+\ell_5^-}\rangle_h^{\ell_6^+\ell_6^-}$
$n_{e_1\ldots}^{\ell_n}$	$\{\ell_n\}\subseteq\varphi(\ell_5)\Rightarrow\{\langle\lambda,\mathcal{R}\rangle\}\subseteq\psi(\ell_a)$	$\{\ell_n\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$	
$\mathsf{int}_f^{\ell_1^+\ell_1^-}$	$\{\ell_1^+\}\subseteq\varphi(\ell_5)\Rightarrow\{\langle\lambda,\mathcal{R}\rangle\}\subseteq\psi(\ell_a)$	$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$	
$\langle\ldots e_1\ \mathsf{int}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$			
$\mathsf{any}_f^{\ell_1^+\ell_1^-}$			
$\langle\ldots e_1\ \mathsf{any}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$			
$(\lambda x^\beta.e^\ell)_{e_1\ldots}^{\ell_\lambda}$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_6)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_a)$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_7^+)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_8^-)$	$\left.\begin{array}{l}\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$(c_g^{\ell_1^+\ell_1^-}\to c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}$	$\{\ell_3^+\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_6)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_a)$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_7^+)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_8^-)$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$ $\left.\begin{array}{l}\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\\ e_3\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle\ldots e_3\ (c_g^{\ell_1^+\ell_1^-}\to c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}\rangle_f^{\ell_4^+\ell_4^-}$			

Table 1. Constraints creation for source-sink pairs.

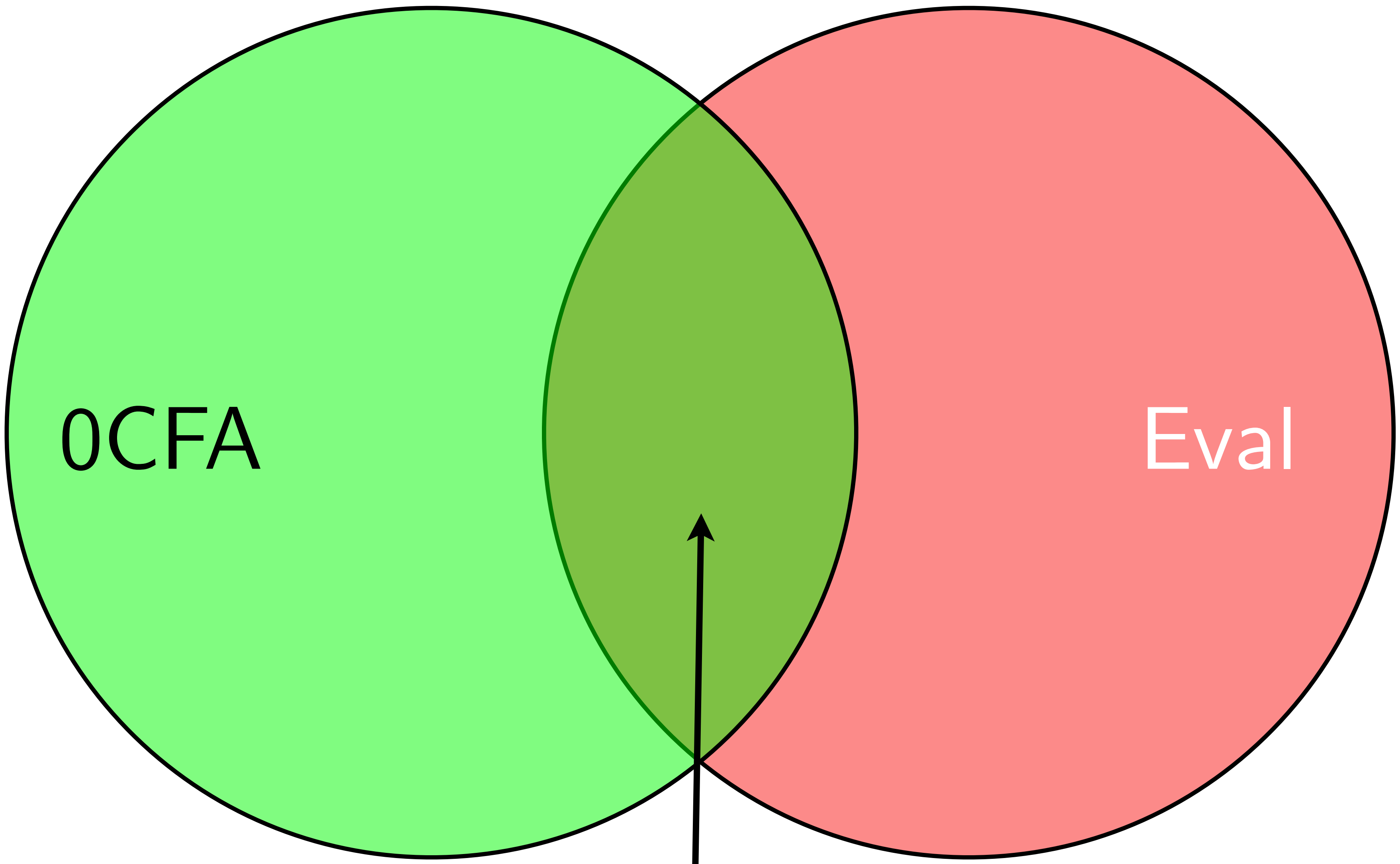
Higher-order Program Analysis is Alive and Well.

(I have a way forward.)

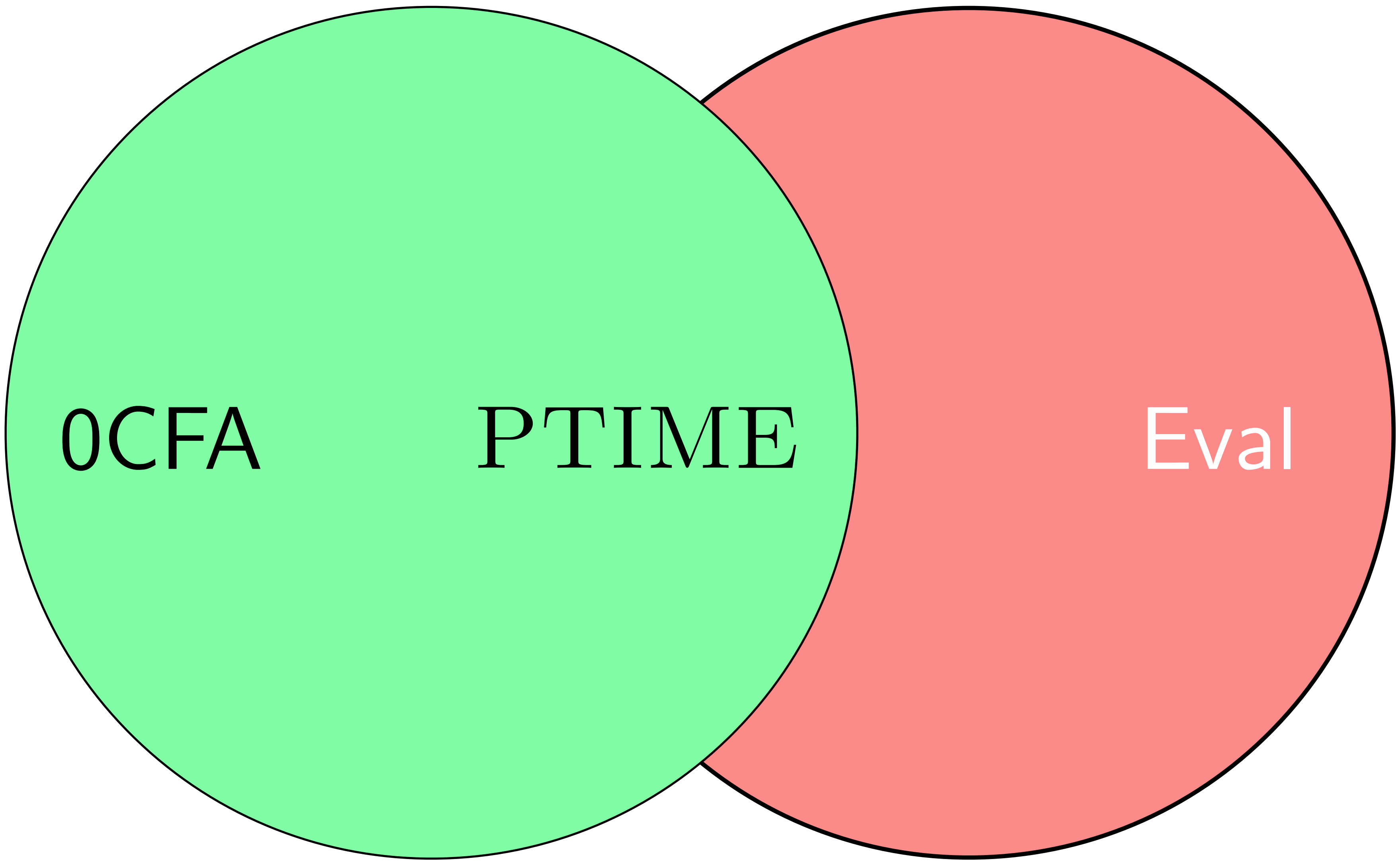
A Way Forward

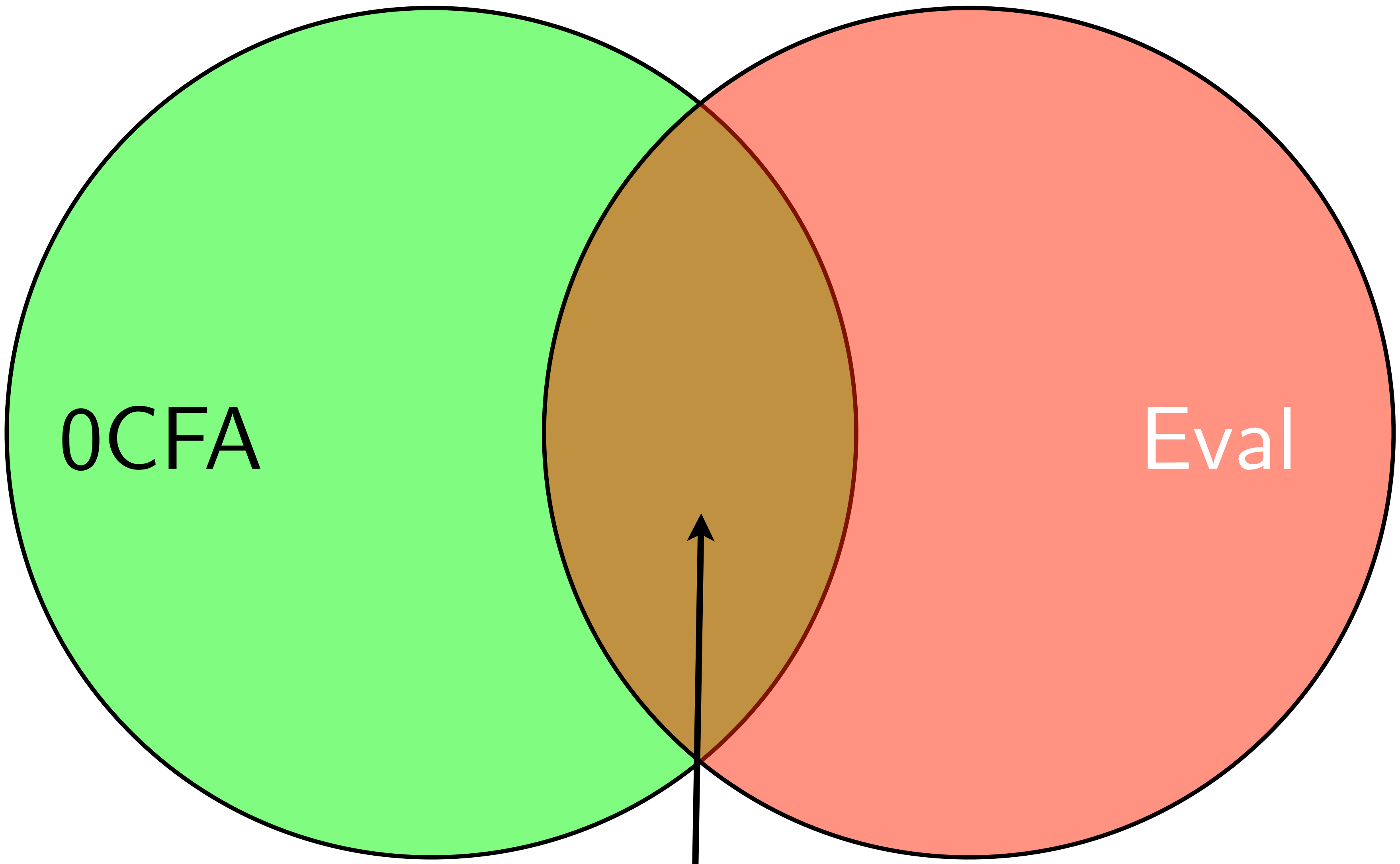
Scalability $\begin{cases} \text{Complexity} \\ \text{Maintenance} \\ \text{Verification} \\ \text{Expressivity} \\ \text{Modularity} \end{cases}$

Key insight: analysis is a kind of evaluation

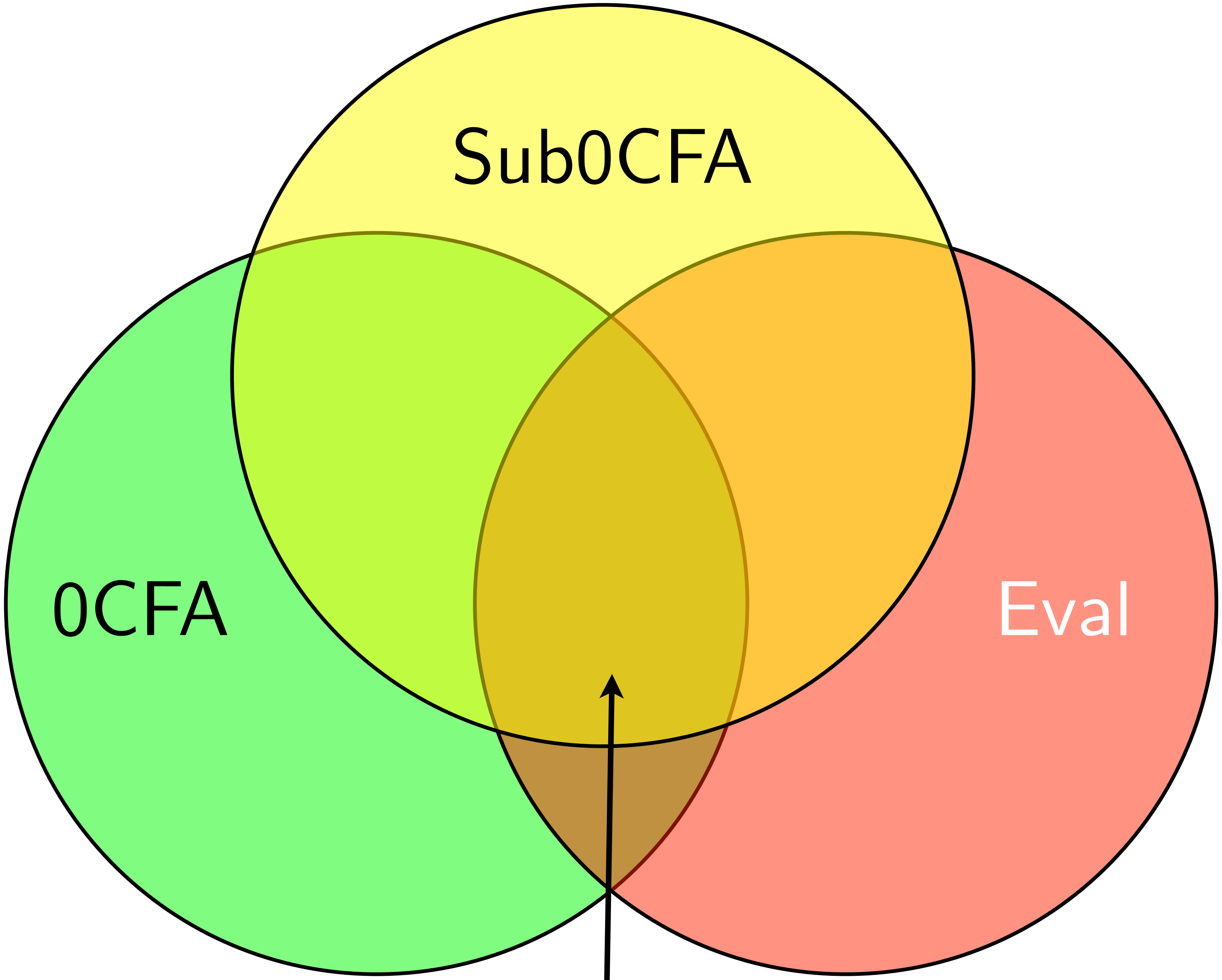


Every variable occurs once.

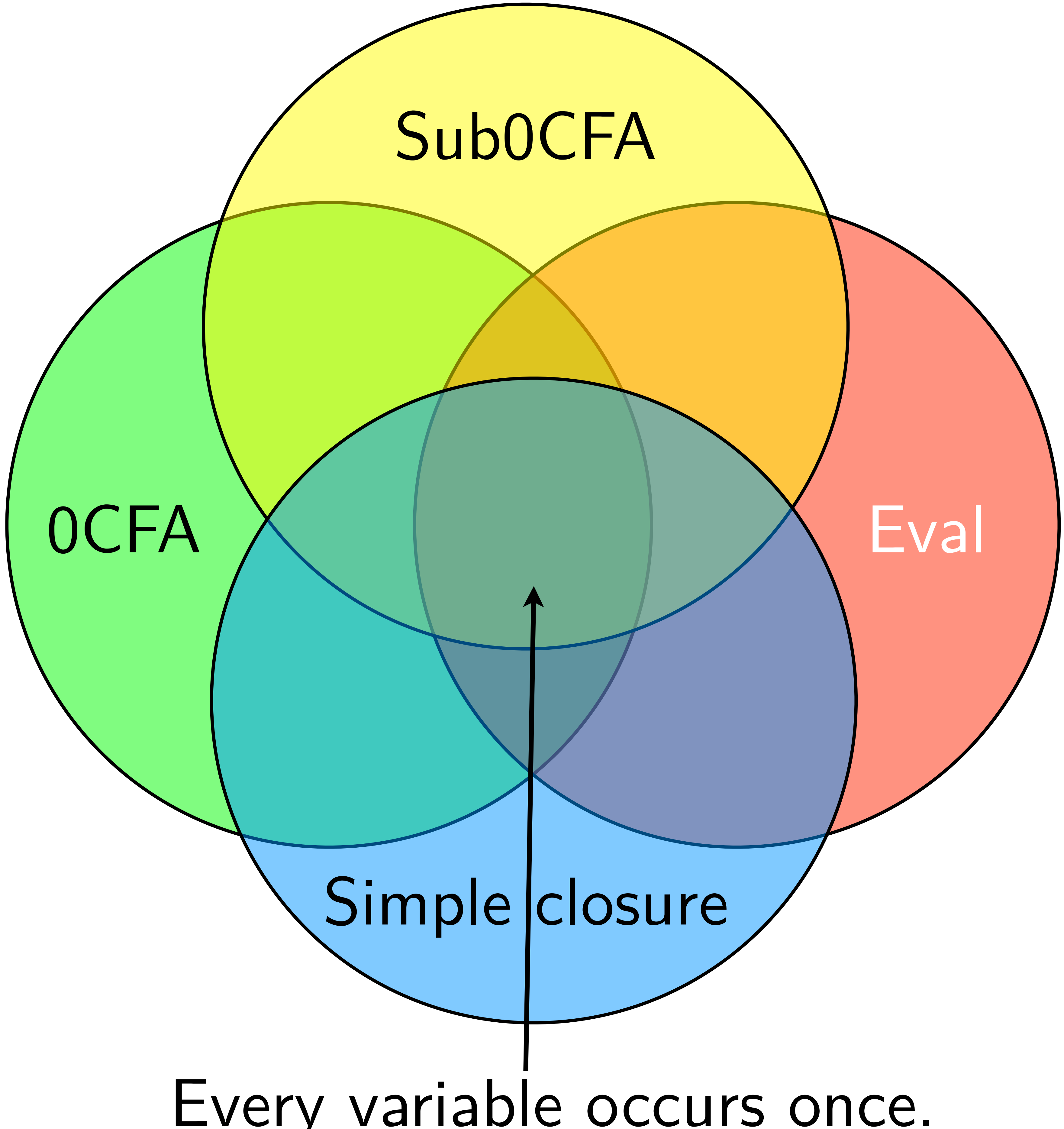

0CFA
PTIME
Eval

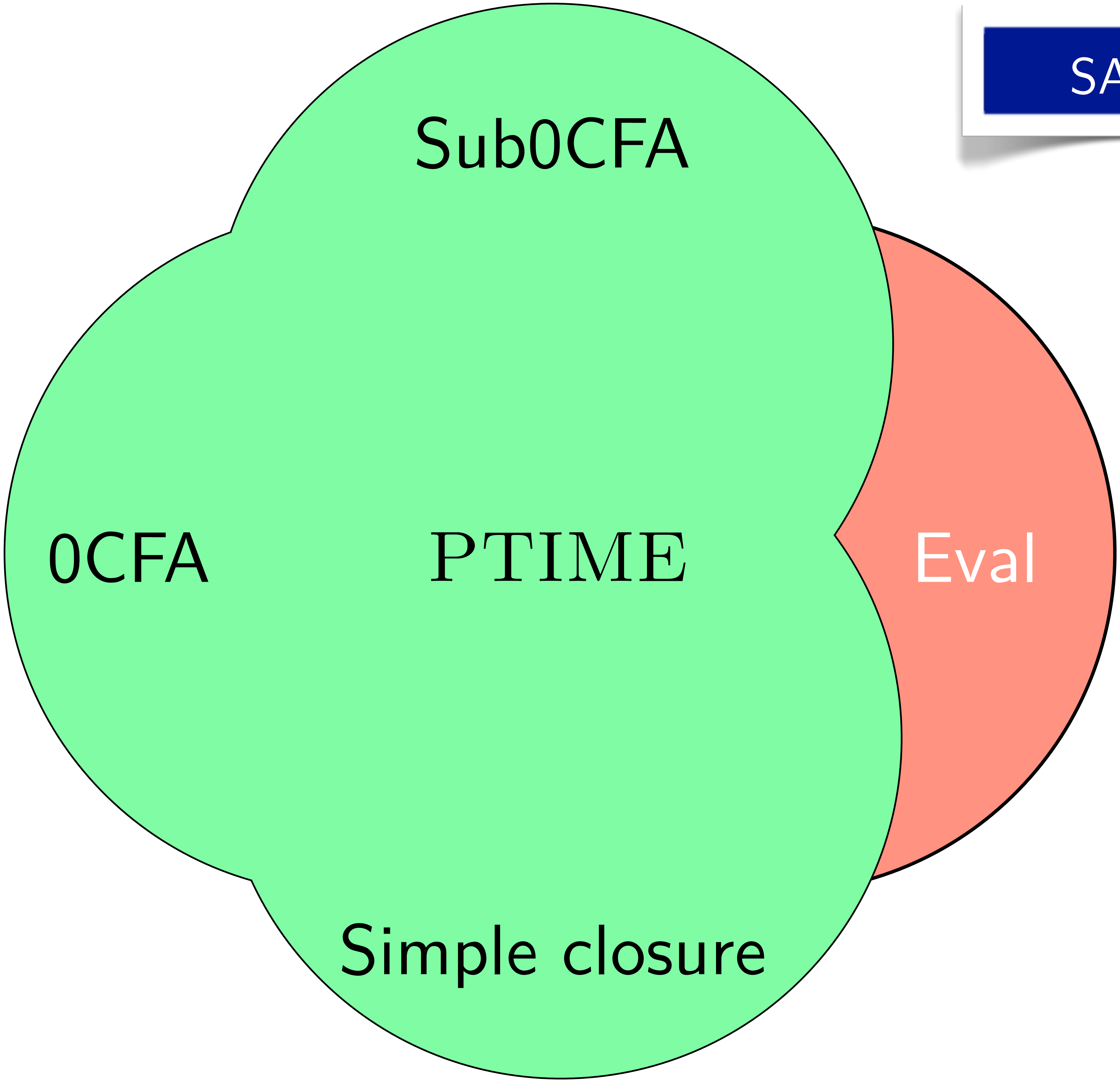


Every variable occurs once.

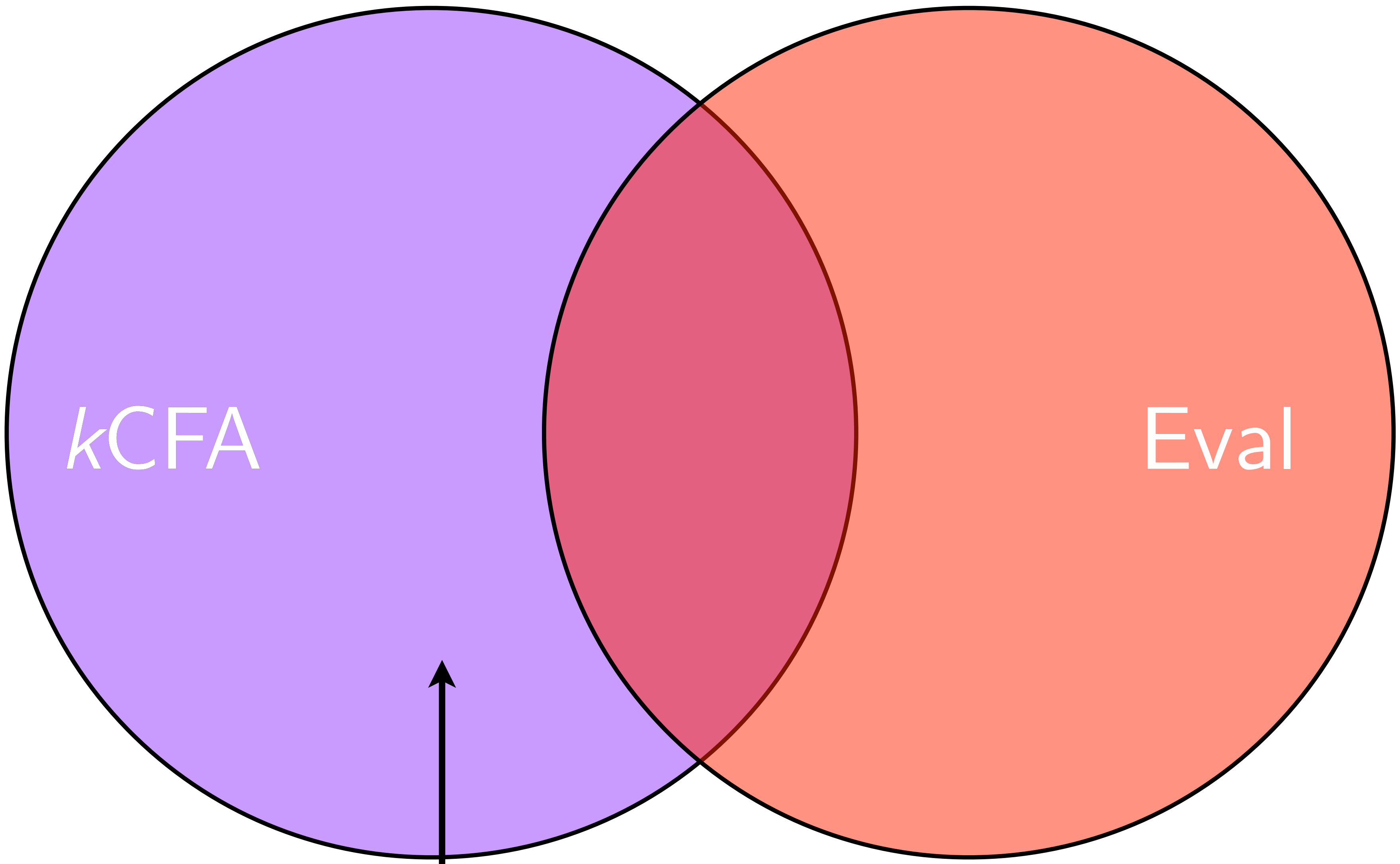


Every variable occurs once.


Sub0CFA
0CFA
Eval
Simple closure
Every variable occurs once.

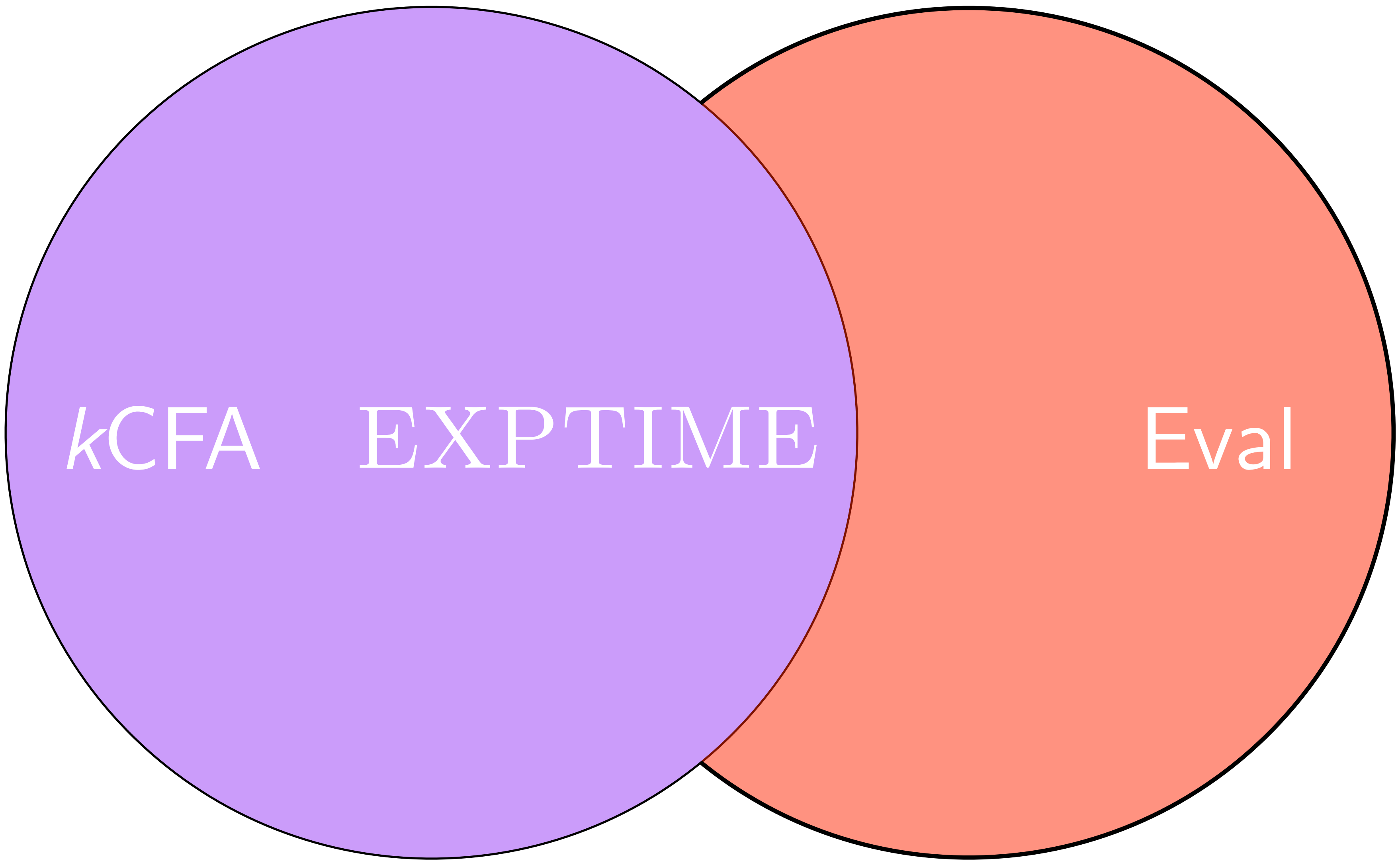


SAS'08

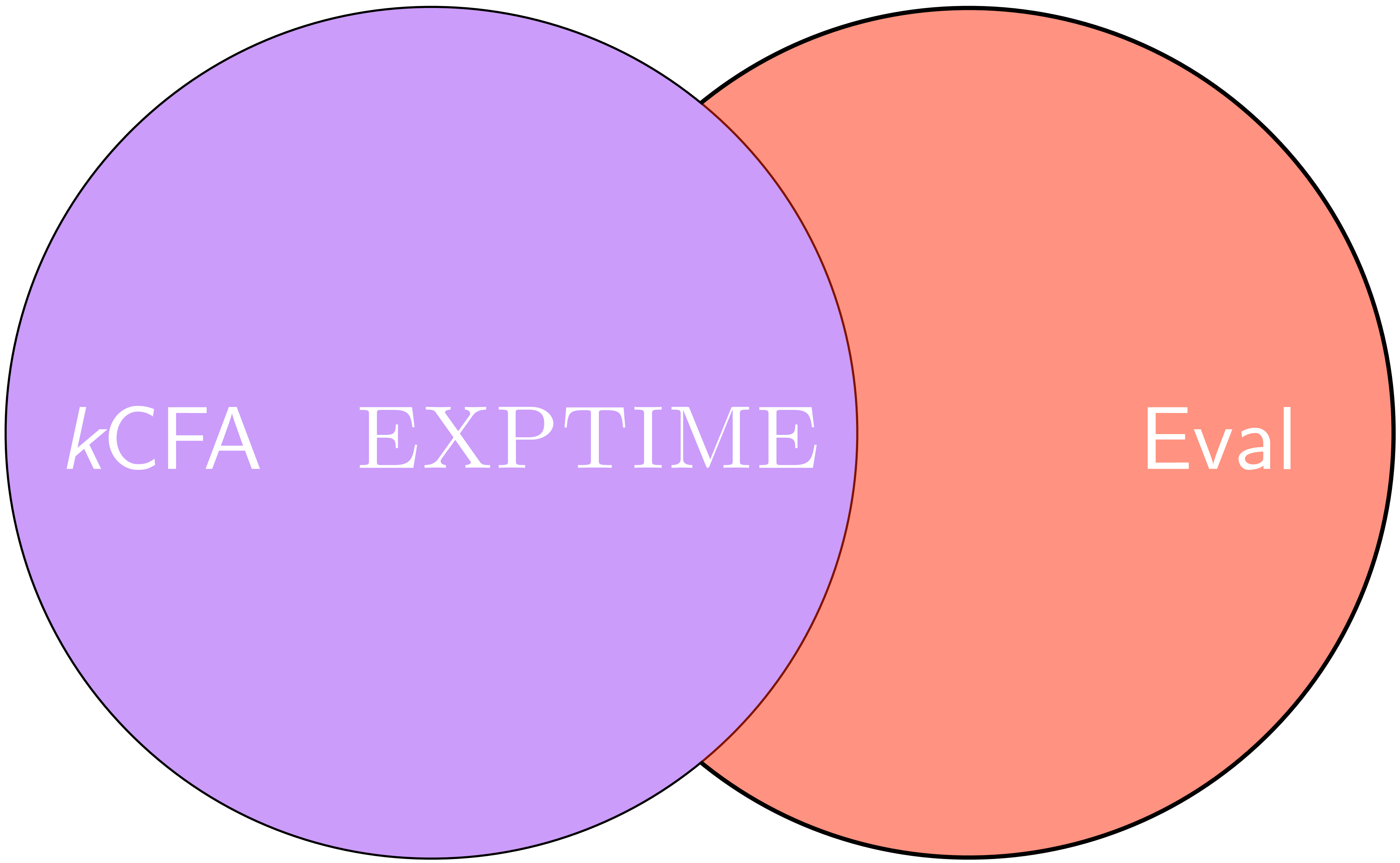


Datalog-style programming with analysis.

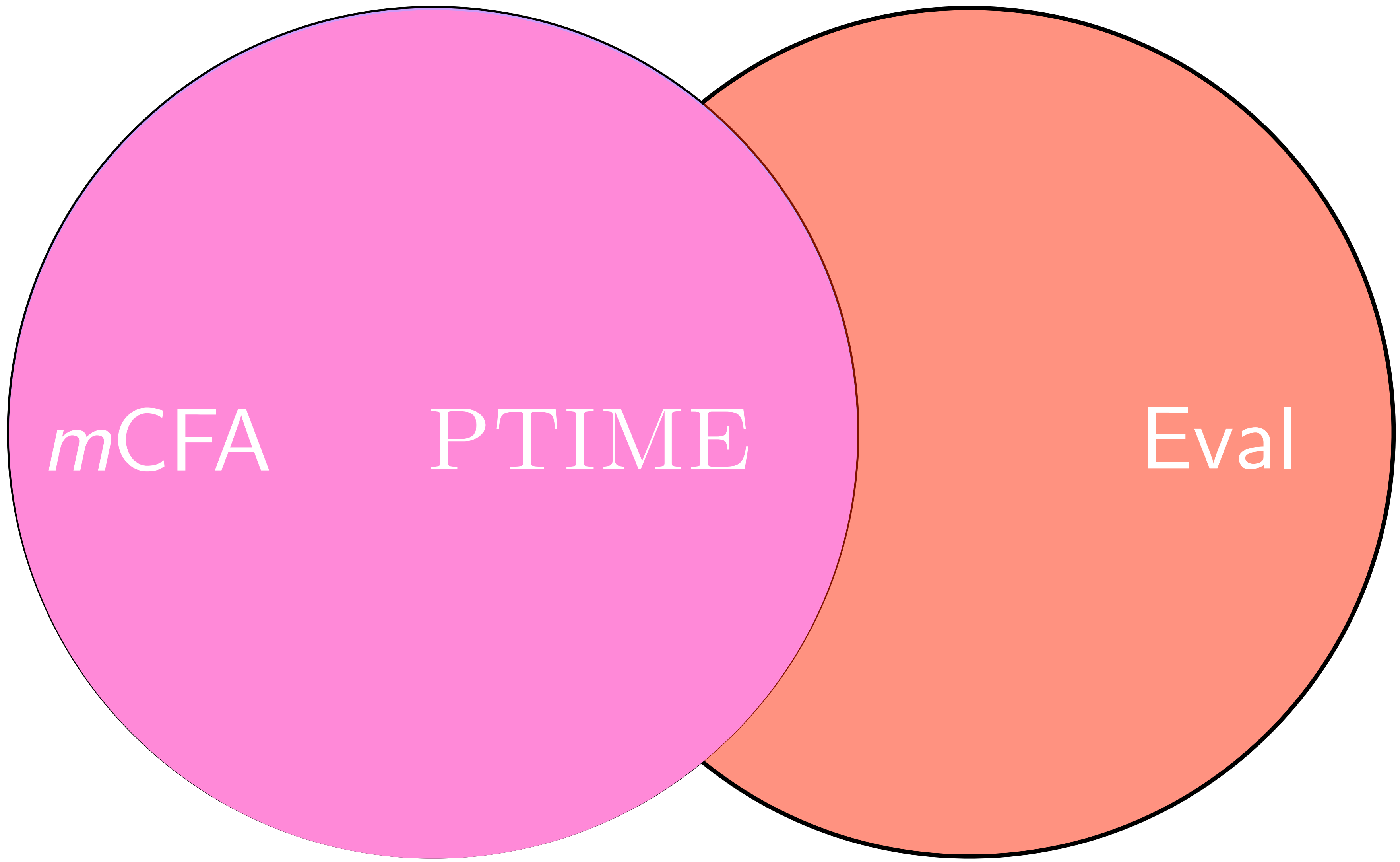
ICFP'08



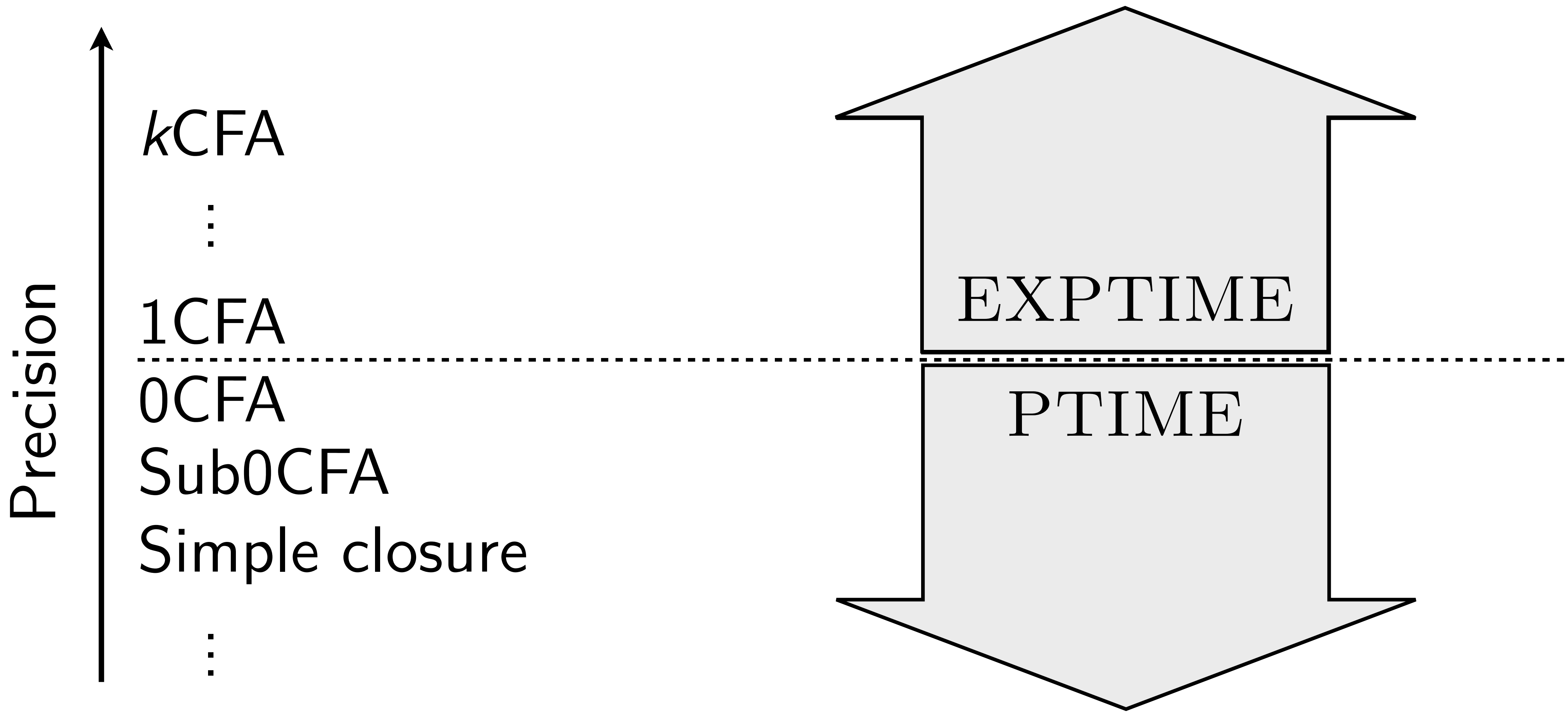
ICFP'08



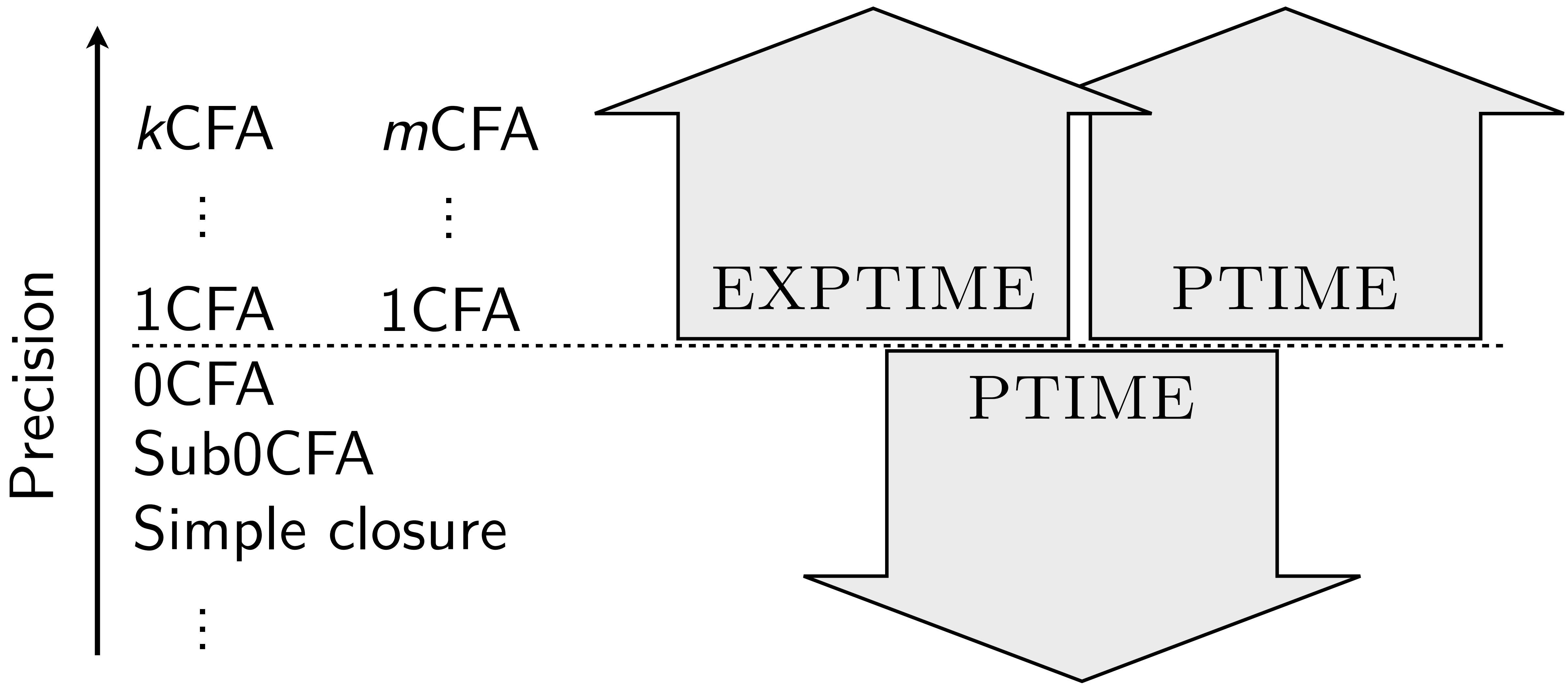
PLDI'10



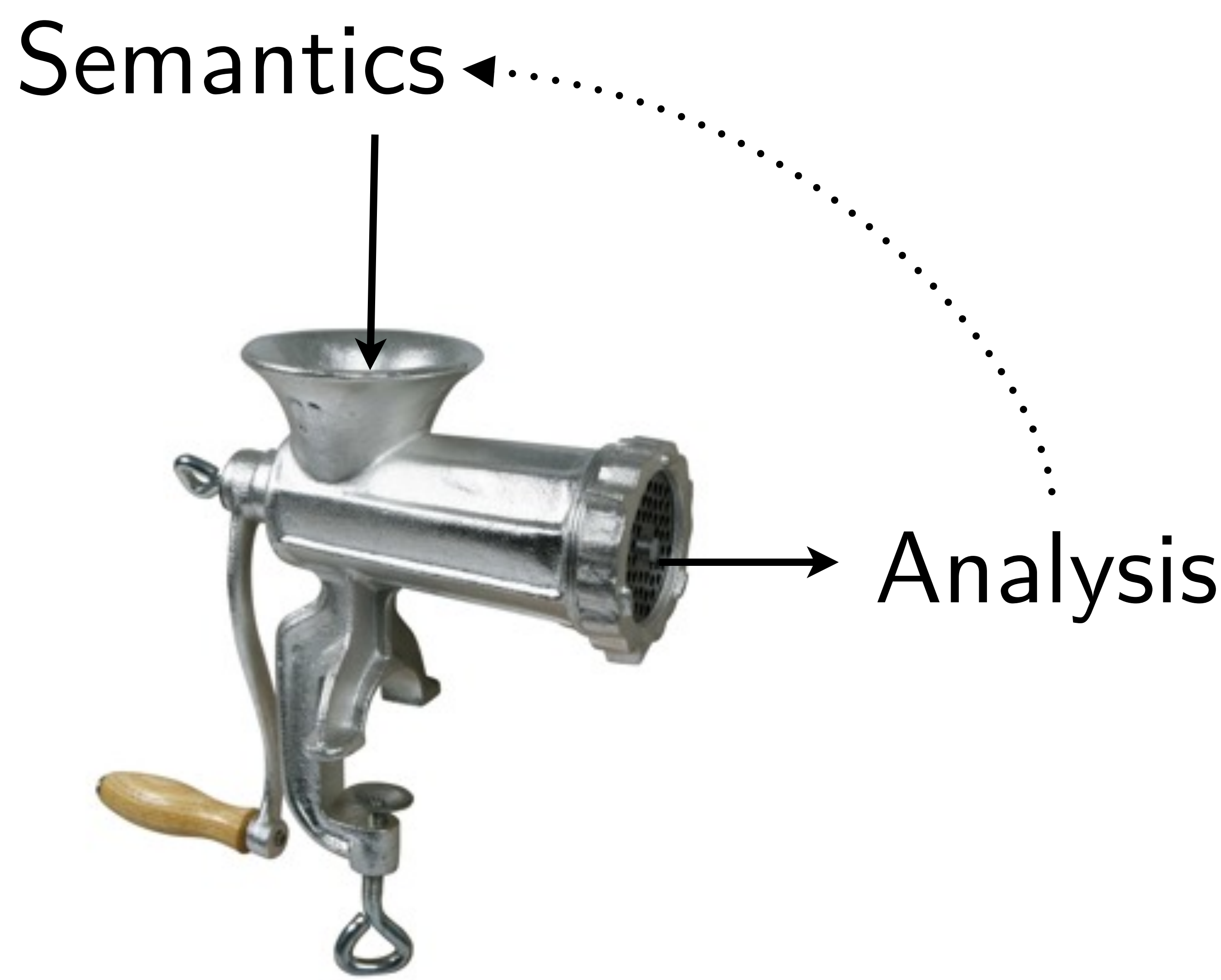
Similar precision, better performance


Precision
kCFA
⋮
1CFA
0CFA
Sub0CFA
Simple closure
⋮
EXPTIME
PTIME

PLDI'10


Precision
*k*CFA
*m*CFA
1CFA
1CFA
0CFA
Sub0CFA
Simple closure
EXPTIME
PTIME
PTIME

A Systematic Approach to Program Analysis Design

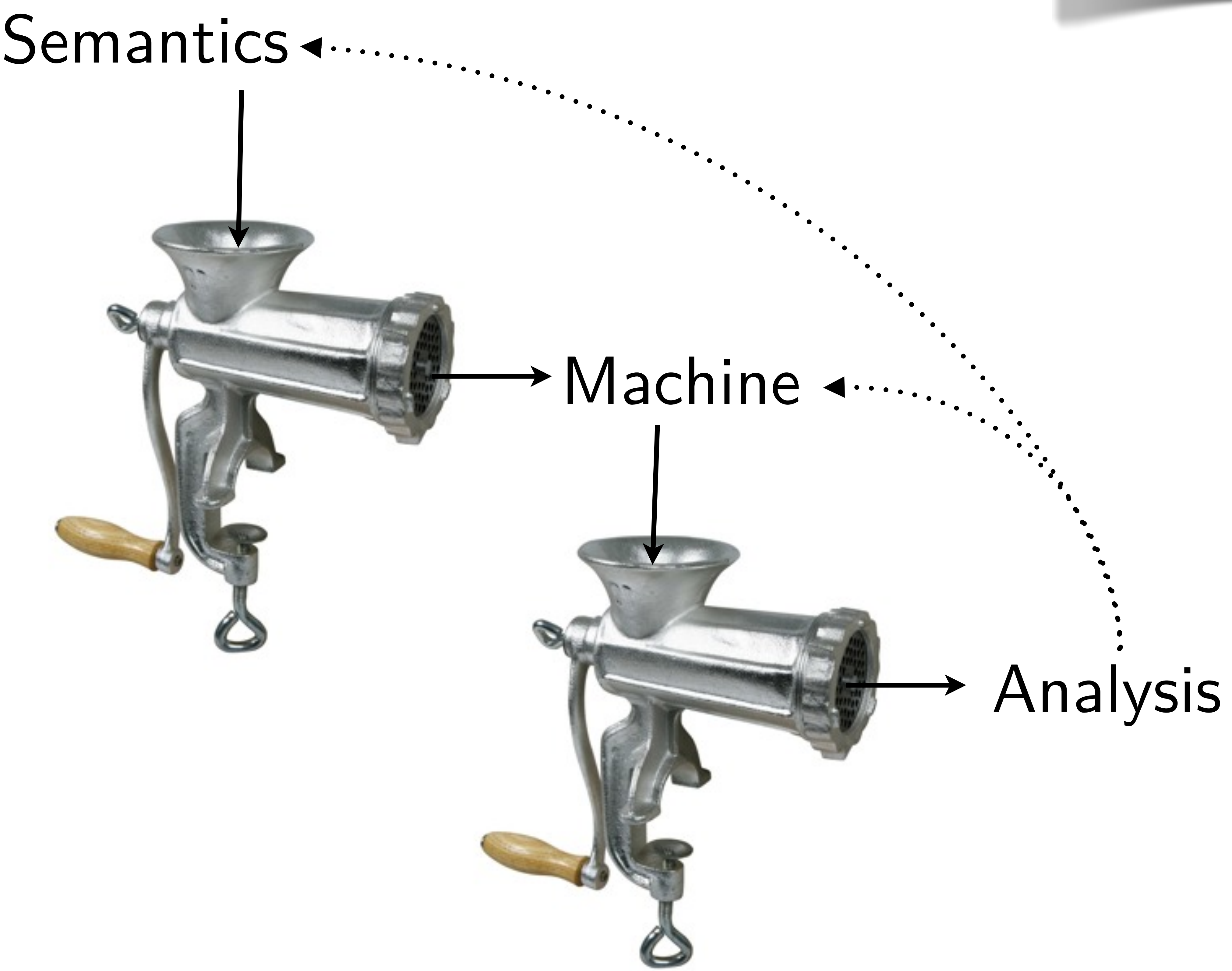

Semantics
Analysis

ICFP'10/CACM'11

Semantics

Analysis

ICFP'10/CACM'11



Analysis machine

ICFP'10/CACM'11

$$\begin{tikzcd} e \arrow[d] \\ s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & s_3 \arrow[r] & s_4 \arrow[r] & \ldots \end{tikzcd}$$

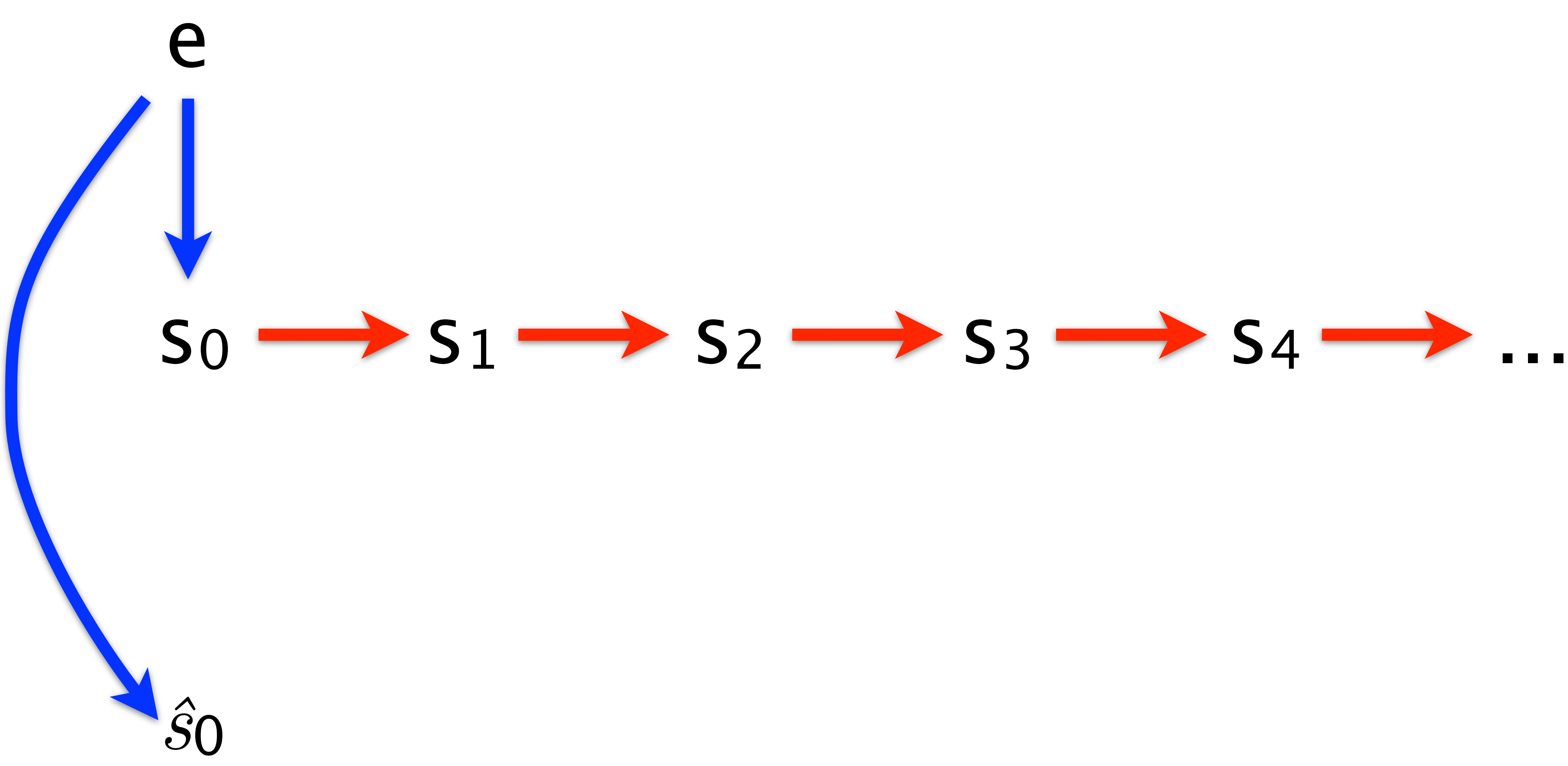
Analysis machine

ICFP'10/CACM'11

$$
\begin{tikzcd}
e \arrow[d] \\
s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & s_3 \arrow[r] & s_4 \arrow[r] & \ldots
\end{tikzcd}
$$

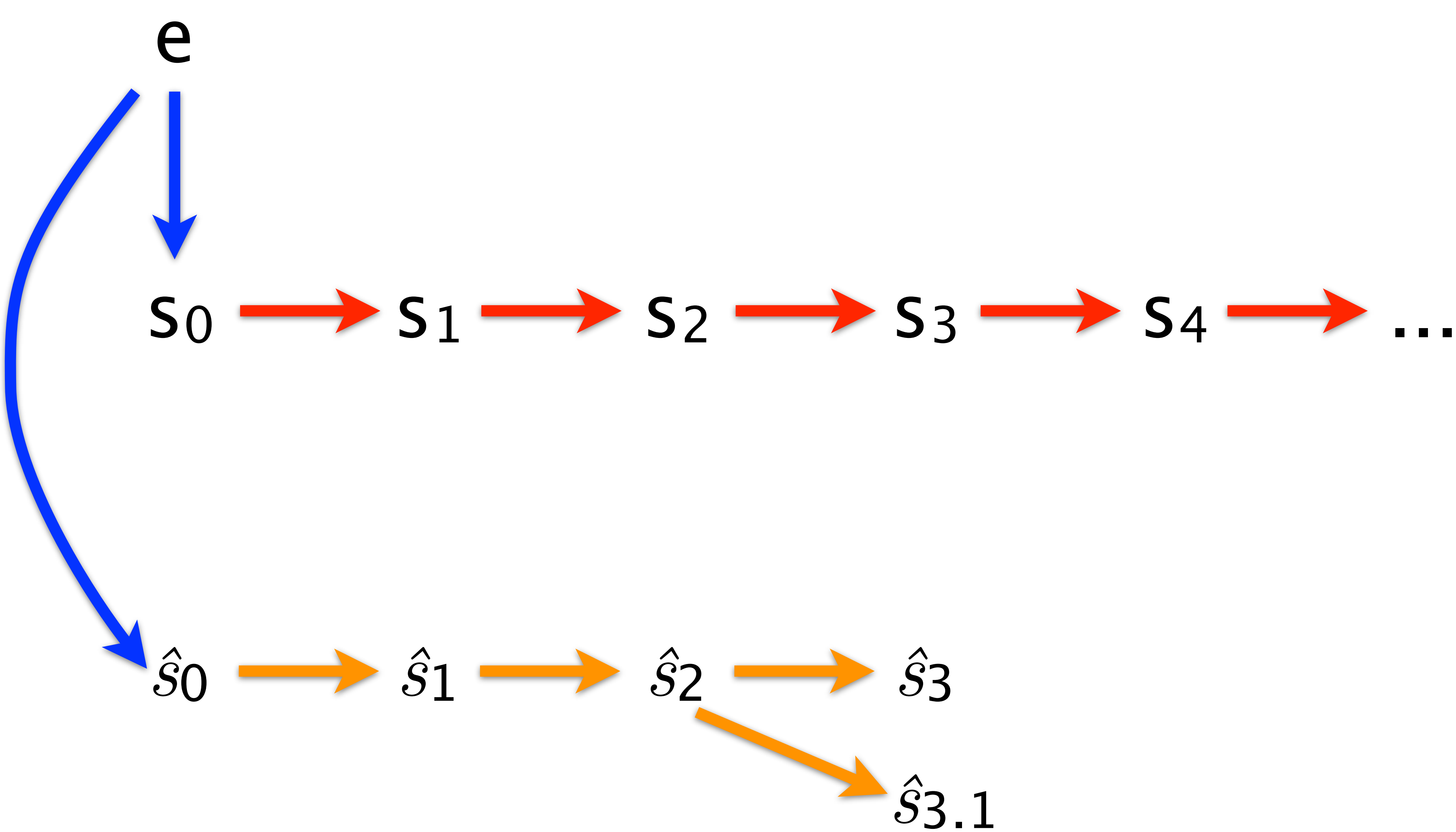
Analysis machine

ICFP'10/CACM'11



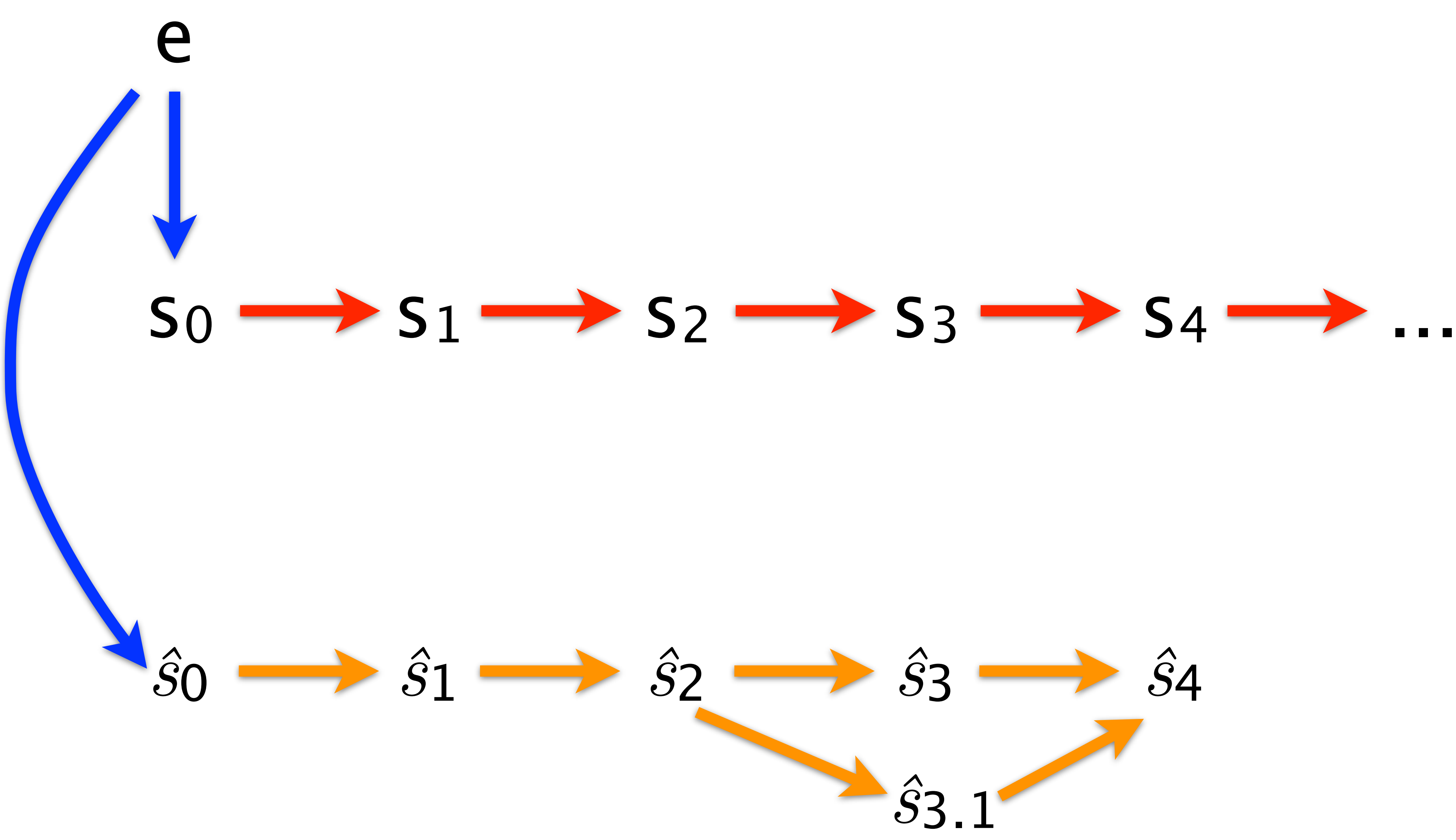
Analysis machine

ICFP'10/CACM'11



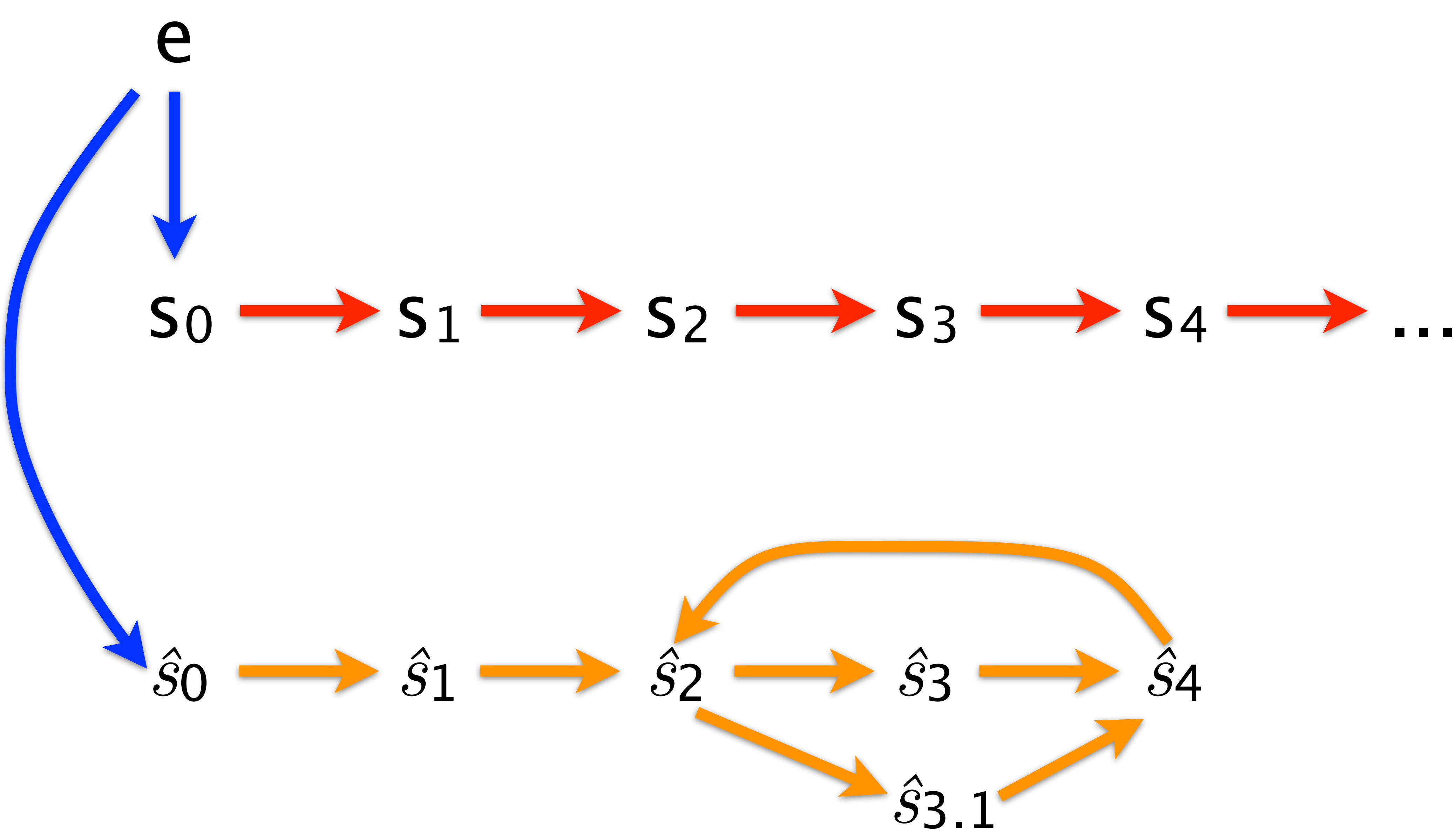
Analysis machine

ICFP'10/CACM'11



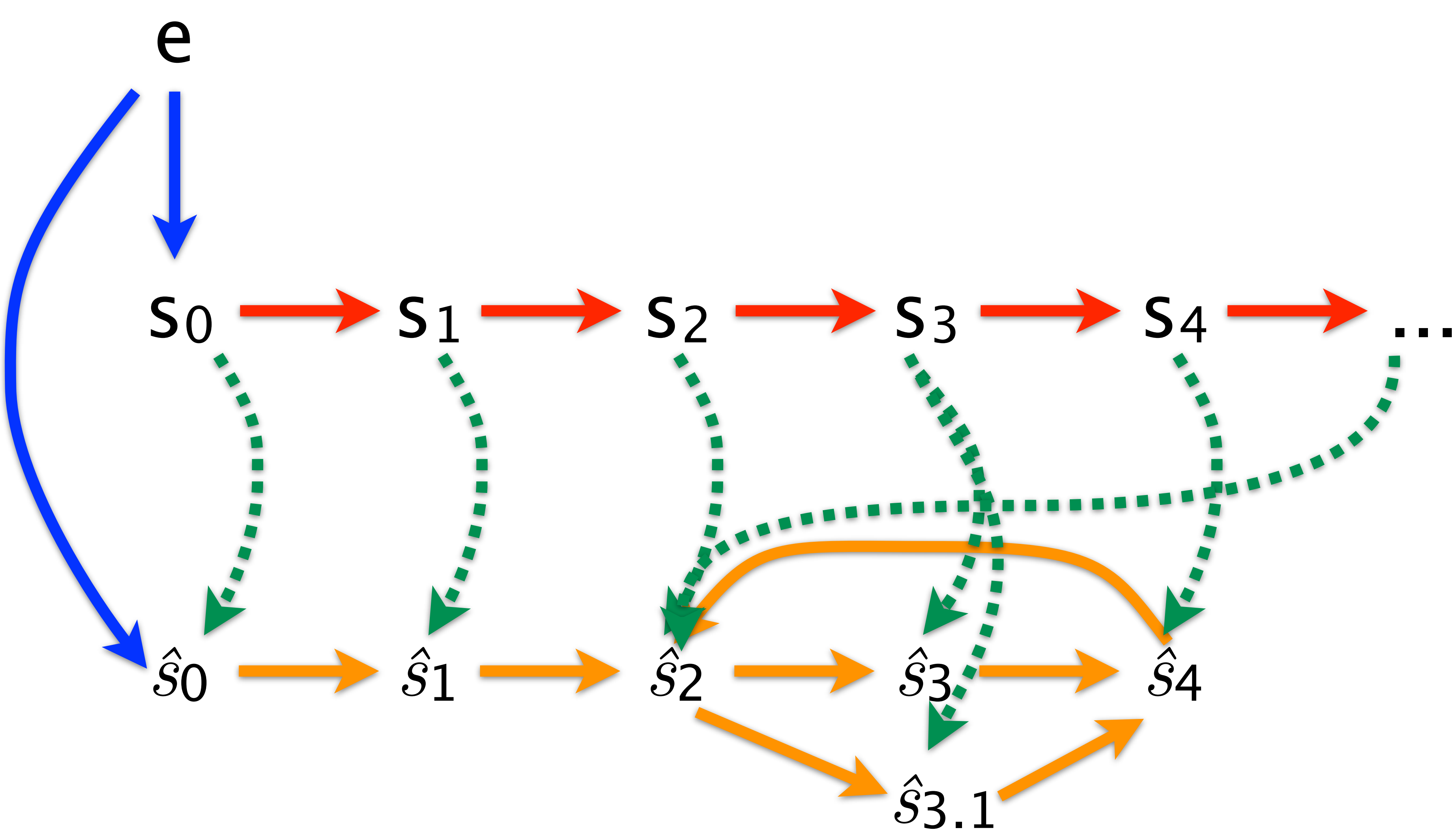
Analysis machine

ICFP'10/CACM'11



Analysis machine

ICFP'10/CACM'11



Theorem: The analysis simulates the machine.

Key idea:

Deterministic state transition system with an infinite state space.

Non-deterministic state transition system with a finite state space.

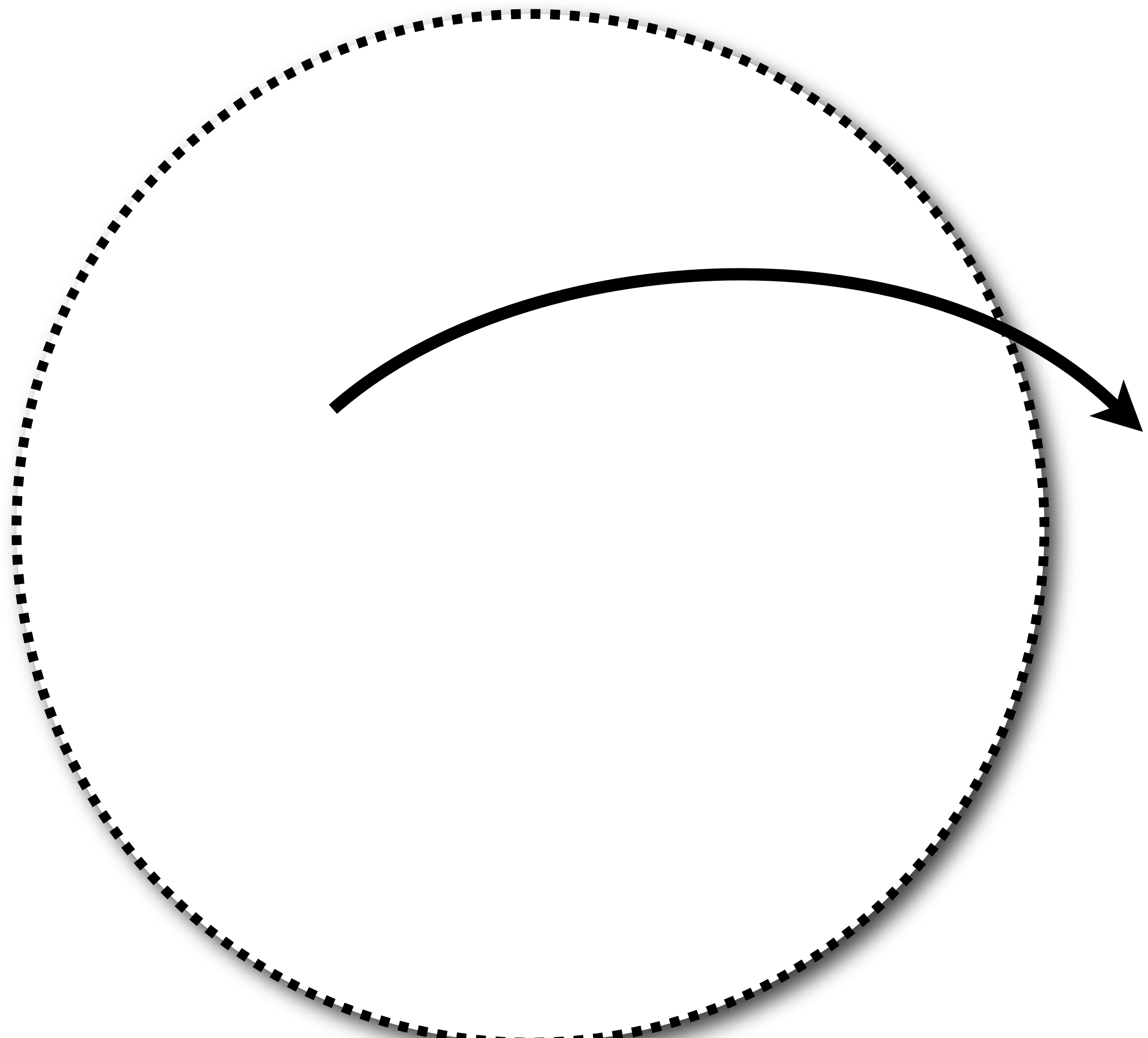
ICFP'10/CACM'11

Key idea:

Deterministic state transition system with an infinite state space.

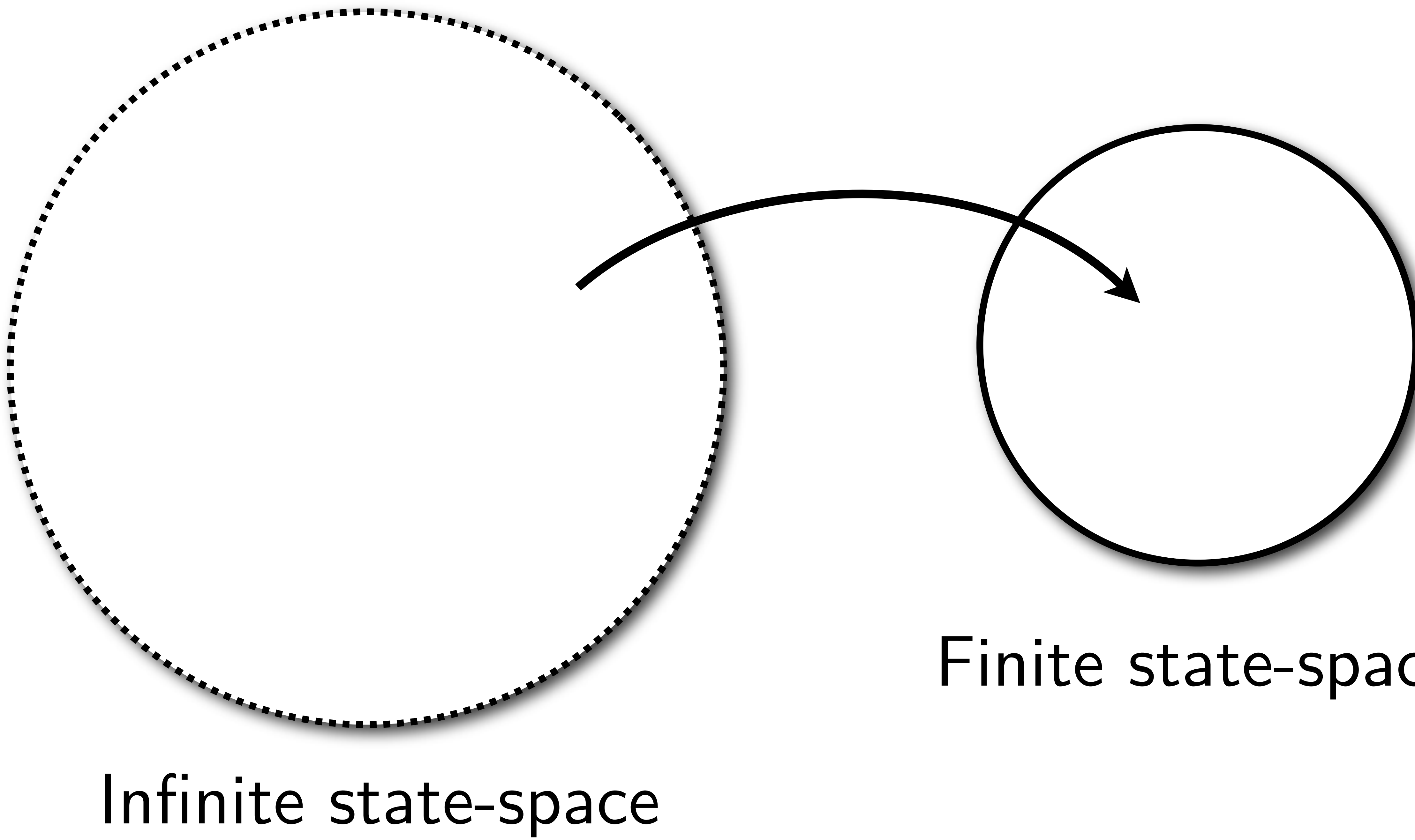
Non-deterministic state transition system with a finite state space.

Program analysis...



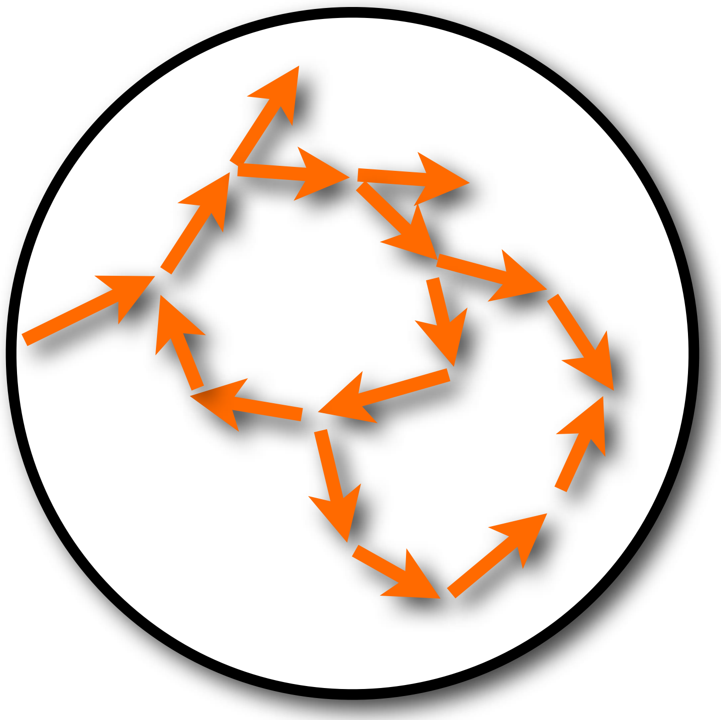
Infinite state-space

Program analysis...



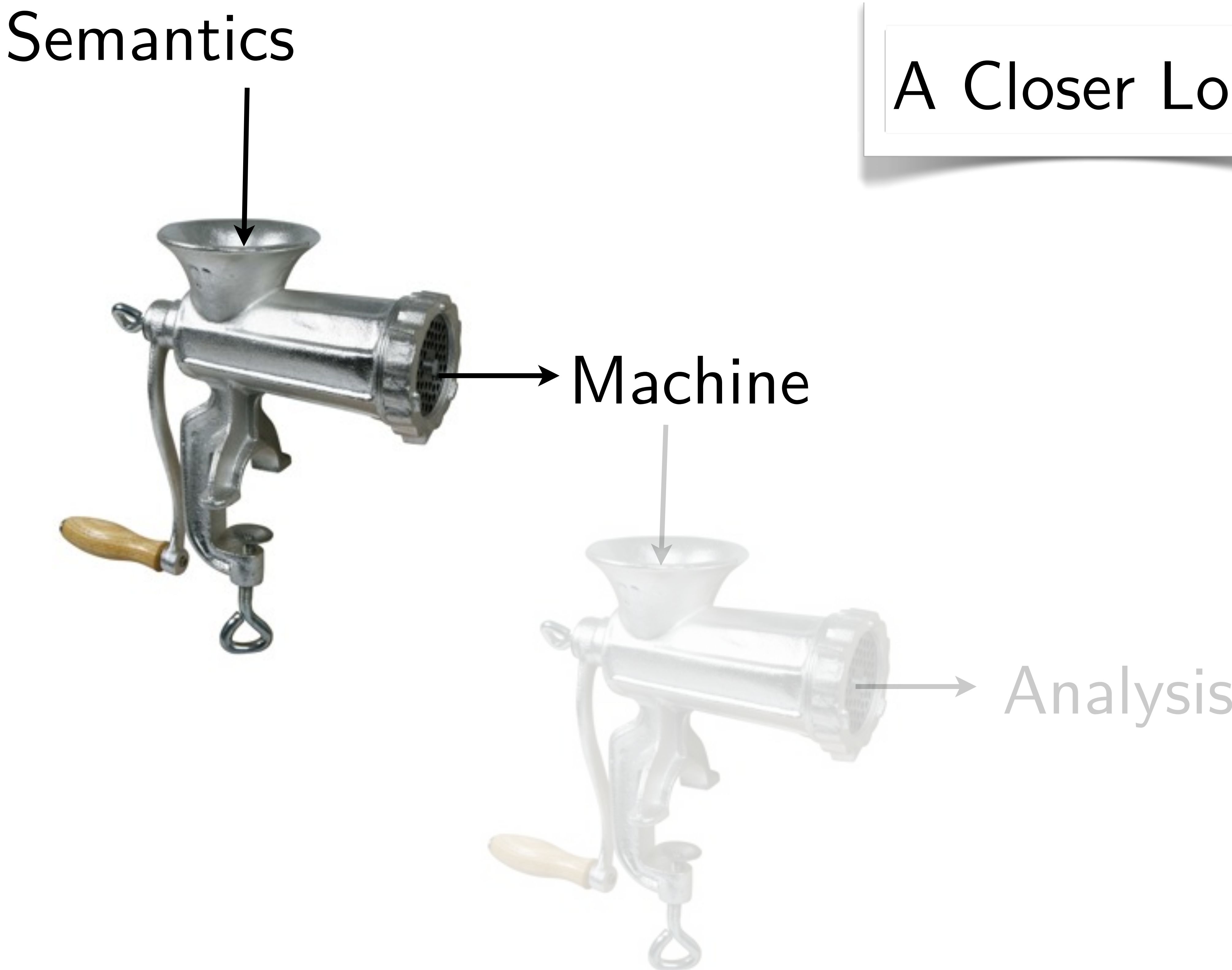
...is bounded graph search.

…is bounded graph search.



Finite state-space

A Closer Look



Reduction semantics:

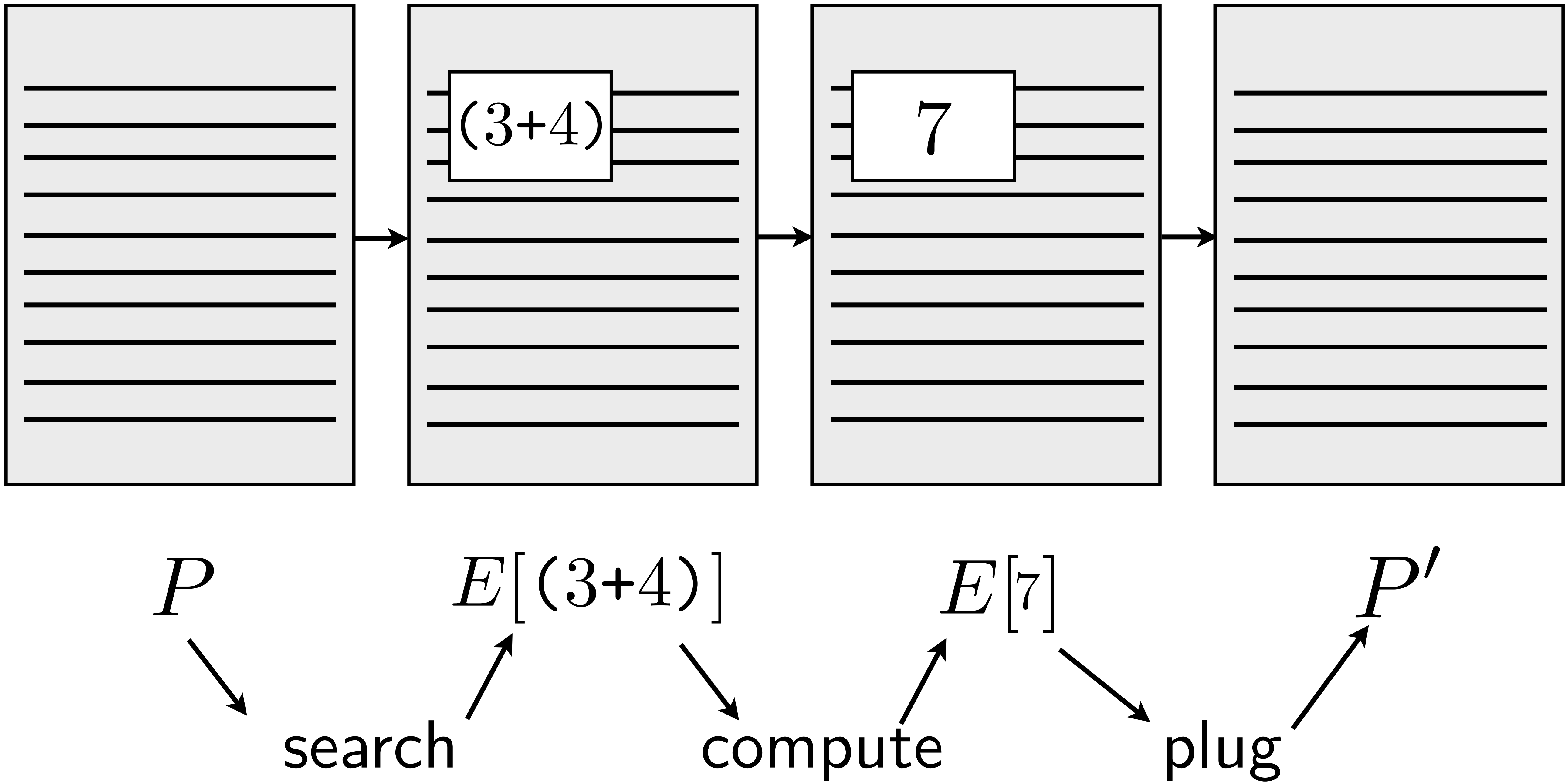
Syntax: $e ::= n \mid x \mid (e\text{+}e)$

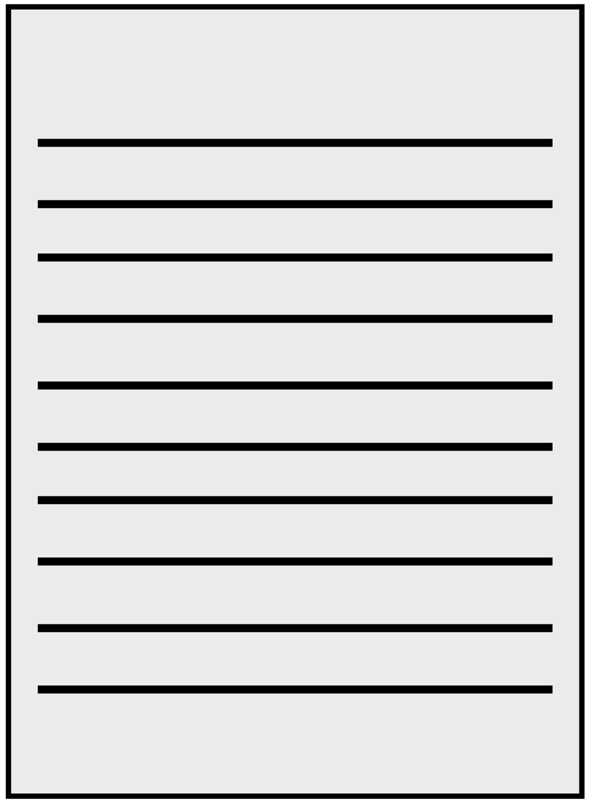
Reduction:

$$(n\text{+}m) \to n + m$$
$$x \to n \text{ where } \rho(x) = n$$

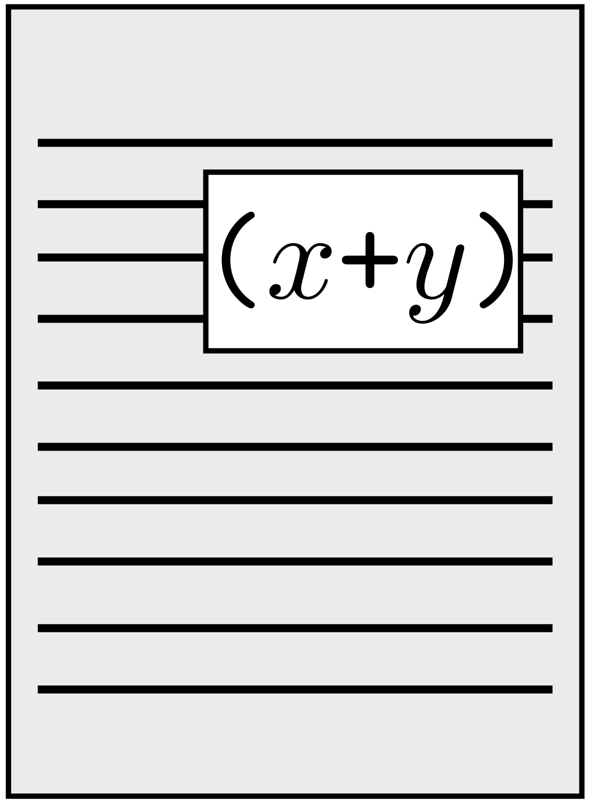
Eval. Contexts: $E ::= [\,] \mid (E\text{+}e) \mid (n\text{+}E)$

Reduction semantics:

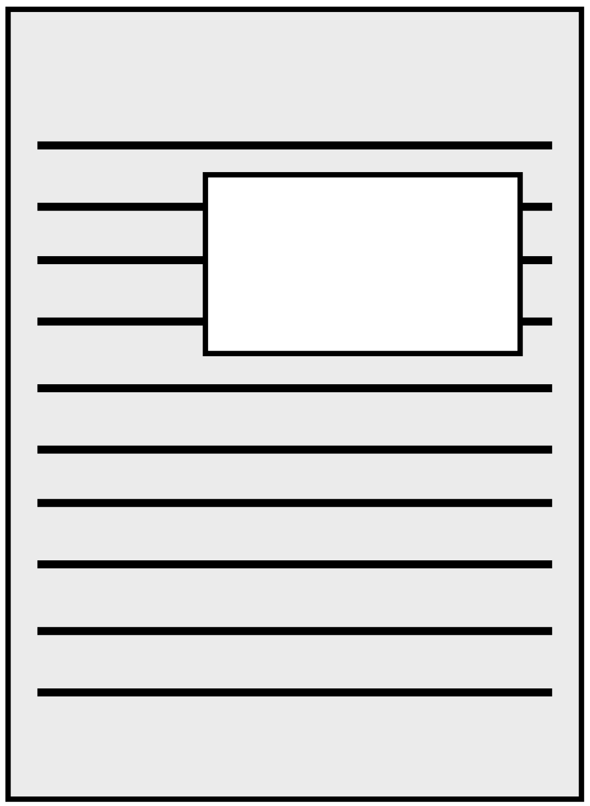




P

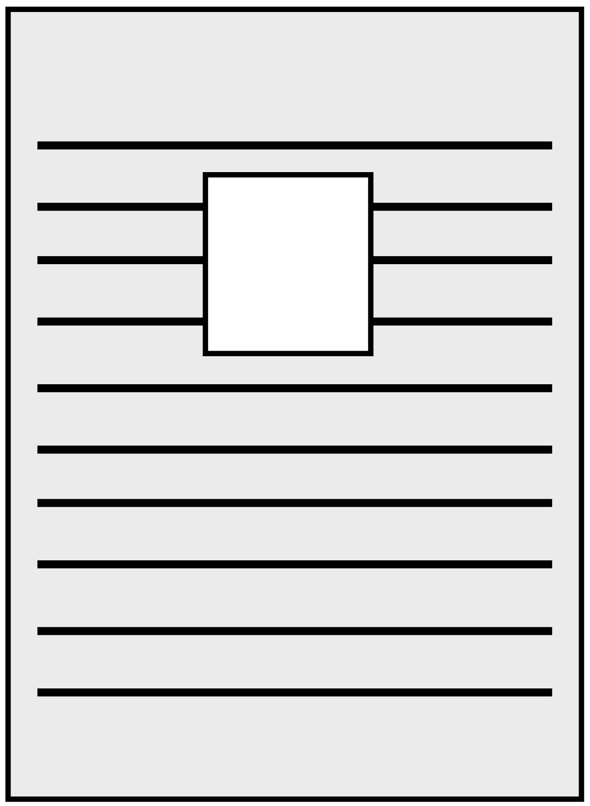


$$E[(x+y)]$$



$(x+y)$

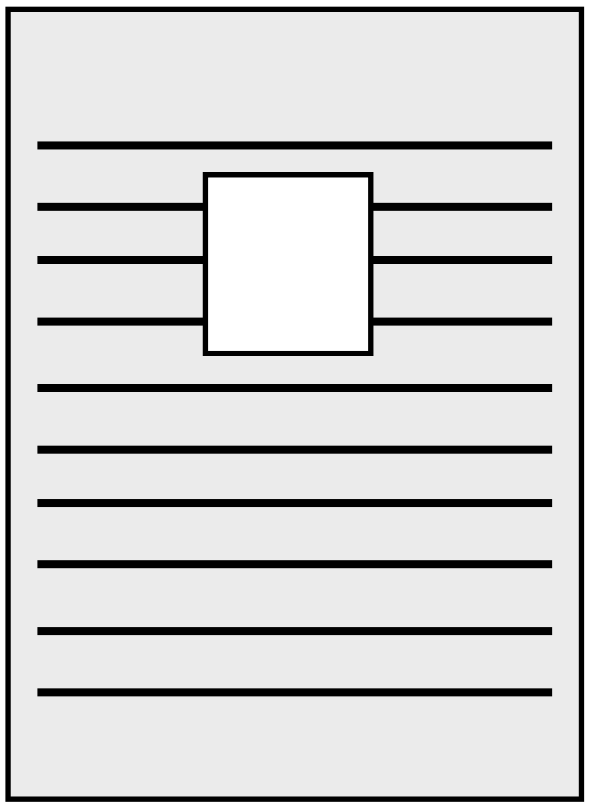
$E[\ ]$ $(x+y)$



x

$E[([\,]+y)]$

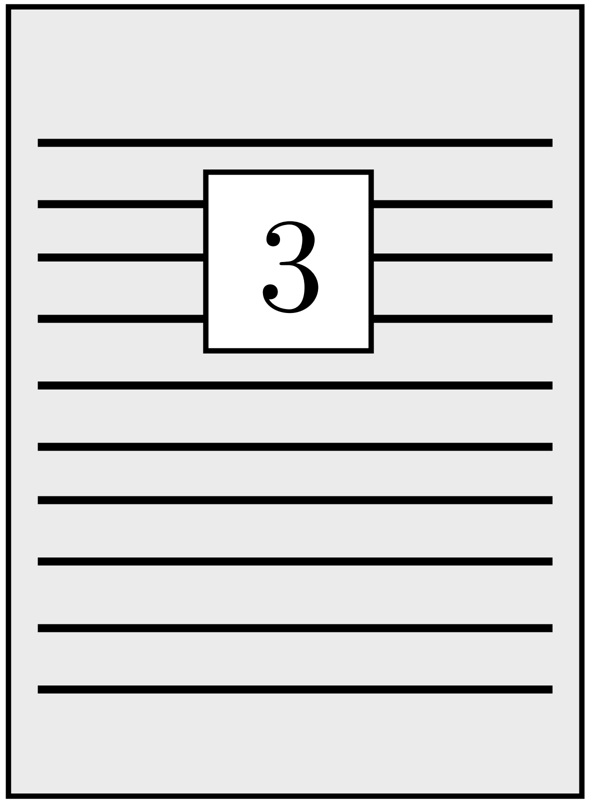
$x \to 3$



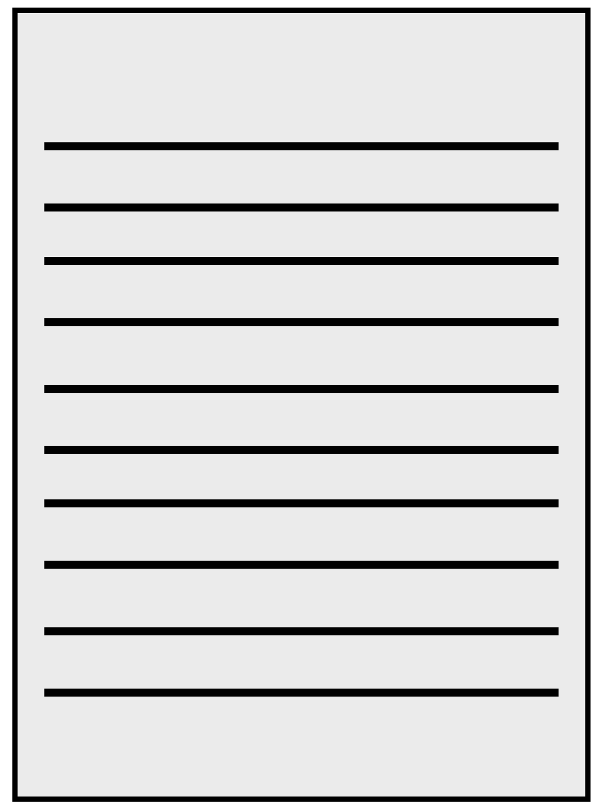
3

$$E[([\ ]+y)]$$

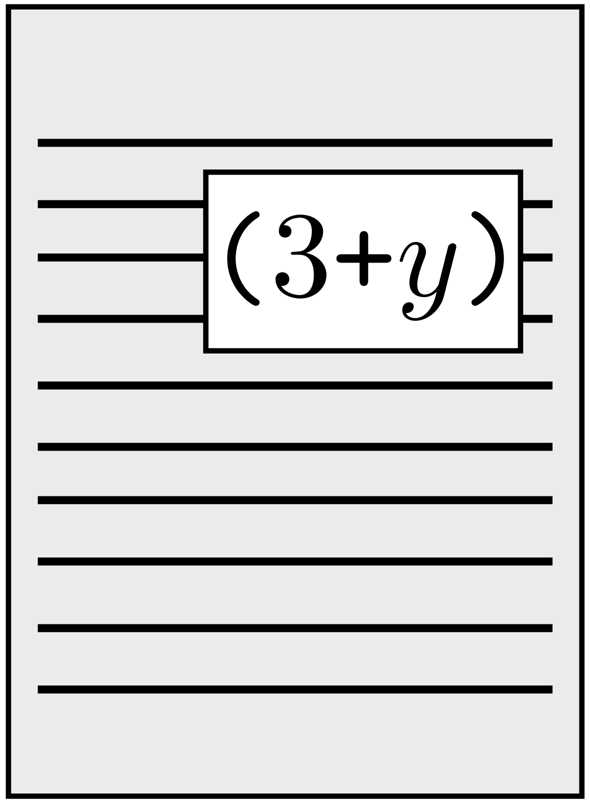
$$x \to 3$$



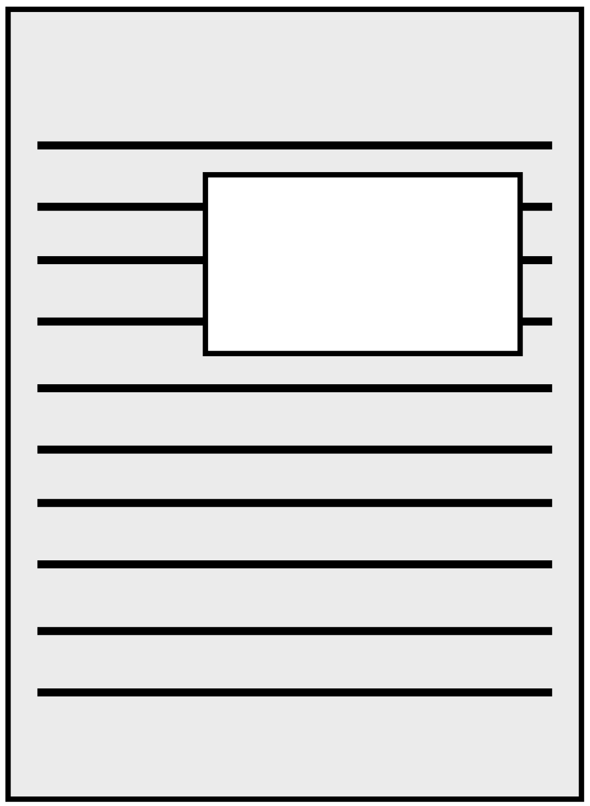
$$E[(\boxed{3}+y)]$$



P'

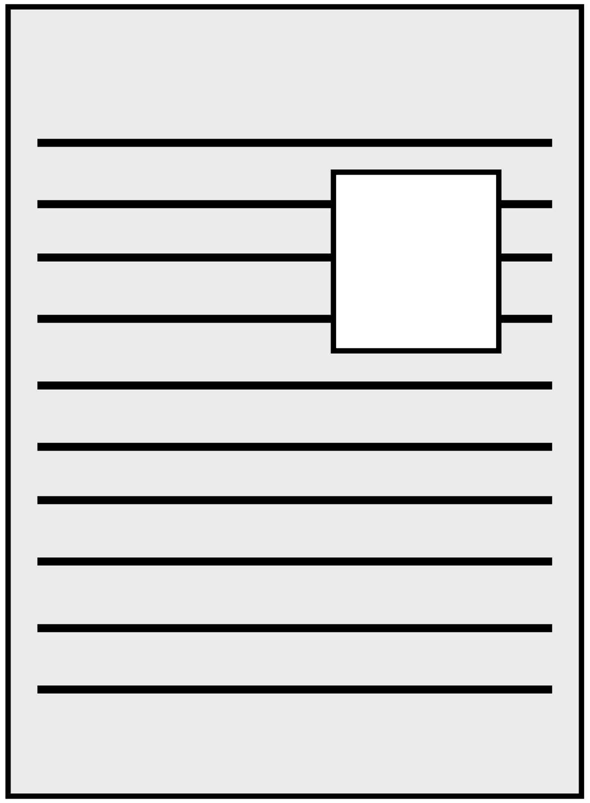


$$E[(3+y)]$$



$(3+y)$

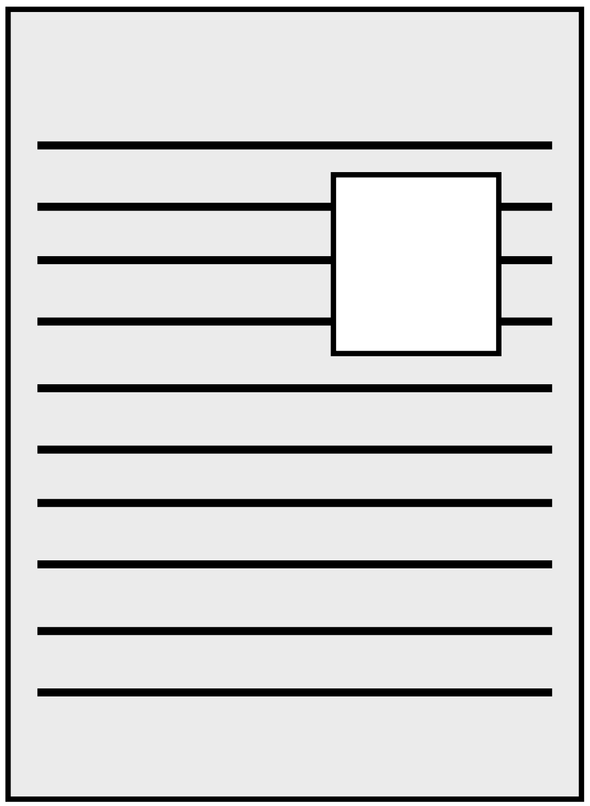
$E[\ ]$ $(3+y)$



y

$E[(3+[\ ])]$

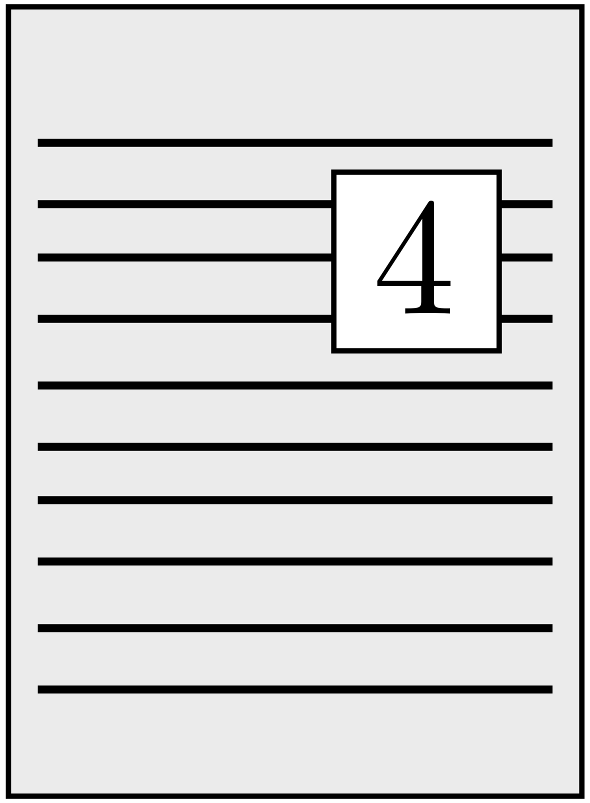
$y \to 4$



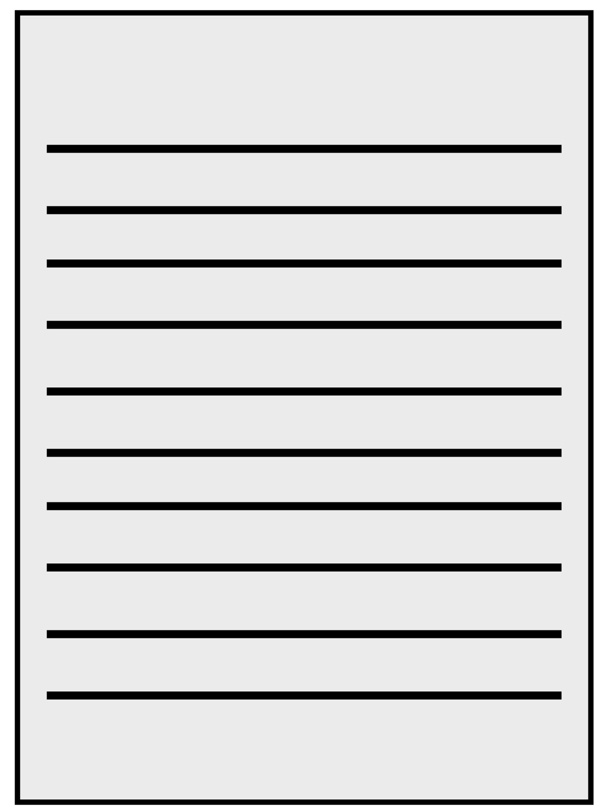
4

$E[\,(3\mathbf{+}[\;])\,]$

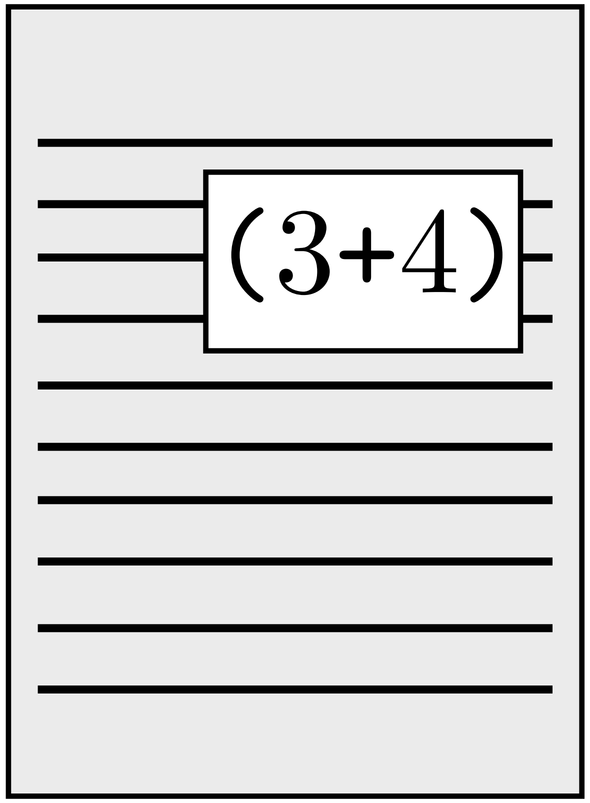
$y \to 4$



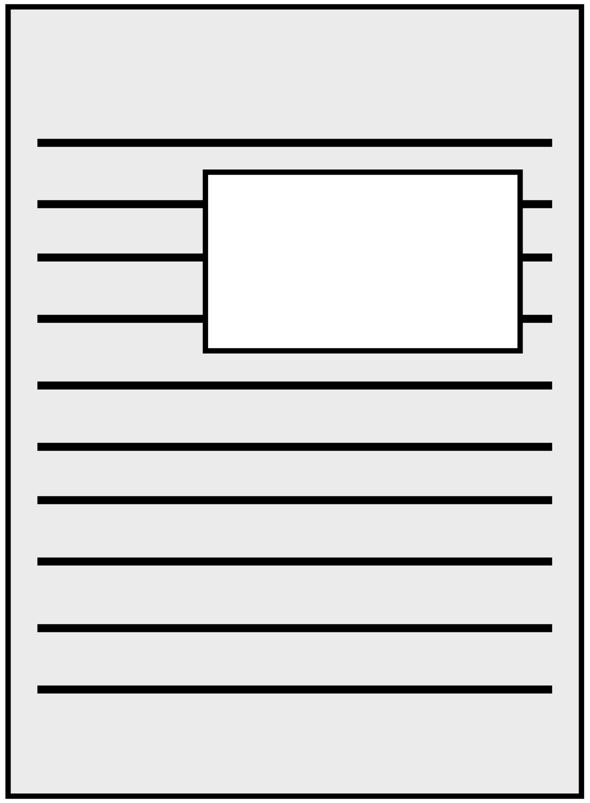
$$E[\,(3+[4])\,]$$



P''



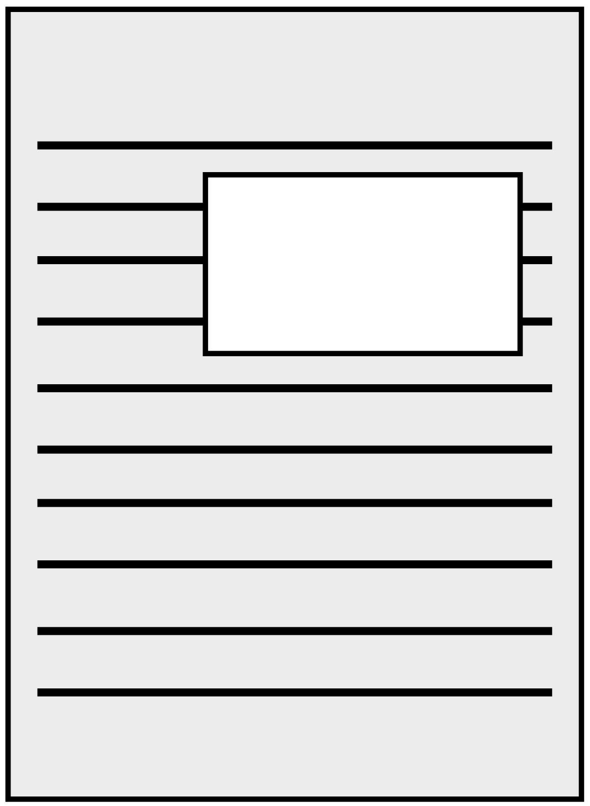
$$E[(3+4)]$$



$$(3+4)$$

$$E[\;]$$

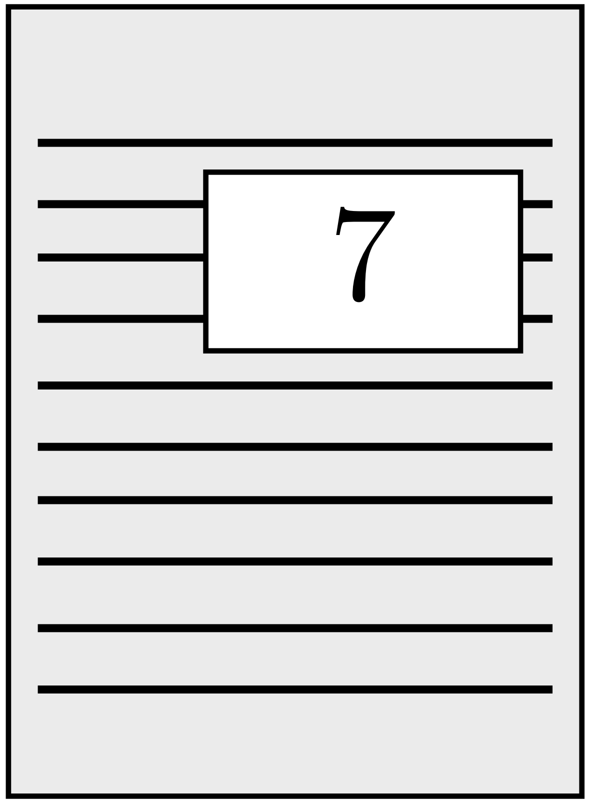
$$(3+4) \to 7$$



$$7$$

$$E[\;]$$

$$(3+4) \to 7$$



$$E[7]$$

Reduction semantics:

Syntax: $$e ::= n \mid x \mid (e\texttt{+}e)$$

Reduction:
$$(n\texttt{+}m) \to n + m$$
$$x \to n \text{ where } \rho(x) = n$$

Eval. Contexts: $$E ::= [\;] \mid (E\texttt{+}e) \mid (n\texttt{+}E)$$

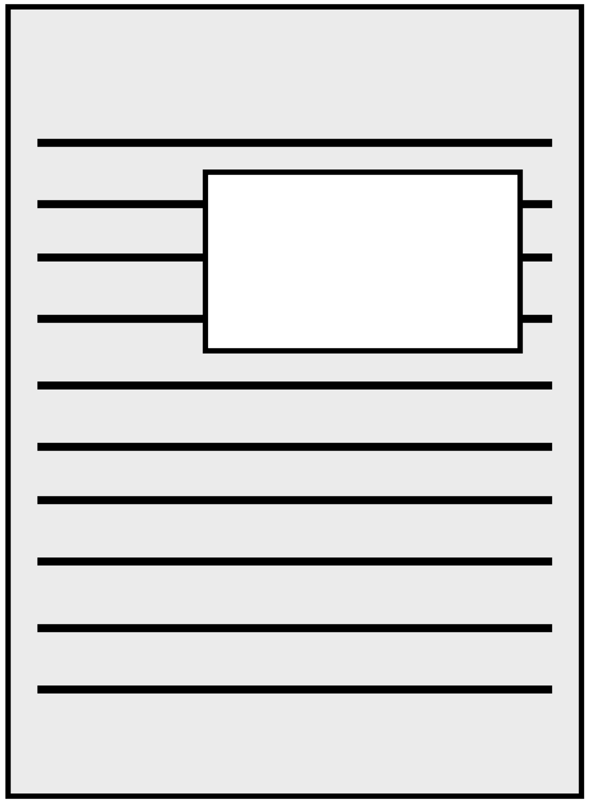
Reduction semantics:

Syntax: $$e ::= n \mid x \mid (e\texttt{+}e)$$

Reduction:
$$(n\texttt{+}m) \to n + m$$
$$x \to n \text{ where } \rho(x) = n$$

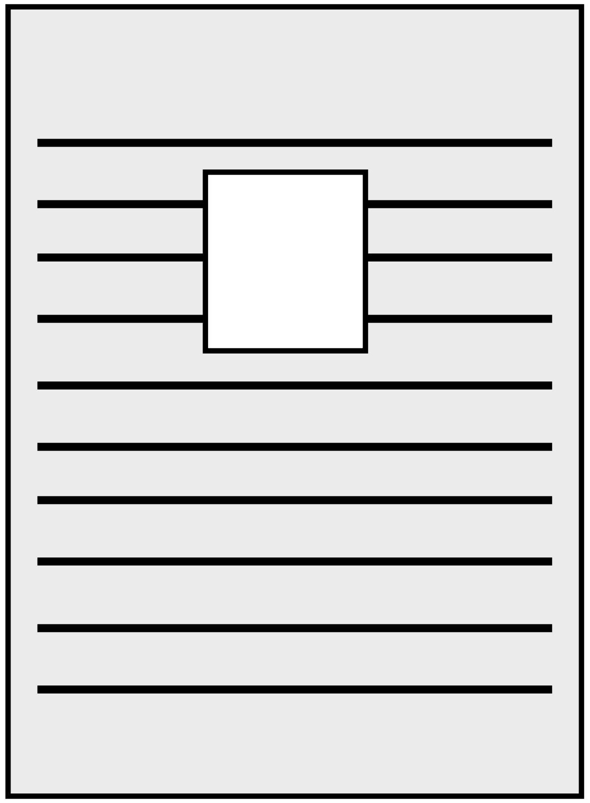
Eval. Contexts: $$E ::= [\,] \mid (E\texttt{+}e) \mid (n\texttt{+}E)$$

Continuations: $$C ::= \mathbf{c_0} \mid \mathbf{c_1}(e, C) \mid \mathbf{c_2}(n, C)$$



$$(x+y)$$

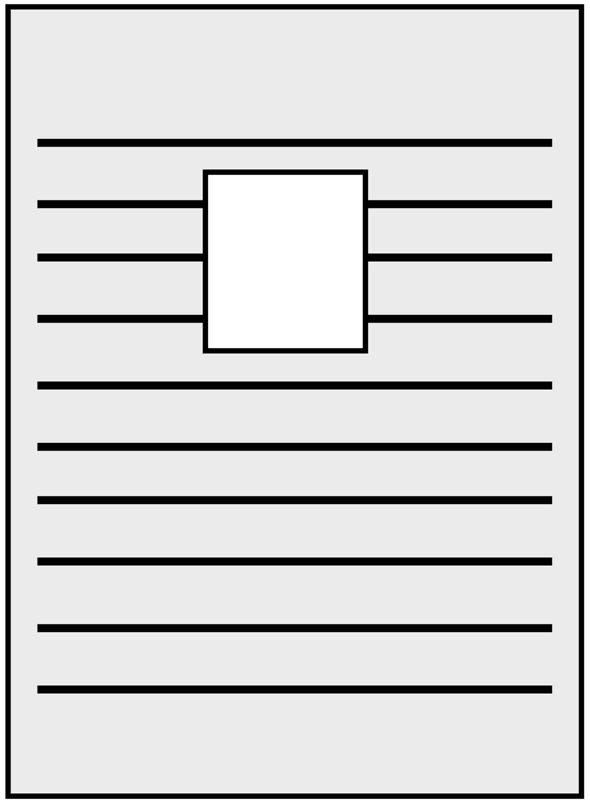
$$C \qquad (x+y)$$



x

$\mathbf{c_1}(C, y)$

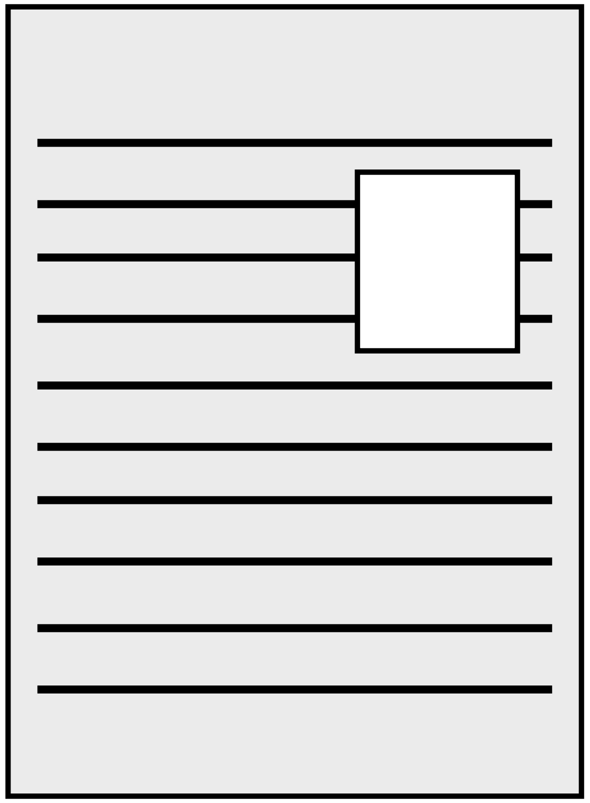
$x \to 3$



3

$$\mathbf{c_1}(C, y)$$

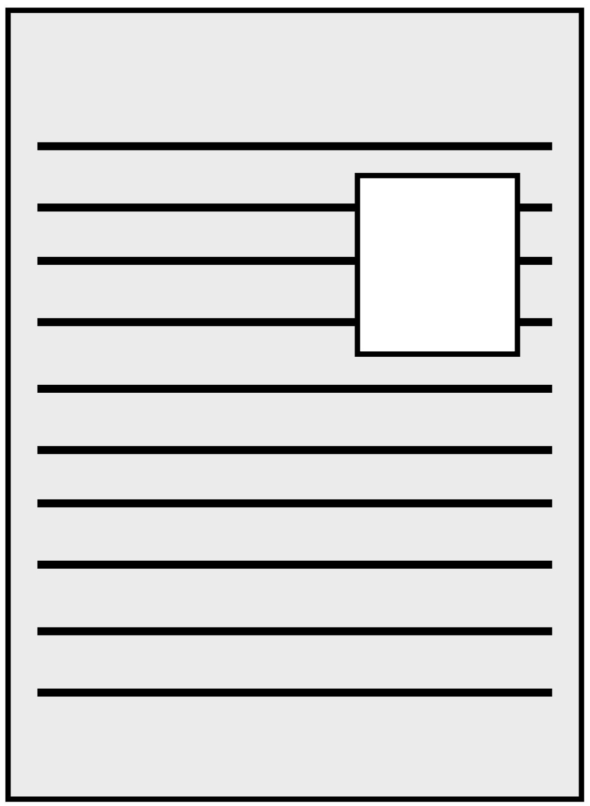
$$x \to 3$$



y

$\mathbf{c_2}(C, 3)$

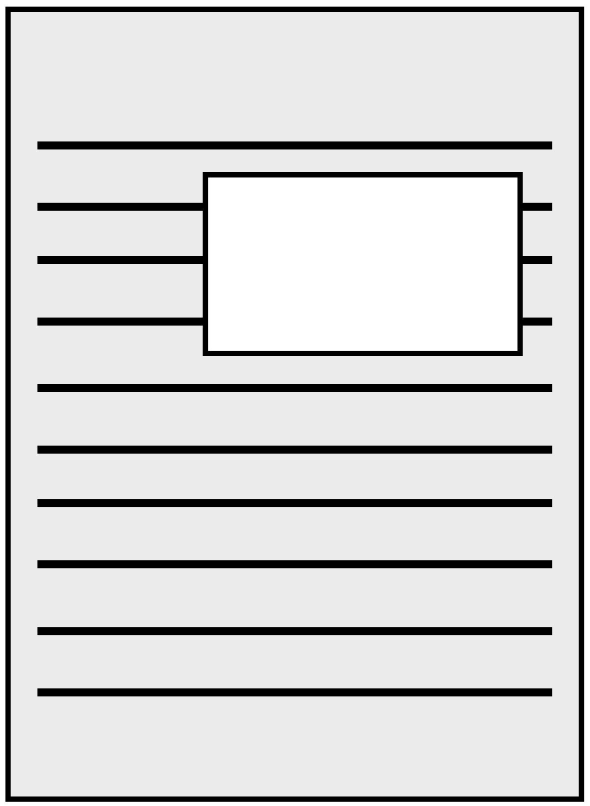
$y \to 4$



$$\mathbf{c_2}(C, 3)$$

$$4$$

$$y \rightarrow 4$$



C

$(3+4)$

$(3+4) \to 7$

Stack machine:

$$
\begin{array}{lcll}
\langle\,\texttt{(}e_1\texttt{+}e_2\texttt{)}\,, C\rangle & \rightarrow & \langle e_1, \mathbf{c_1}(C, e_2)\rangle & \\
\langle x, C\rangle & \rightarrow & \langle n, C\rangle & \text{where } \rho(x) = n \\
\langle n, \mathbf{c_1}(C, e)\rangle & \rightarrow & \langle e, \mathbf{c_2}(n, C)\rangle & \\
\langle m, \mathbf{c_2}(n, C)\rangle & \rightarrow & \langle n + m, C\rangle &
\end{array}
$$

Correctness:

$$P \rightarrow^\star n \iff \langle P, \mathbf{c_0}\rangle \rightarrow^\star \langle n, \mathbf{c_0}\rangle$$

A Closer Look

$$e ::= n \mid x \mid (e\texttt{+}e)$$

$$(n\texttt{+}m) \rightarrow n + m$$

$$x \rightarrow n \text{ where } \rho(x) = n$$

$$E ::= [\,] \mid (E\texttt{+}e) \mid (n\texttt{+}E)$$

$$\begin{array}{lcl}
\langle (e_1\texttt{+}e_2), C \rangle & \rightarrow & \langle e_1, \mathbf{c_1}(C, e_2) \rangle \\
\langle x, C \rangle & \rightarrow & \langle n, C \rangle \\
\langle n, \mathbf{c_1}(C, e) \rangle & \rightarrow & \langle e, \mathbf{c_2}(n, C) \rangle \\
\langle m, \mathbf{c_2}(n, C) \rangle & \rightarrow & \langle n + m, C \rangle
\end{array}$$

Analysis

JavaScript

Syntax:

$$c ::= num \mid str \mid bool \mid \texttt{undefined} \mid \texttt{null}$$

$$v ::= c \mid \texttt{func(}\vec{x}\texttt{) \{ return } e \texttt{ \}} \mid \texttt{\{ } str \texttt{ . } v \ldots \texttt{ \}}$$

$$p ::= str \texttt{:} e$$

$$\begin{aligned} e ::= &\ x \mid v \mid \texttt{\{ } \vec{p} \texttt{ \}} \mid \texttt{let (} x \texttt{ = } e \texttt{) } e \mid e\texttt{(}\vec{e}\texttt{)} \\ \mid &\ e\texttt{[}e\texttt{]} \mid e\texttt{[}e\texttt{] = } e \mid \texttt{del } e\texttt{[}e\texttt{]} \end{aligned}$$

JavaScript

Reductions:

$$\texttt{let}\ (x = v)\ e \rightarrow [v/x]e$$

$$(\texttt{func}(\vec{x})\ \texttt{\{ return}\ e\ \texttt{\}})(\vec{v}) \rightarrow [\vec{v}/\vec{x}]e$$

$$\texttt{\{}\ \ldots str_i : v \ldots \texttt{\}}[str_i] \rightarrow v$$

$$\frac{str_x \notin (str_1 \ldots)}{\texttt{\{}\ str_1 : v_1 \ldots \texttt{\}}[str_x] \rightarrow \texttt{undefined}}$$

$$\texttt{\{}\ \ldots str_i : v_i \ldots \texttt{\}}[str_i] = v \rightarrow \texttt{\{}\ \ldots str_i : v \ldots \texttt{\}}$$

$$\frac{str_x \notin (str_1 \ldots)}{\texttt{\{}\ str_1 : v_1 \ldots \texttt{\}}[str_x] = v \rightarrow \texttt{\{}\ str_x : v, str_1 : v_1 \ldots \texttt{\}}}$$

$$\texttt{del}\ \texttt{\{}\ \ldots str_i : v_i \ldots \texttt{\}}[str_i] \rightarrow \texttt{\{}\ \ldots \texttt{\}}$$

$$\frac{str_x \notin (str_1 \ldots)}{\texttt{del}\ \texttt{\{}\ str_1 : v_1 \ldots \texttt{\}}[str_x] \rightarrow \texttt{\{}\ str_1 : v_1 \ldots \texttt{\}}}$$

JavaScript

Eval. Contexts:

$$
\begin{aligned}
E ::= \; & [\,] \\
\mid \; & \texttt{let (}x\texttt{ = }E\texttt{)}\; e \\
\mid \; & E\texttt{(}\vec{e}\texttt{)} \\
\mid \; & v\texttt{(}e \ldots E, v \ldots\texttt{)} \\
\mid \; & \texttt{\{}\; str\texttt{:}v \ldots str\texttt{:}E, \vec{p}\;\texttt{\}} \\
\mid \; & E\texttt{[}e\texttt{]} \\
\mid \; & v\texttt{[}E\texttt{]} \\
\mid \; & E\texttt{[}e\texttt{] = }e \\
\mid \; & v\texttt{[}E\texttt{] = }e \\
\mid \; & v\texttt{[}v\texttt{] = }E \\
\mid \; & \texttt{del}\; E\texttt{[}e\texttt{]} \\
\mid \; & \texttt{del}\; v\texttt{[}E\texttt{]}
\end{aligned}
$$

Continuations:

$$
\begin{aligned}
C ::= \; & \mathbf{c_1} \\
\mid \; & \mathbf{c_2}(x, e, \rho, C) \\
\mid \; & \mathbf{c_3}(\vec{e}, \rho, C) \\
\mid \; & \mathbf{c_4}(c, \vec{c}, \vec{e}, \rho, C) \\
\mid \; & \mathbf{c_5}(str, \vec{q}, \vec{p}, \rho, C) \\
\mid \; & \mathbf{c_6}(e, \rho, C) \\
\mid \; & \mathbf{c_7}(c, C) \\
\mid \; & \mathbf{c_8}(e, e, \rho, C) \\
\mid \; & \mathbf{c_9}(c, e, \rho, C) \\
\mid \; & \mathbf{c_{10}}(c, c, C) \\
\mid \; & \mathbf{c_{11}}(e, \rho, C) \\
\mid \; & \mathbf{c_{12}}(c, C)
\end{aligned}
$$

Machine:

$$
\begin{array}{lcll}
\langle\langle x, \rho\rangle, \sigma, C\rangle & \rightarrow & \langle c, \sigma, C\rangle & \text{where } \sigma(\rho(x)) = c \\
\langle c, \sigma, \mathbf{c_2}(x, e, \rho, C)\rangle & \rightarrow & \langle\langle e, \rho[x \mapsto a]\rangle, \sigma[a \mapsto c], C\rangle & \text{where } a \text{ is fresh} \\
\langle c, \sigma, \mathbf{c_4}(\langle \mathtt{func(}\vec{x}\mathtt{)\ \{\ return}\ e\ \mathtt{\}}, \rho\rangle, c_n \ldots c_0, , \rho', C)\rangle & \rightarrow & \langle\langle e, \rho[\vec{x} \mapsto \vec{a}]\rangle, \sigma[\vec{a} \mapsto c_0 \ldots c_n c], C\rangle & \text{where } \vec{a} \text{ are fresh} \\
\langle\langle str_i, \rho\rangle, \sigma, \mathbf{c_7}(\mathtt{\{}\ \ldots str_i \mathtt{.} c_i \ldots\ \mathtt{\}}, C)\rangle & \rightarrow & \langle c_i, \sigma, C\rangle & \\
\langle\langle str_x, \rho\rangle, \sigma, \mathbf{c_7}(\mathtt{\{}\ str_1 \mathtt{.} c_1 \ldots\ \mathtt{\}}, C)\rangle & \rightarrow & \langle \mathtt{undefined}, \sigma, C\rangle & \text{where } str_x \notin (str_1 \ldots) \\
\langle c, \sigma, \mathbf{c_{10}}(\mathtt{\{}\ \ldots str_i \mathtt{.} c_i \ldots\ \mathtt{\}}, \langle str_i, \rho\rangle, C)\rangle & \rightarrow & \langle \mathtt{\{}\ \ldots str_i \mathtt{.} c \ldots\ \mathtt{\}}, \sigma, C\rangle & \\
\langle c, \sigma, \mathbf{c_{10}}(\mathtt{\{}\ str_1 \mathtt{.} c_1 \ldots\ \mathtt{\}}, \langle str_x, \rho\rangle, C)\rangle & \rightarrow & \langle \mathtt{\{}\ str_x \mathtt{.} c, str_1 \mathtt{.} c_1 \ldots\ \mathtt{\}}, \sigma, C\rangle & \text{where } str_x \notin (str_1 \ldots) \\
\langle\langle str_i, \rho\rangle, \sigma, \mathbf{c_{12}}(\mathtt{\{}\ \ldots str_i \mathtt{.} c_i \ldots\ \mathtt{\}}, C)\rangle & \rightarrow & \langle \mathtt{\{}\ \ldots\ \mathtt{\}}, \sigma, C\rangle & \\
\langle\langle str_x, \rho\rangle, \sigma, \mathbf{c_{12}}(\mathtt{\{}\ str_1 \mathtt{.} c_1 \ldots\ \mathtt{\}}, C)\rangle & \rightarrow & \langle \mathtt{\{}\ str_1 \mathtt{.} c_1 \ldots\ \mathtt{\}}, \sigma, C\rangle & \text{where } str_x \notin (str_1 \ldots) \\
\langle c, \sigma, \mathbf{c_3}(e\vec{e}, \rho, C)\rangle & \rightarrow & \langle\langle e, \rho\rangle, \sigma, \mathbf{c_4}(c, , \vec{e}, \rho, C)\rangle & \\
\langle\langle \mathtt{func()\ \{\ return}\ e\ \mathtt{\}}, \rho\rangle, \sigma, \mathbf{c_3}(, \rho', C)\rangle & \rightarrow & \langle\langle e, \rho\rangle, \sigma, C\rangle & \\
\langle c, \sigma, \mathbf{c_5}(str, \vec{q}, , \rho, C)\rangle & \rightarrow & \langle \mathtt{\{}\ str \mathtt{:} c, \vec{q}\ \mathtt{\}}, C\rangle & \\
\langle c, \sigma, \mathbf{c_5}(str, \vec{q}, str_1 \mathtt{:} e_1\vec{p}, \rho, C)\rangle & \rightarrow & \langle\langle e_1, \rho\rangle, \sigma, \mathbf{c_5}(str_1, str \mathtt{:} c\vec{q}, \vec{p}, \rho, C)\rangle & \\
\langle\langle \mathtt{let\ (}x\ \mathtt{=}\ e_0\mathtt{)}\ e_1, \rho\rangle, \sigma, C\rangle & \rightarrow & \langle\langle e_0, \rho\rangle, \sigma, \mathbf{c_2}(x, e_1, \rho, C)\rangle & \\
\langle\langle e\mathtt{()}, \rho\rangle, \sigma, C\rangle & \rightarrow & \langle\langle e, \rho\rangle, \sigma, C\rangle & \\
\langle\langle e_0\mathtt{(}e\vec{e}\mathtt{)}, \rho\rangle, \sigma, C\rangle & \rightarrow & \langle\langle e_0, \rho\rangle, \sigma, \mathbf{c_3}(e\vec{e}, \rho, C)\rangle & \\
\langle\langle \mathtt{\{\ \}}, \rho\rangle, \sigma, C\rangle & \rightarrow & \langle \mathtt{\{\ \}}, \sigma, C\rangle & \\
\langle\langle \mathtt{\{}\ str_0 \mathtt{:} e_0\vec{p}\ \mathtt{\}}, \rho\rangle, \sigma, C\rangle & \rightarrow & \langle\langle e_0, \rho\rangle, \sigma, \mathbf{c_5}(str_0, , \vec{p}, \rho, C)\rangle & \\
\langle\langle e_0\mathtt{[}e_1\mathtt{]}, \rho\rangle, \sigma, C\rangle & \rightarrow & \langle\langle e_0, \rho\rangle, \sigma, \mathbf{c_6}(e_1, \rho, C)\rangle & \\
\langle\langle e_0\mathtt{[}e_1\mathtt{]}\ \mathtt{=}\ e_2, \rho\rangle, \sigma, C\rangle & \rightarrow & \langle\langle e_0, \rho\rangle, \sigma, \mathbf{c_8}(e_1, e_2, \rho, C)\rangle & \\
\langle\langle \mathtt{del}\ e_0\mathtt{[}e_1\mathtt{]}, \rho\rangle, \sigma, C\rangle & \rightarrow & \langle\langle e_0, \rho\rangle, \sigma, \mathbf{c_{11}}(e_1, \rho, C)\rangle &
\end{array}
$$

$$\langle e, C \rangle$$

Var→Addr

Addr→Value

$$\langle e, \rho, \sigma, C \rangle$$

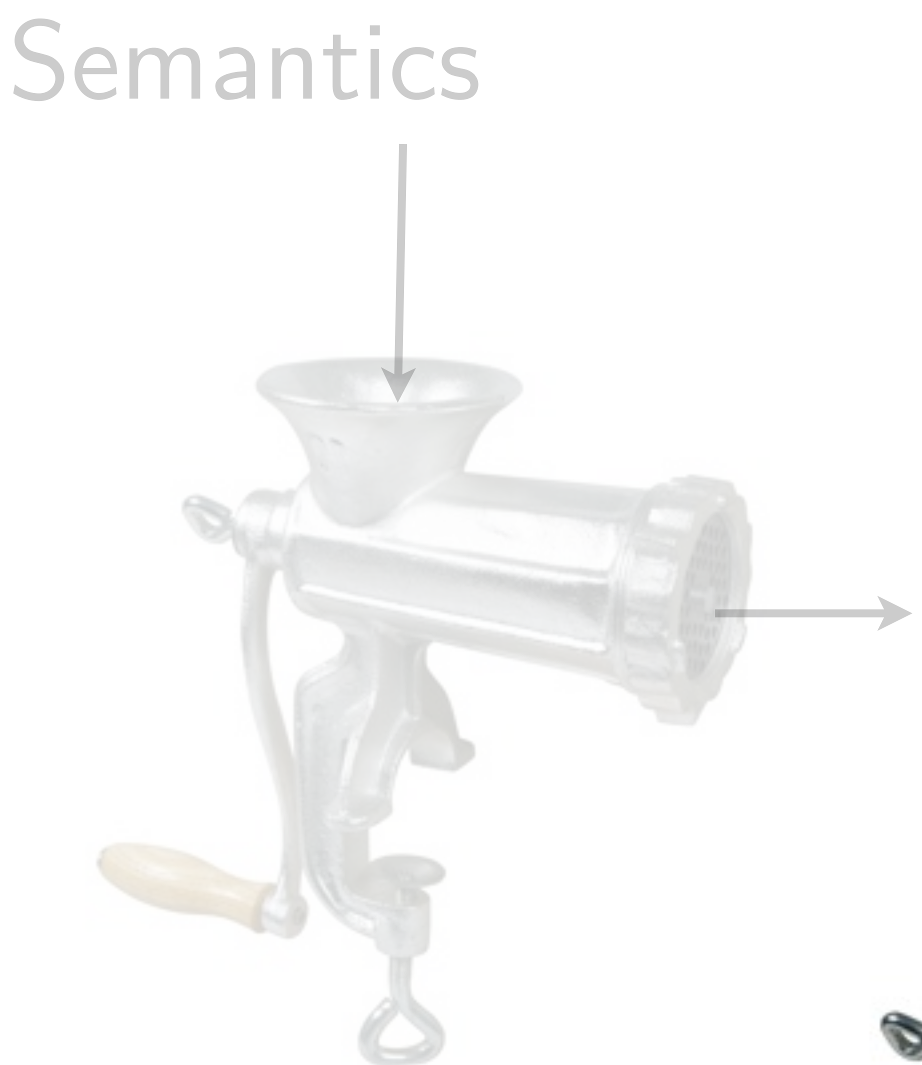
A Closer Look

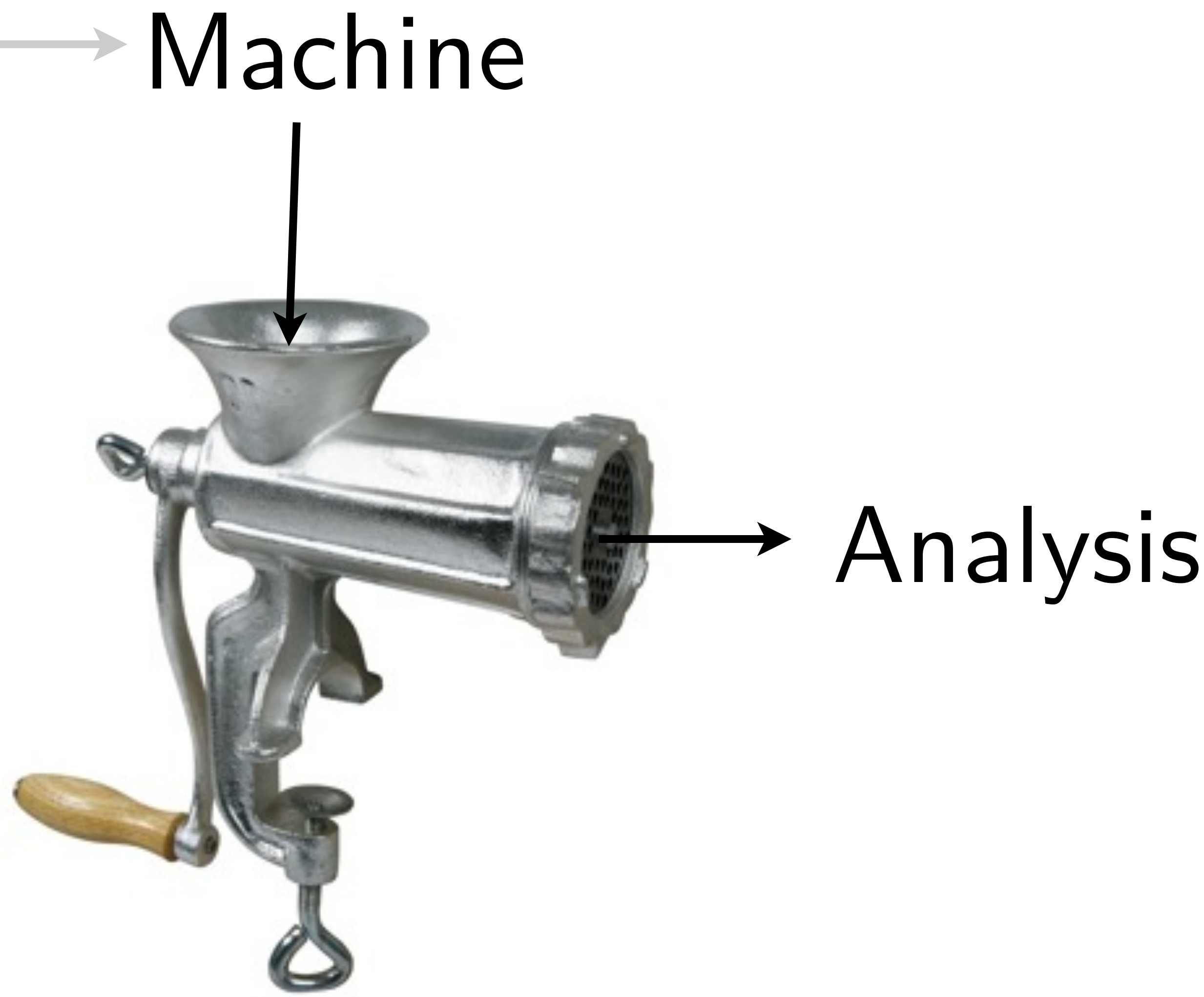
Semantics

Machine

Analysis

A Closer Look





Key idea:

Deterministic state transition system with an infinite state space.

Non-Deterministic state transition system with a finite state space.

Step 1:

$$\langle e, \rho, \sigma, C \rangle$$

Move continuations into heap.

Step 1:

$$\langle e, \rho, \sigma, C \rangle$$

Move continuations into heap.

Step 1:

$$\langle e, \rho, \sigma, a \rangle$$

$$\sigma(a) = C$$

Step 1: Var$\to$Addr Addr$\to$Value$+$Cont

$$\langle e, \rho, \sigma, a \rangle$$

$$\sigma(a) = C$$

Step 1:

$$\langle (e_1 \texttt{+} e_2), \rho, \sigma, C\rangle \quad \to \quad \langle e_1, \rho, \sigma, \mathbf{c_1}(C, e_2)\rangle$$

$$\Longleftrightarrow$$

$$\langle (e_1 \texttt{+} e_2), \rho, \sigma, a\rangle \quad \to \quad \langle e_1, \rho, \sigma[a' \mapsto \mathbf{c_1}(a, e_2)], a'\rangle$$

Step 2:

$$\langle e, \rho, \hat{\sigma}, a \rangle$$

$$\hat{\sigma}(a) \ni C$$

Step 2:

Var→Addr

Addr→$\wp$(Value+Cont)

$$\langle e, \rho, \hat{\sigma}, a \rangle$$

$$\hat{\sigma}(a) \ni C$$

Step 1:

$$\langle (e_1 + e_2), \rho, \sigma, C\rangle \quad \rightarrow \quad \langle e_1, \rho, \sigma, \mathbf{c_1}(C, e_2)\rangle$$

$$\Longleftrightarrow$$

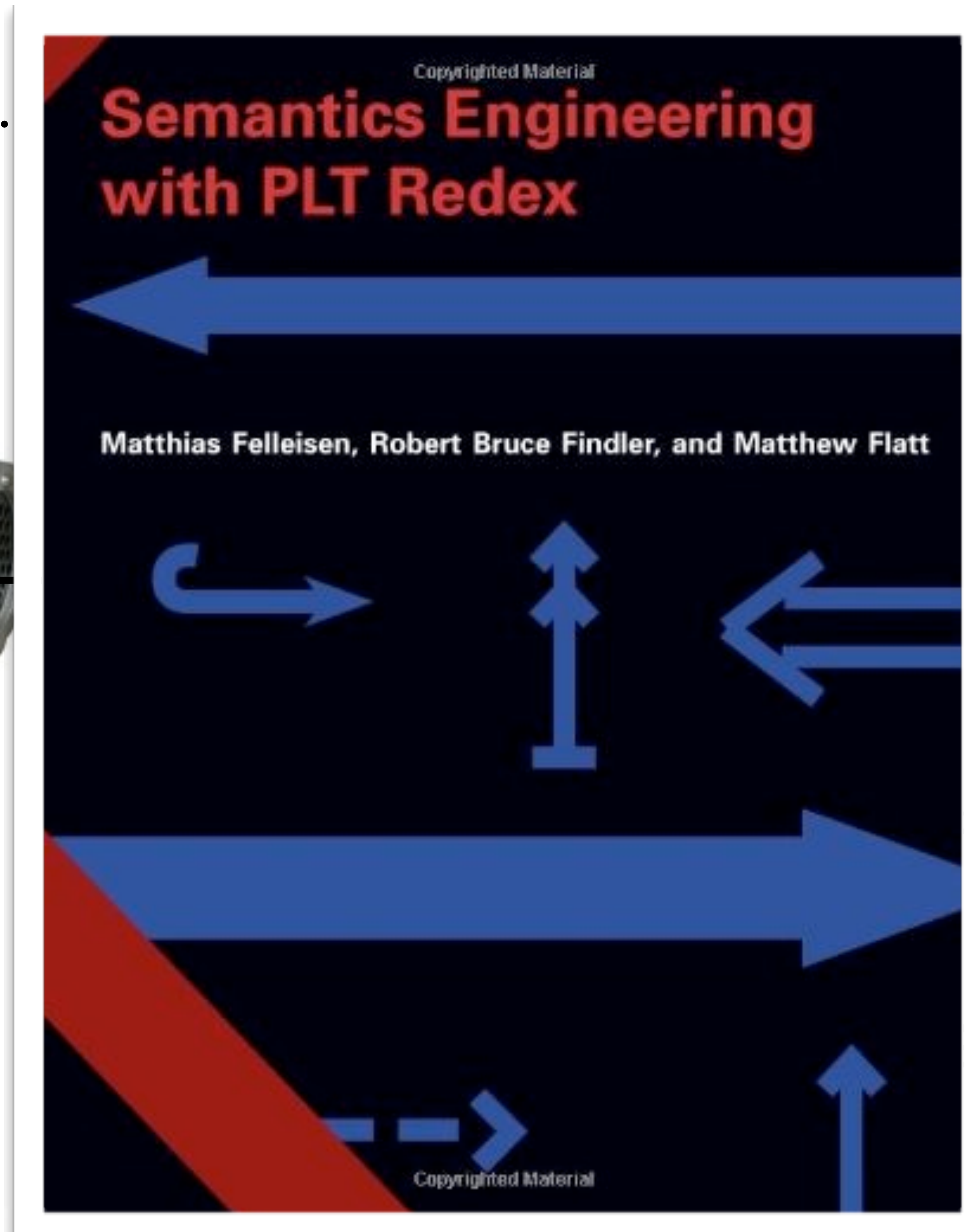
$$\langle (e_1 + e_2), \rho, \sigma, a\rangle \quad \rightarrow \quad \langle e_1, \rho, \sigma[a' \mapsto \mathbf{c_1}(a, e_2)], a'\rangle$$

$$\Rightarrow$$

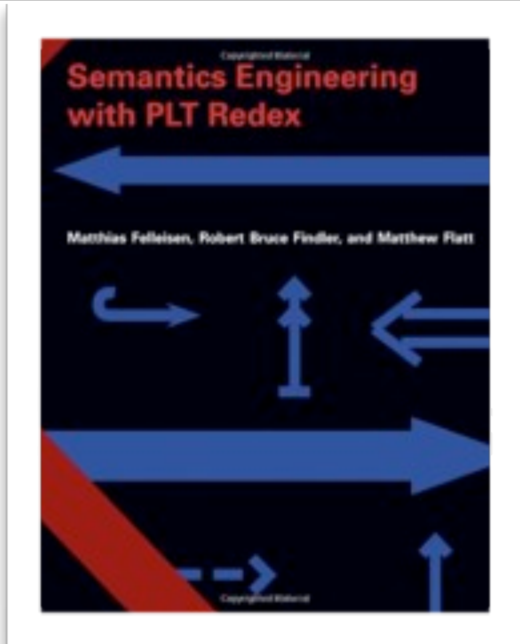
$$\langle (e_1 + e_2), \rho, \hat{\sigma}, a\rangle \quad \rightarrow \quad \langle e_1, \rho, \hat{\sigma} \sqcup [a' \mapsto \mathbf{c_1}(a, e_2)], a'\rangle$$

ICFP'10/CACM'11

Semantics



lysis



ICFP'10/CACM'11

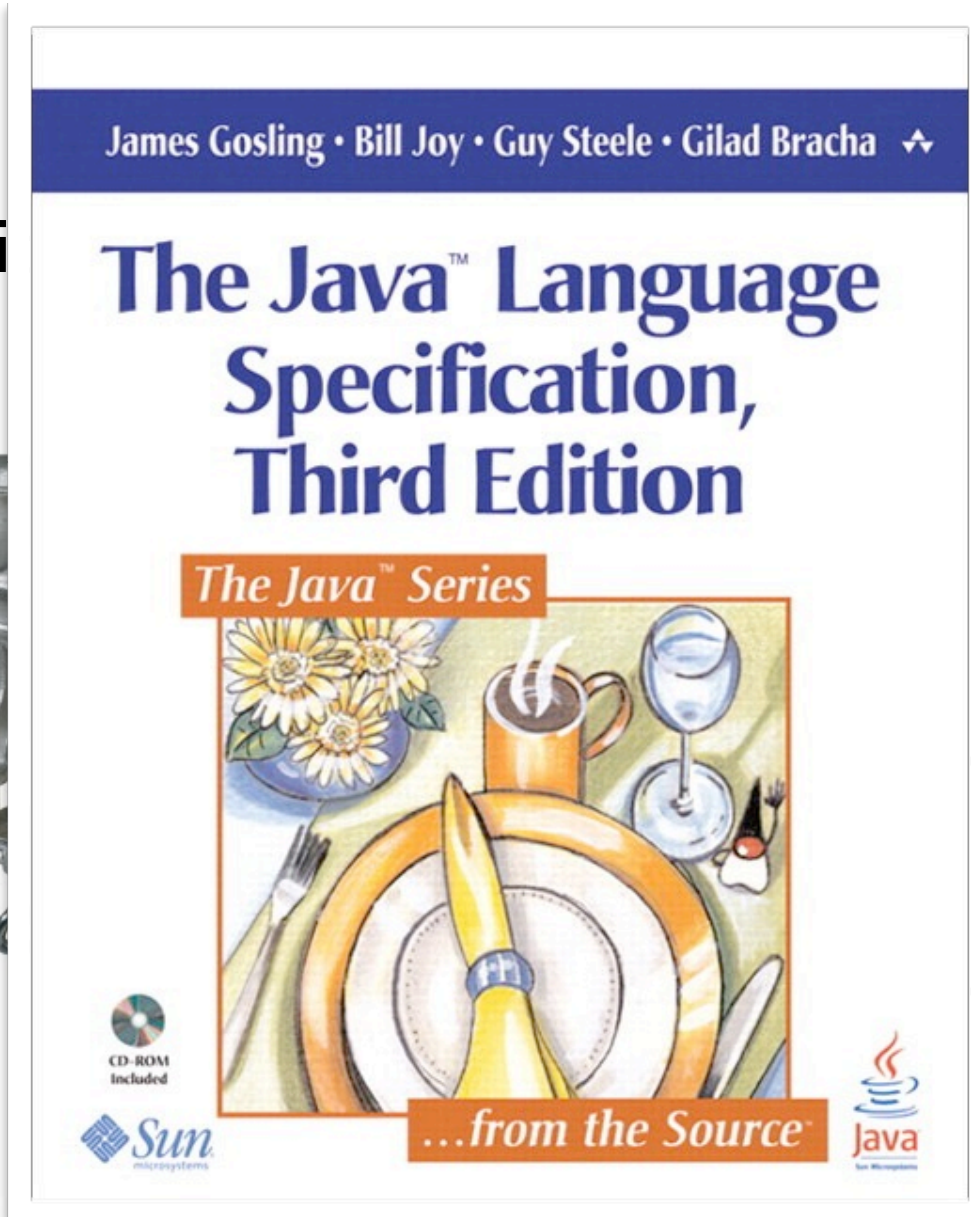
Analysis of:
First-class control
Exceptions
Mutation
Base values
...

Machine

Analysis

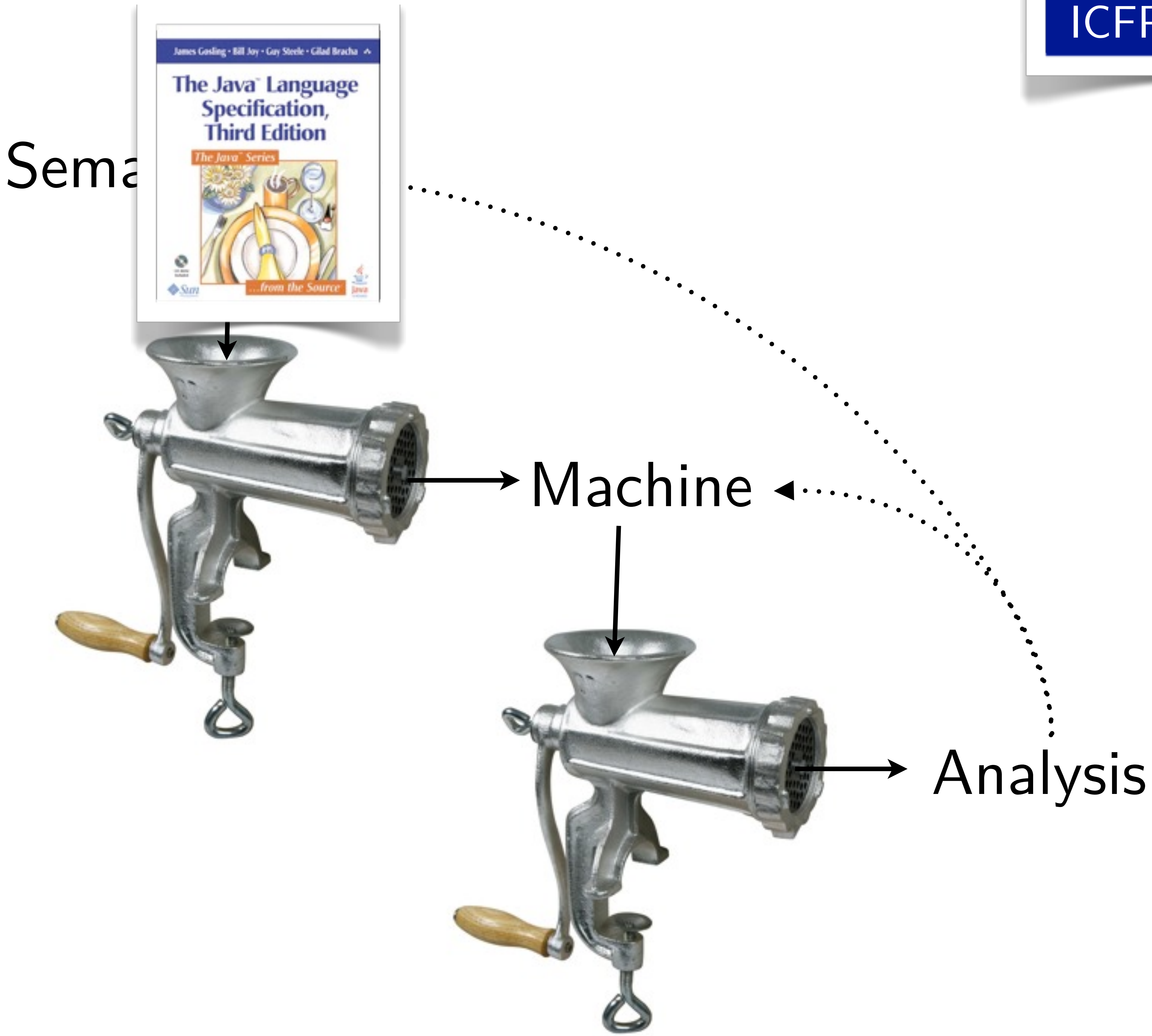
ICFP'10/CACM'11

Semanti



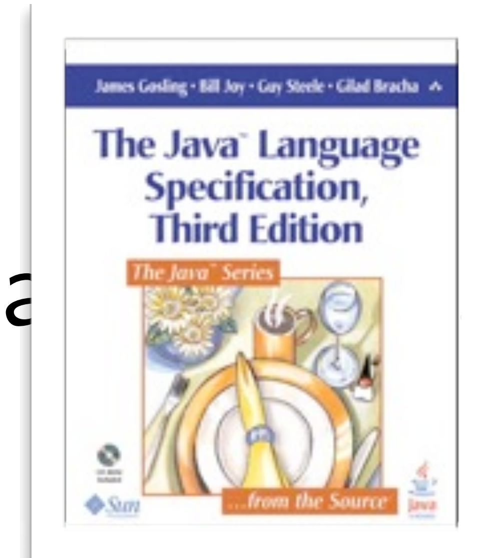
alysis

ICFP'10/CACM'11



ICFP'10/CACM'11

Sema



Static verification of security via stack inspection

Machine

Analysis

ICFP'10/CACM'11

M

Abstract Models of Memory Management*

Greg Morrisett
Carnegie Mellon
jgmorris@cs.cmu.edu

Matthias Felleisen
Rice University
matthias@cs.rice.edu

Robert Harper
Carnegie Mellon
rwh@cs.cmu.edu

Abstract

Most specifications of garbage collectors concentrate on the low-level algorithmic details of *how* to find and preserve accessible objects. Often, they focus on bit-level manipulations such as "scanning stack frames," "marking objects," "tagging data," *etc.* While these details are important in some contexts, they often obscure the more fundamental aspects of memory management: *what* objects are garbage and *why*?

We develop a series of calculi that are just low-level enough that we can express allocation and garbage collection, yet are sufficiently abstract that we may formally prove the correctness of various memory management strategies. By making the heap of a program syntactically apparent, we can specify memory actions as rewriting rules that allocate values on the heap and automatically dereference pointers to such objects when needed. This formulation permits the specification of garbage collection as a relation that removes portions of the heap without affecting the outcome of the evaluation.

Our high-level approach allows us to specify in a compact manner a wide variety of memory management techniques, including standard trace-based garbage collection (*i.e.*, the family of copying and mark/sweep collection algorithms), generational collection, and type-based, tag-free collection. Furthermore, since the definition of garbage is based on the *semantics* of the underlying language instead of the conservative approximation of inaccessibility, we are able to specify and prove the idea that type inference can be used to collect some objects that are accessible but never used.

*This work was sponsored in part by the Advanced Research Projects Agency (ARPA), CSTO, under the title "The Fox Project: Advanced Development of Systems Software," ARPA Order No. 8313, issued by ESD/AVS under Contract No. F19628–91–C–0168, Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and ARPA grant No. F33615-93-1-1330. Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of Wright Laboratory or the United States Government.

1 Memory Safety

Advanced programming languages manage memory allocation and deallocation automatically. Automatic memory managers, or garbage collectors, significantly facilitate the programming process because programmers can rely on the language implementation for the delicate tasks of finding and freeing unneeded objects. Indeed, the presence of a garbage collector ensures *memory safety* in the same way that a type system guarantees *type safety*: no program written in an advanced programming language will crash due to dangling pointer problems while allocation, access, and deallocation are transparent. However, in contrast to type systems, memory management strategies and particularly garbage collectors rarely come with a compact formulation and a formal proof of soundness. Since garbage collectors work on the machine representations of abstract values, the very idea of providing a proof of memory safety sounds unrealistic given the lack of simple models of memory operations.

The recently developed syntactic approaches to the specification of language semantics by Felleisen and Hieb [11] and Mason and Talcott [18, 19] are the first execution models that are intensional enough to permit the specification of memory management actions and yet are sufficiently abstract to permit compact proofs of important properties. Starting from the λ_v-**S** calculus of Felleisen and Hieb, we design compact specifications of a number of memory management ideas and prove several correctness theorems.

The basic idea underlying the development of our garbage collection calculi is the representation of a program's run-time memory as a global series of syntactic declarations. The program evaluation rules allocate large objects in the global declaration, which represents the heap, and automatically dereference pointers to such objects when needed. As a result, garbage collection can be specified as any relation that removes portions of the current heap without affecting the result of a program's execution.

In Section 2, we present a small functional programming language, λgc, with a rewriting semantics that makes allocation explicit. We define a semantic notion of garbage collection for λgc and prove that there is no *optimal* collection strategy that is computable. In Section 3, we specify the "free-variable" garbage collection rule which models trace-based collectors including mark/sweep and copying collectors. We prove that the free-variable rule is correct and provide two "implementations" at the syntactic level: the first corresponds to a copying collector, the second to a generational one.

In Section 4, we formalize so-called "tag-free" collection algorithms for explicitly-typed, monomorphic languages such as Pascal and Algol [7, 29, 8]. We show how to *recover*

ICFP’10/CACM’11

Abstract Models of Memory Management*

Greg Morrisett, Carnegie Mellon, jgmorris@cs.cmu.edu
Matthias Felleisen, Rice University, matthias@cs.rice.edu
Robert Harper, Carnegie Mellon, rwh@cs.cmu.edu

Abstract

Most specifications of garbage collectors concentrate on the low-level algorithmic details of *how* to find and preserve accessible objects. Often, they focus on bit-level manipulations such as “scanning stack frames,” “marking objects,” “tagging data,” *etc.* While these details are important in some contexts, they often obscure the more fundamental aspects of memory management: *what* objects are garbage and *why*?

We develop a series of calculi that are just low-level enough that we can express allocation and garbage collection, yet are sufficiently abstract that we may formally prove the correctness of various memory management strategies. By making the heap of a program syntactically apparent, we can specify memory actions as rewriting rules that allocate values on the heap and automatically dereference pointers to such objects when needed. This formulation permits the specification of garbage collection as a relation that removes portions of the heap without affecting the outcome of the evaluation.

Our high-level approach allows us to specify in a compact manner a wide variety of memory management techniques, including standard trace-based garbage collection (*i.e.*, the family of copying and mark/sweep collection algorithms), generational collection, and type-based, tag-free collection. Furthermore, since the definition of garbage is based on the *semantics* of the underlying language instead of the conservative approximation of inaccessibility, we are able to specify and prove the idea that type inference can be used to collect some objects that are accessible but never used.

*This work was sponsored in part by the Advanced Research Projects Agency (ARPA), CSTO, under the title “The Fox Project: Advanced Development of Systems Software,” ARPA Order No. 8313, issued by ESD/AVS under Contract No. F19628–91–C–0168, Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and ARPA grant No. F33615-93-1-1330. Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of Wright Laboratory or the United States Government.

1 Memory Safety

Advanced programming languages manage memory allocation and deallocation automatically. Automatic memory managers, or garbage collectors, significantly facilitate the programming process because programmers can rely on the language implementation for the delicate tasks of finding and freeing unneeded objects. Indeed, the presence of a garbage collector ensures *memory safety* in the same way that a type system guarantees *type safety*: no program written in an advanced programming language will crash due to dangling pointer problems while allocation, access, and deallocation are transparent. However, in contrast to type systems, memory management strategies and particularly garbage collectors rarely come with a compact formulation and a formal proof of soundness. Since garbage collectors work on the machine representations of abstract values, the very idea of providing a proof of memory safety sounds unrealistic given the lack of simple models of memory operations.

The recently developed syntactic approaches to the specification of language semantics by Felleisen and Hieb [11] and Mason and Talcott [18, 19] are the first execution models that are intensional enough to permit the specification of memory management actions and yet are sufficiently abstract to permit compact proofs of important properties. Starting from the λ_v-**S** calculus of Felleisen and Hieb, we design compact specifications of a number of memory management ideas and prove several correctness theorems.

The basic idea underlying the development of our garbage collection calculi is the representation of a program’s run-time memory as a global series of syntactic declarations. The program evaluation rules allocate large objects in the global declaration, which represents the heap, and automatically dereference pointers to such objects when needed. As a result, garbage collection can be specified as any relation that removes portions of the current heap without affecting the result of a program’s execution.

In Section 2, we present a small functional programming language, λgc, with a rewriting semantics that makes allocation explicit. We define a semantic notion of garbage collection for λgc and prove that there is no *optimal* collection strategy that is computable. In Section 3, we specify the “free-variable” garbage collection rule which models trace-based collectors including mark/sweep and copying collectors. We prove that the free-variable rule is correct and provide two “implementations” at the syntactic level: the first corresponds to a copying collector, the second to a generational one.

In Section 4, we formalize so-called “tag-free” collection algorithms for explicitly-typed, monomorphic languages such as Pascal and Algol [7, 29, 8]. We show how to *recover*

66

Analysis

ICFP'10/CACM'11

Abstract Models of Memory Management*

Greg Morrisett, Carnegie Mellon, jgmorris@cs.cmu.edu
Matthias Felleisen, Rice University, matthias@cs.rice.edu
Robert Harper, Carnegie Mellon, rwh@cs.cmu.edu

Abstract

Most specifications of garbage collectors concentrate on the low-level algorithmic details of *how* to find and preserve accessible objects. Often, they focus on bit-level manipulations such as "scanning stack frames," "marking objects," "tagging data," *etc.* While these details are important in some contexts, they often obscure the more fundamental aspects of memory management: *what* objects are garbage and *why*?

We develop a series of calculi that are just low-level enough that we can express allocation and garbage collection, yet are sufficiently abstract that we may formally prove the correctness of various memory management strategies. By making the heap of a program syntactically apparent, we can specify memory actions as rewriting rules that allocate values on the heap and automatically dereference pointers to such objects when needed. This formulation permits the specification of garbage collection as a relation that removes portions of the heap without affecting the outcome of the evaluation.

Our high-level approach allows us to specify in a compact manner a wide variety of memory management techniques, including standard trace-based garbage collection (*i.e.*, the family of copying and mark/sweep collection algorithms), generational collection, and type-based, tag-free collection. Furthermore, since the definition of garbage is based on the *semantics* of the underlying language instead of the conservative approximation of inaccessibility, we are able to specify and prove the idea that type inference can be used to collect some objects that are accessible but never used.

*This work was sponsored in part by the Advanced Research Projects Agency (ARPA), CSTO, under the title "The Fox Project: Advanced Development of Systems Software," ARPA Order No. 8313, issued by ESD/AVS under Contract No. F19628-91-C-0168, Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and ARPA grant No. F33615-93-1-1330. Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of Wright Laboratory or the United States Government.

Permission to make digital/hard copies of all or part of this material without fee is granted provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the Association for Computing Machinery, Inc. (ACM). To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.
FPCA '95 La Jolla, CA USA© 1995 ACM 0-89791-7/95/0006...$3.50

1 Memory Safety

Advanced programming languages manage memory allocation and deallocation automatically. Automatic memory managers, or garbage collectors, significantly facilitate the programming process because programmers can rely on the language implementation for the delicate tasks of finding and freeing unneeded objects. Indeed, the presence of a garbage collector ensures *memory safety* in the same way that a type system guarantees *type safety*: no program written in an advanced programming language will crash due to dangling pointer problems while allocation, access, and deallocation are transparent. However, in contrast to type systems, memory management strategies and particularly garbage collectors rarely come with a compact formulation and a formal proof of soundness. Since garbage collectors work on the machine representations of abstract values, the very idea of providing a proof of memory safety sounds unrealistic given the lack of simple models of memory operations.

The recently developed syntactic approaches to the specification of language semantics by Felleisen and Hieb [11] and Mason and Talcott [18, 19] are the first execution models that are intensional enough to permit the specification of memory management actions and yet are sufficiently abstract to permit compact proofs of important properties. Starting from the λ_v-$\mathbf{S}$ calculus of Felleisen and Hieb, we design compact specifications of a number of memory management ideas and prove several correctness theorems.

The basic idea underlying the development of our garbage collection calculi is the representation of a program's run-time memory as a global series of syntactic declarations. The program evaluation rules allocate large objects in the global declaration, which represents the heap, and automatically dereference pointers to such objects when needed. As a result, garbage collection can be specified as any relation that removes portions of the current heap without affecting the result of a program's execution.

In Section 2, we present a small functional programming language, λgc, with a rewriting semantics that makes allocation explicit. We define a semantic notion of garbage collection for λgc and prove that there is no *optimal* collection strategy that is computable. In Section 3, we specify the "free-variable" garbage collection rule which models trace-based collectors including mark/sweep and copying collectors. We prove that the free-variable rule is correct and provide two "implementations" at the syntactic level: the first corresponds to a copying collector, the second to a generational one.

In Section 4, we formalize so-called "tag-free" collection algorithms for explicitly-typed, monomorphic languages such as Pascal and Algol [7, 29, 8]. We show how to *recover*

66

Improved precision and efficiency via abstract GC

Analysis

ICFP'10/CACM'11

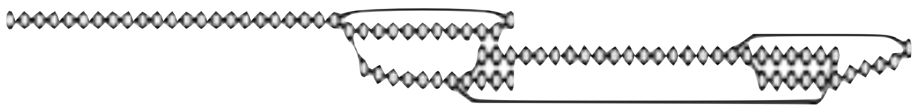
Effects of abstract GC

ICFP'10/CACM'11

Effects of abstract GC

ICFP'10/CACM'11

Effects of abstract GC



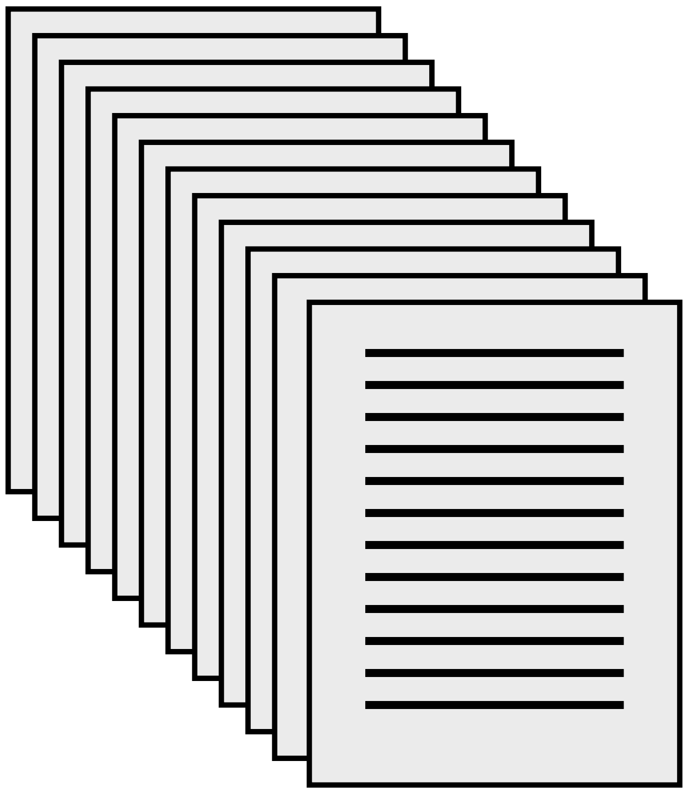
My challenge to ICFP:

Develop a program analysis for reasoning about:

- ✓ Space-consumption in a lazy language
- ✓ State and control in a language with effects
- ✓ Security in a language with stack inspection
- ✓ Blame in a language with behavioral contracts
- ✍ Safe parallelism in a language with futures

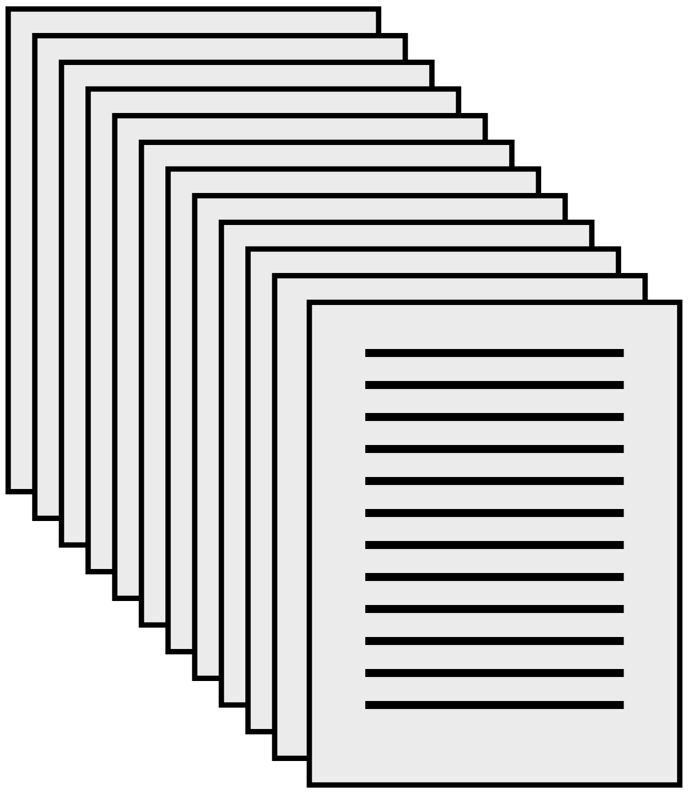
- ✓ Space-consumption in a lazy language
- ✓ State and control in a language with effects
- ✓ Security in a language with stack inspection
- ✓ Blame in a language with behavioral contracts
- ✍ Safe parallelism in a language with futures
- ✓ Garbage collection
- ✓ Java
- ✓ JavaScript
- ✍ May happen in parallel for threads

Complexity and Modularity



1M LOC

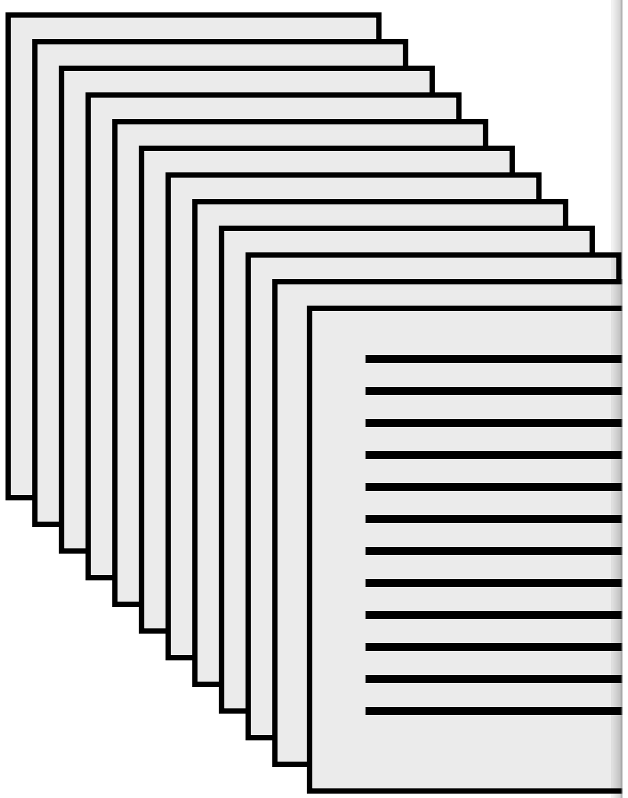
$$+ \quad O(n^3) \quad = \quad ☹$$



1M LOC

$+ \ O(n^3) \ =$ ☹

☝



IM LOC

On the Cubic Bottleneck in Subtyping and Flow Analysis

Nevin Heintze* David McAllester†

Abstract

We prove that certain data-flow and control-flow problems are 2NPDA-complete. This means that these problems are in the class 2NPDA and that they are hard for that class. The fact that they are in 2NPDA demonstrates the richness of the class. The fact that they are hard for 2NPDA can be interpreted as evidence they can not be solved in sub-cubic time — the cubic time decision procedure for an arbitrary 2NPDA problem has not been improved since its discovery in 1968.

1. Introduction

Cubic time complexity has become a common feature of algorithms for the automated analysis of computer programs. There is a general feeling that many of these algorithms are inherently cubic time — no sub-cubic procedure has been found. Such cubic time algorithms include Shivers' control flow analysis [17], the Palsberg and O'Keefe method of determining typability in the Amadio-Cardelli type system [15, 1], and various set-based analyses [5, 10, 11]. At an intuitive level the inherent cubic complexity in all these problems arises from the need to compute a dynamic transitive closure — one must compute the transitive closure of a directed graph while adding edges to the input graph as a consequence of edges derived for the output graph. Not only do these problems all seem inherently cubic, they all seem structurally similar and inherently cubic for the same reason.

In order to better understand the "cubic bottleneck" in flow analysis, Melski and Reps have investigated a simple data-flow reachability problem [13].[1] They relate this data-flow reachability problem to the problem of context-free-language reachability (CFL-reachability). An instance of the CFL-reachability problem consists of a context free grammar and a directed graph where each arc is labeled with a symbol from the terminal alphabet. The problem is to determine whether there is a path between two given nodes such that the sequence of labels on the arcs in that path is a string in the language generated by the given grammar. The CFL-reachability problem can be solved in $O(|G|n^3)$ time where $|G|$ is the size of the grammar (the number of productions in a Chomsky normal form grammar) and n is the number of nodes in the graph. Melski and Reps give a linear time reduction from data-flow reachability to CFL-reachability. This reduction produces a grammar of size n, so the reduction appears to yield an $O(n^4)$ method of solving data-flow reachability. However, Melski and Reps show that the reduction produces problems with special structure and that the overall running time of solving a data-flow problem by reduction to CFL-reachability is $O(n^3)$. More significantly, Melski and Reps give a reduction of CFL-reachability to data-flow reachability which runs in $O(|G|n)$ time. For a fixed grammar this reduction is linear time. If the data-flow reachability problem could be solved in sub-cubic time then the CFL-reachability problem over a fixed grammar could also be solved in sub-cubic time.

Here we investigate the cubic bottleneck by relating it to the class 2NPDA. 2NPDA is the class of languages (or problems) definable by a two way nondeterministic pushdown automata. In 1968 it was shown that any problem in the class 2NPDA can be solved in cubic time [2]. But no sub-cubic procedure for an arbitrary 2NPDA problem is known. Neal has shown that a certain 2NPDA problem — ground monadic rewriting reachability (GMR-reachability) — is 2NPDA complete [14].[2] In other words, this problem is both in the class 2NPDA and is 2NPDA-hard, i.e., if GMR-reachability can be solved in sub-cubic time then all 2NPDA problems can be solved in sub-cubic time. We review Neal's result here. We also show that data-flow reachability, control-flow reachability, and the complement of Amadio-

*Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974, nch@bell-labs.com.

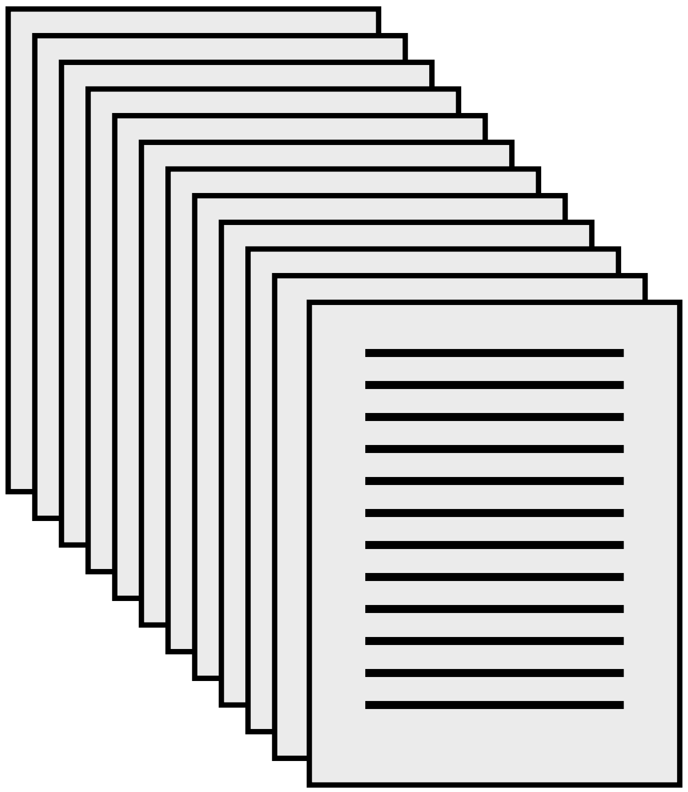
†AT&T Labs, 600 Mountain Ave, Murray Hill, NJ 07974, dmac@research.att.com.

[1] Following Heintze and Jaffar [4], Melski and Reps formulate this data-flow reachability problem as a set-constraint problem. We use the data-flow formulation here because it seems closer to applications.

[2] Neal uses a "monotone closure" formulation of GMR-problem. We find the GMR formulation more natural.

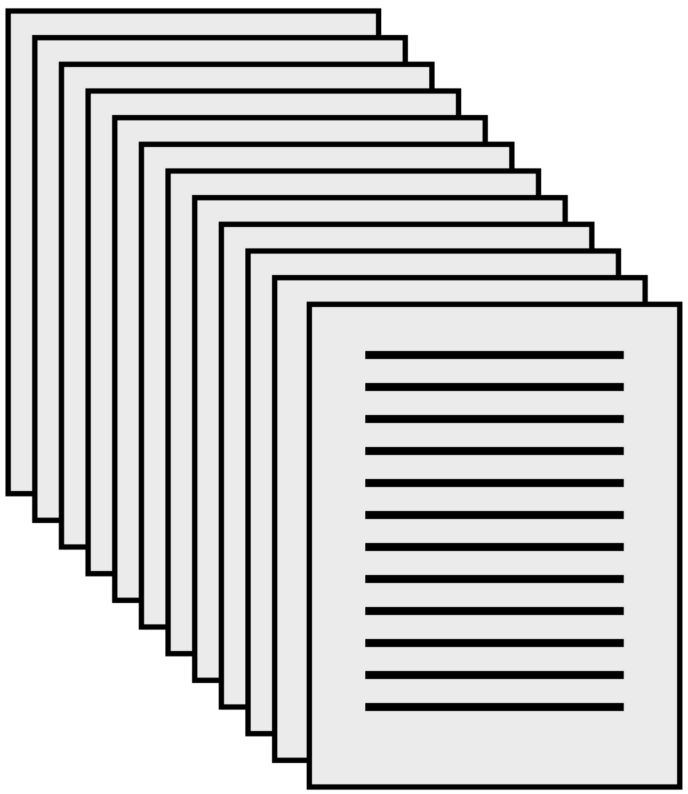
1043-6871/97 $10.00 © 1997 IEEE

HOSC'11



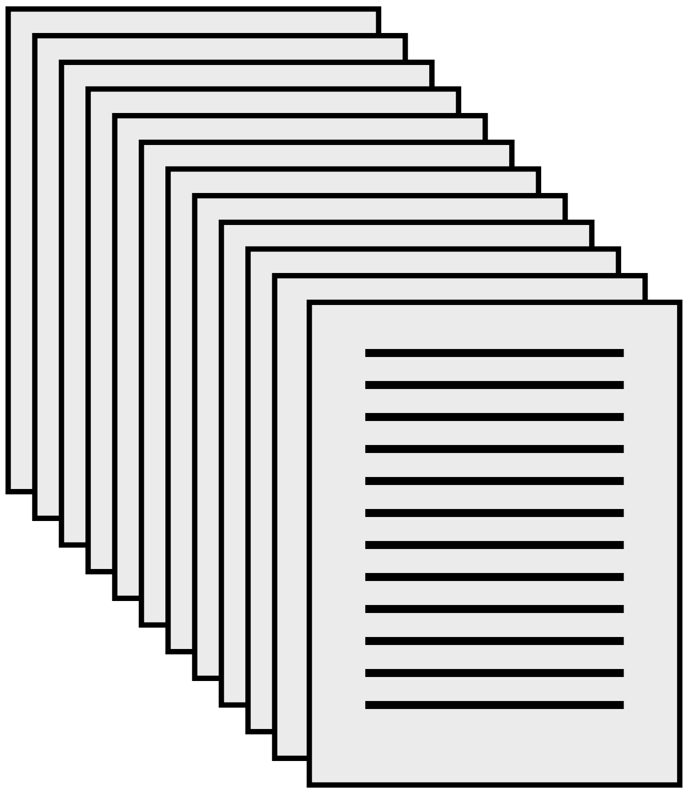
1M LOC

$$+O(n^3/\lg n) = \text{☹}$$



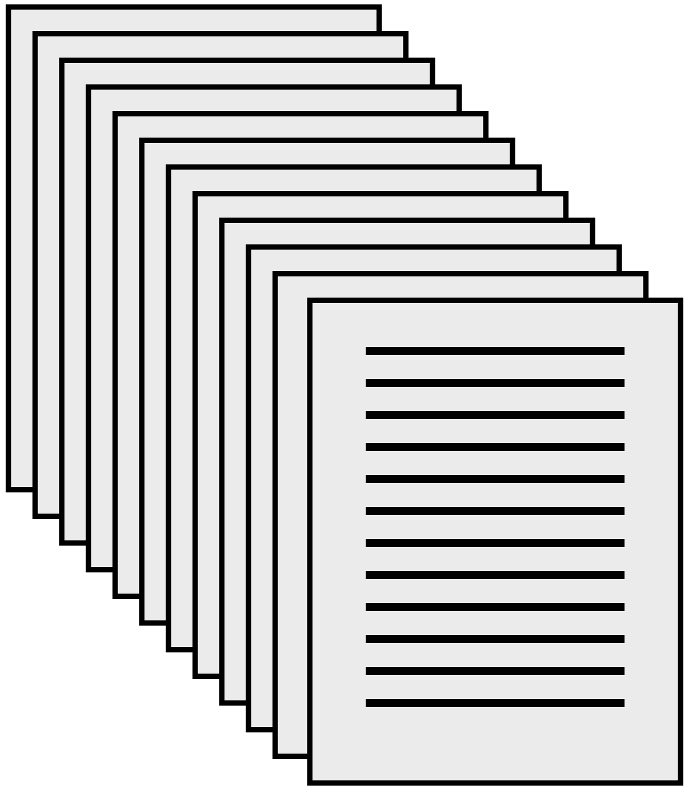
1M LOC

$$+\ O(n^{2.9}) \quad = ☹$$

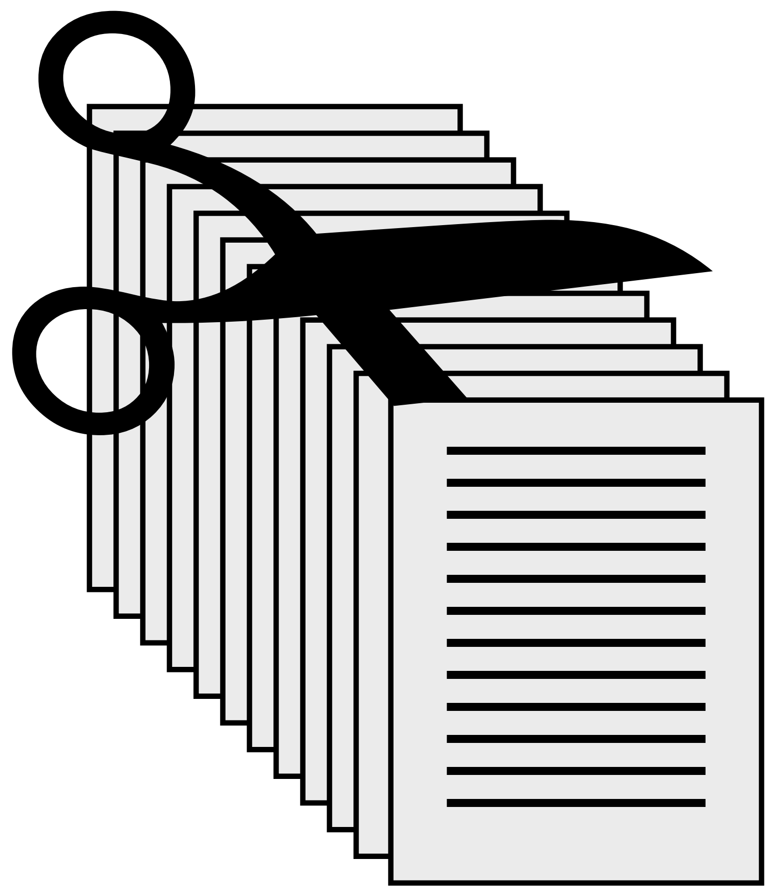


1M LOC

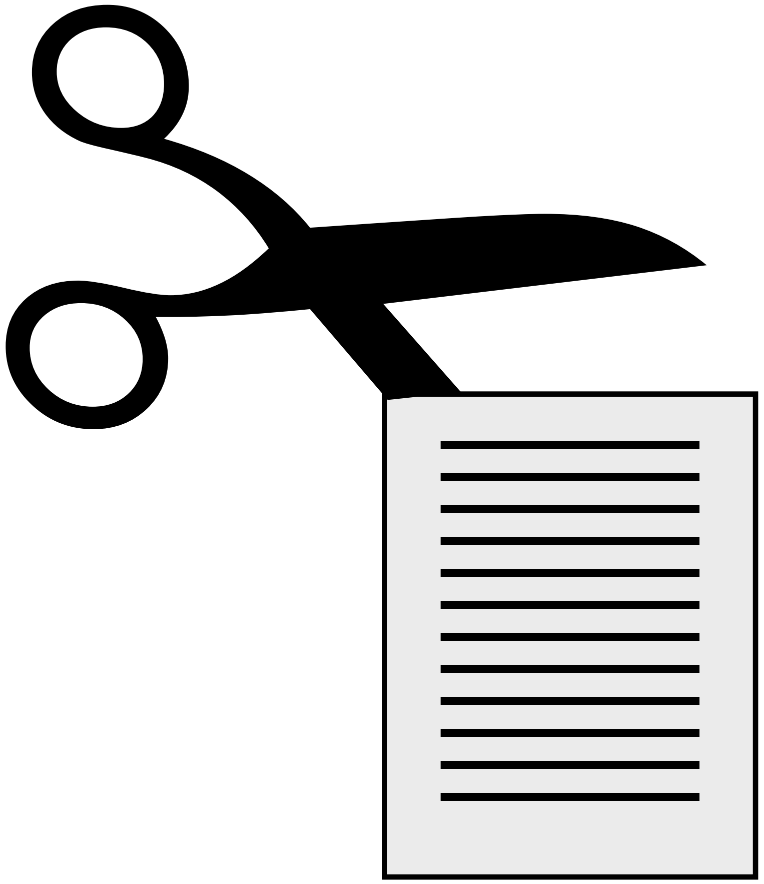
$$+ O(n^{2.9}) = $$



IM LOC



IM LOC



1K LOC

Semantics

Analysis

...of **whole** programs

Semantics

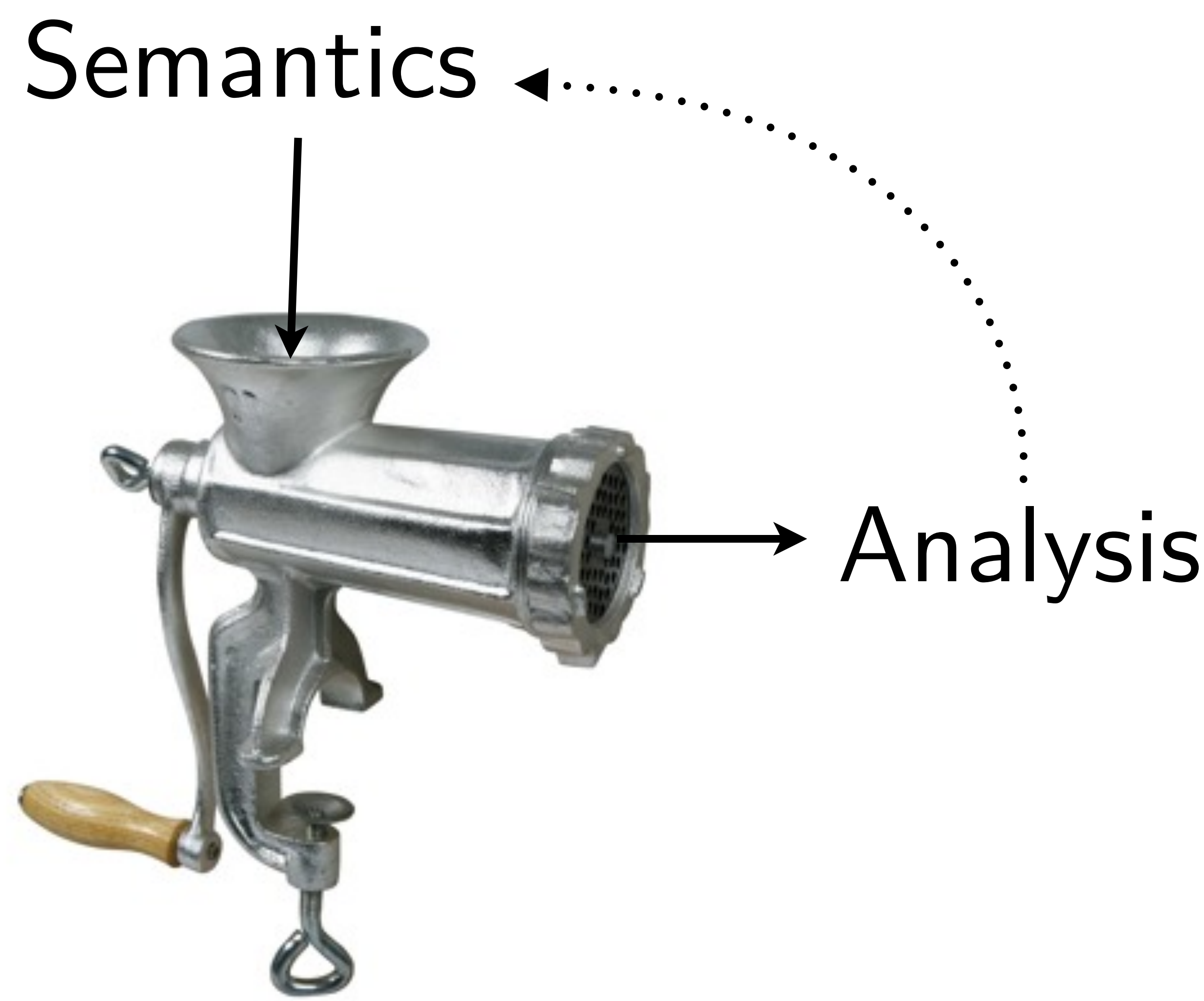
Analysis

...of **whole** programs

Semantics

...of **whole** programs

Analysis


Semantics
Analysis

...of **partial** programs

Semantics

Analysis

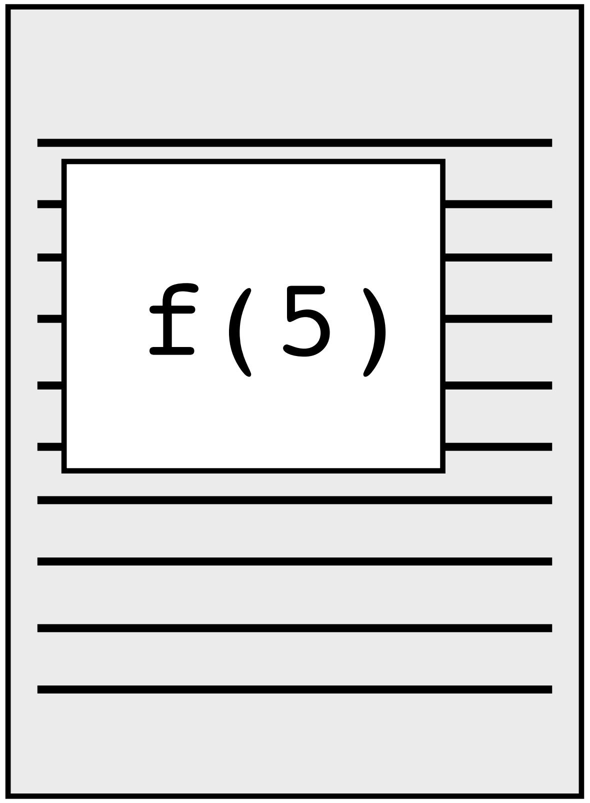
arXiv 1103.1362

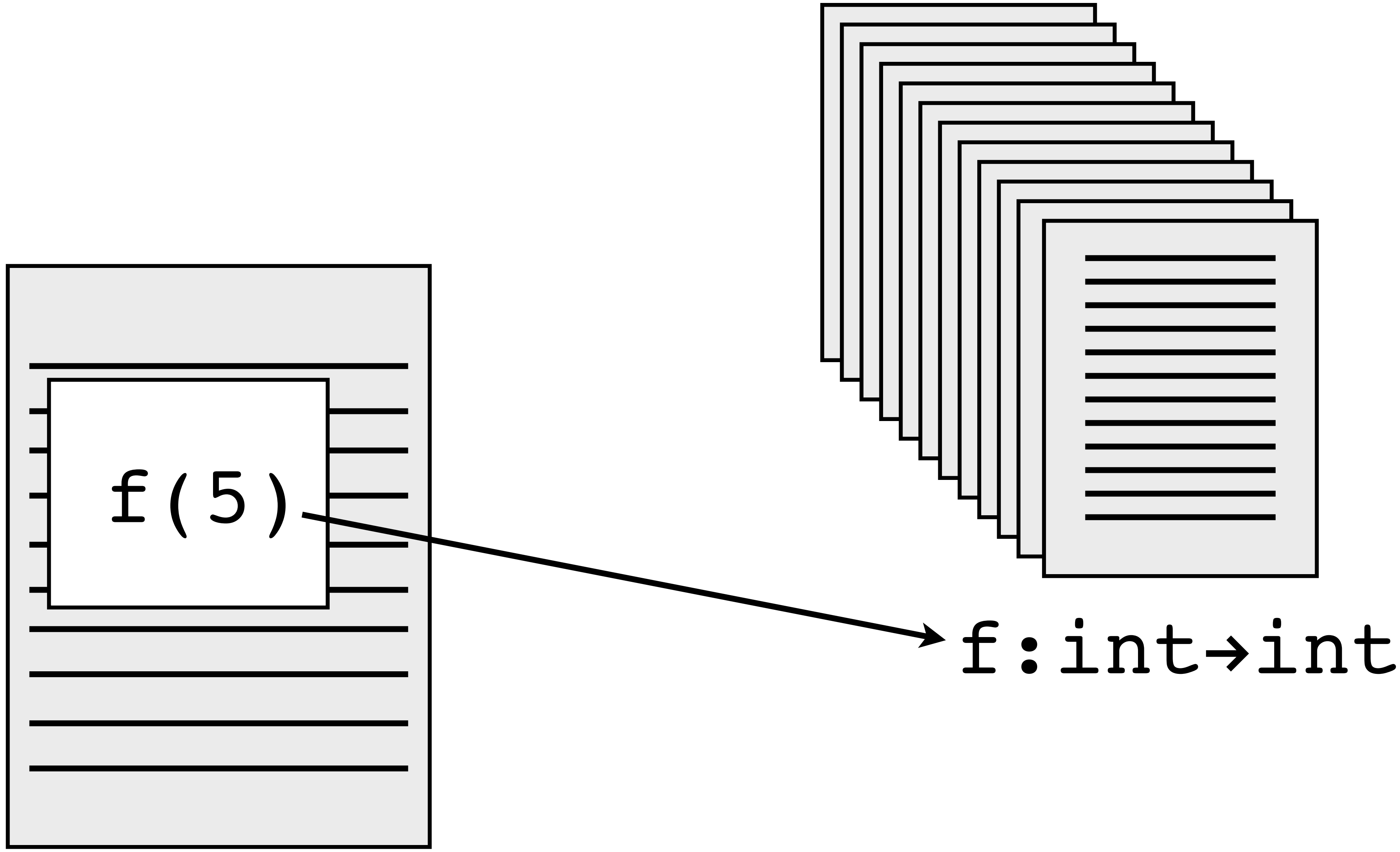
...of **partial** programs

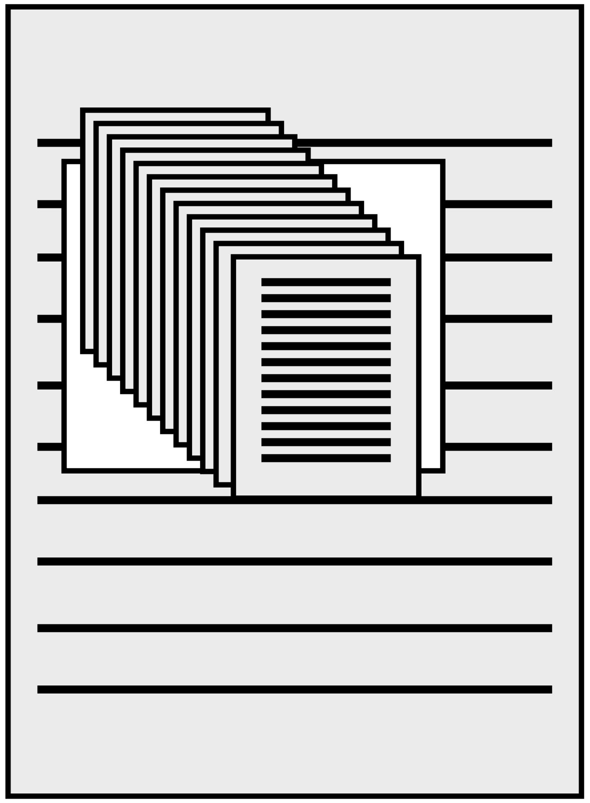
Semantics

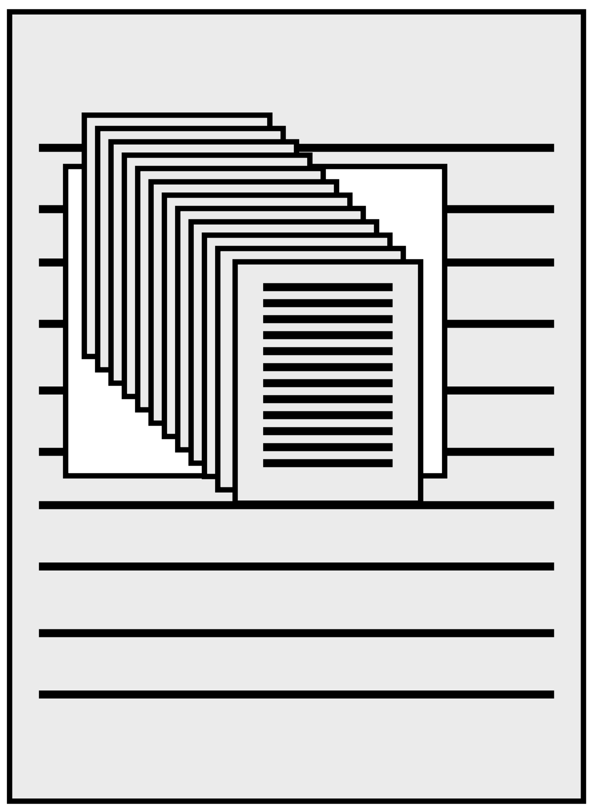
Analysis

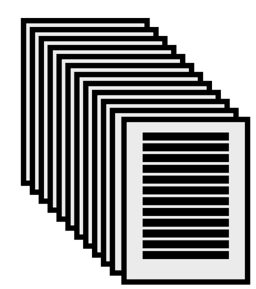
...of **partial** programs

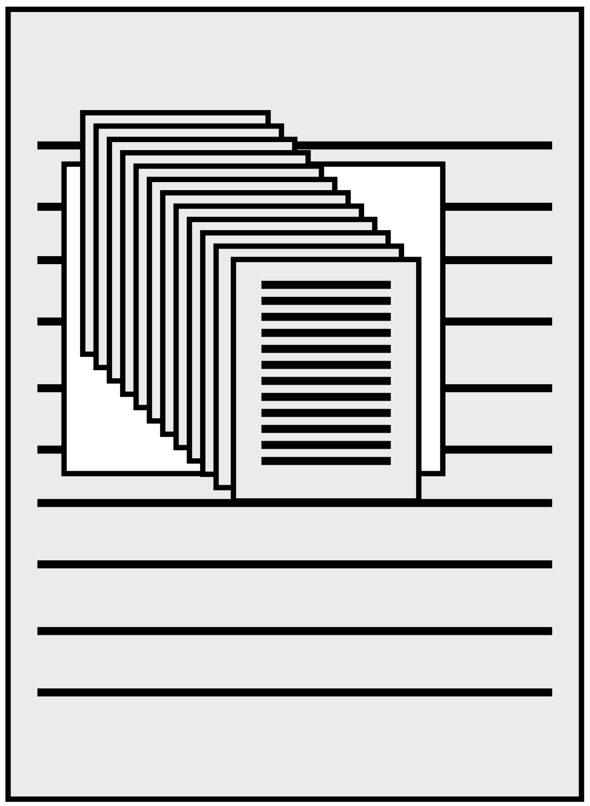

f(5)

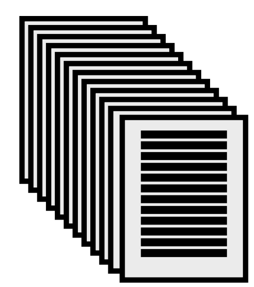

f(5)
f:int→int

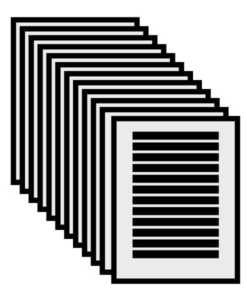


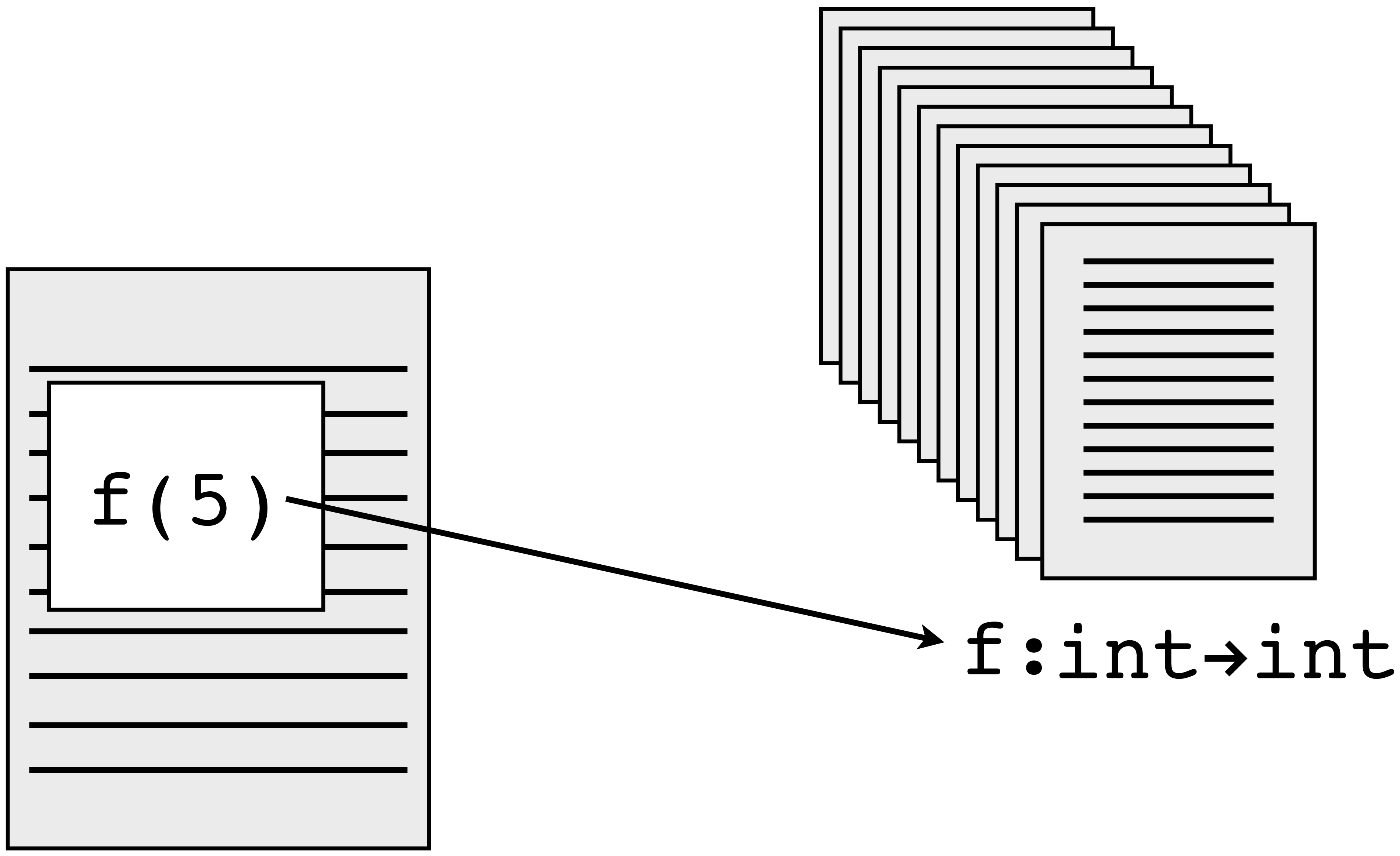


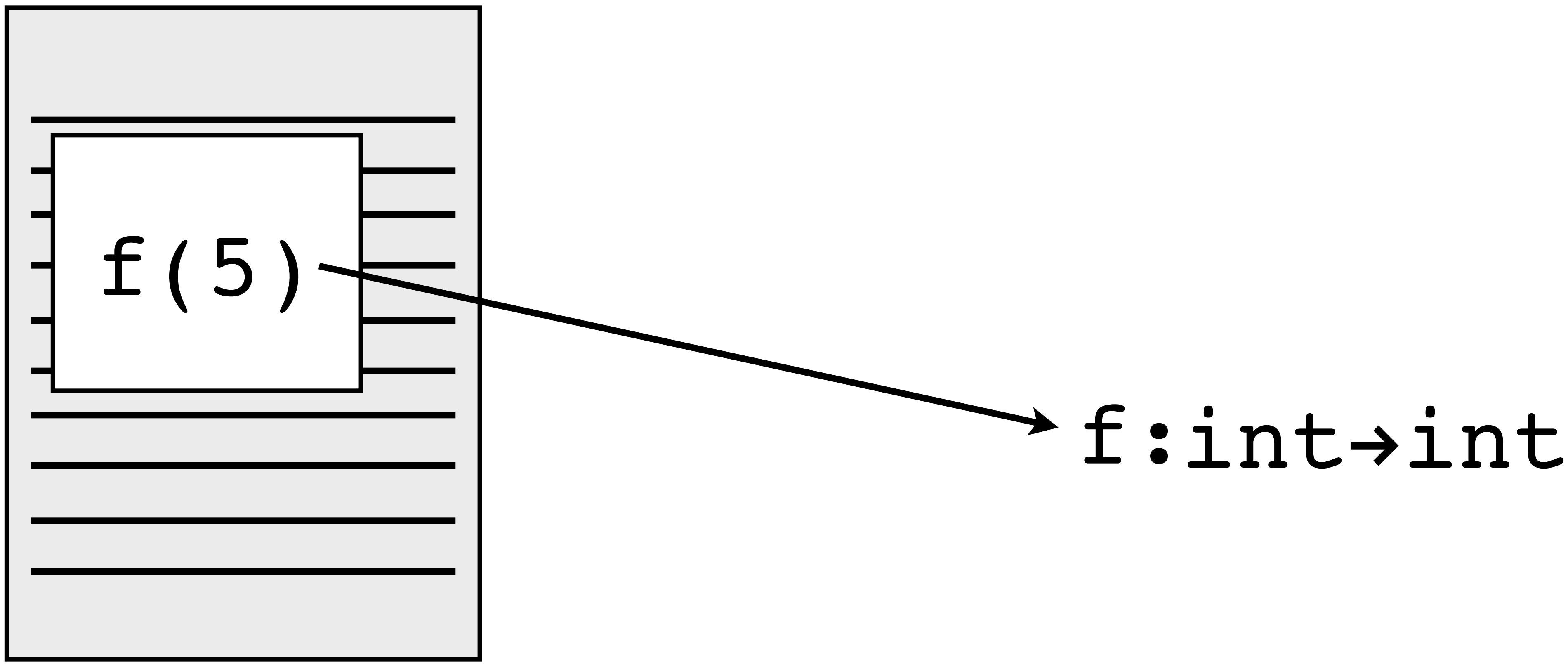
`f` $\rightarrow$ 

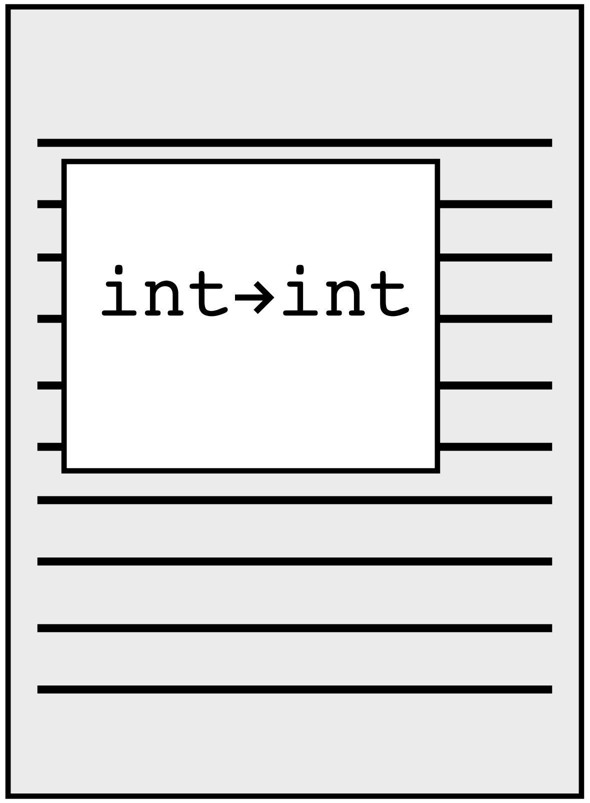


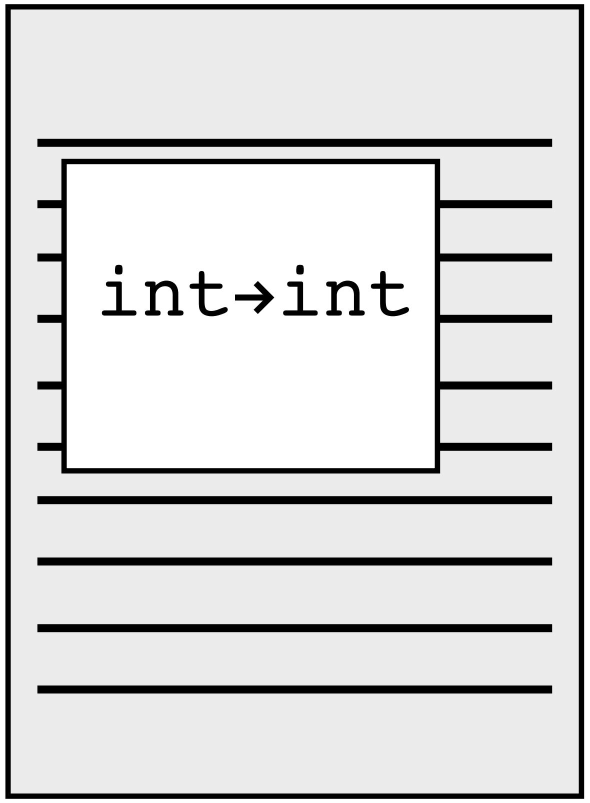
f $\rightarrow$ 

(5) $\rightarrow^{*}$ 120

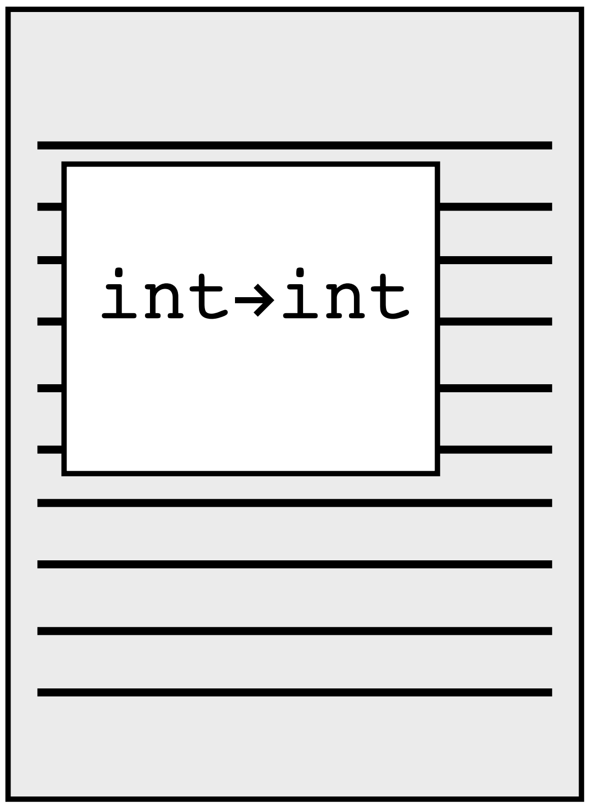

f(5)
f:int→int


f(5)
f:int→int


int→int



f $\rightarrow$ (int→int)



$$\texttt{f} \;\rightarrow\; \texttt{(int→int)}$$

$$\texttt{(int→int)(5)} \;\rightarrow\; \texttt{int}$$

Syntax:

$$e ::= v \mid x \mid (e\texttt{+}e)$$

$$v ::= n \mid \texttt{int}$$

Reduction:

$$(m\texttt{+}n) \rightarrow m + n$$

$$(\texttt{int+}v) \rightarrow \texttt{int}$$

$$(v\texttt{+int}) \rightarrow \texttt{int}$$

Eval. Contexts:

$$E ::= [\,] \mid (E\texttt{+}e) \mid (v\texttt{+}E)$$

Syntax: $e ::= \ldots \mid (\texttt{if0}\ e\ e\ e)$

Reduction: $(\texttt{if0}\ 0\ e_1\ e_2) \to e_1$

$(\texttt{if0}\ n\ e_1\ e_2) \to e_2 \text{ where } n \neq 0$

Eval. Contexts:

$E ::= [\,] \mid (E\texttt{+}e) \mid (v\texttt{+}E) \mid (\texttt{if0}\ E\ e\ e)$

Syntax: $$e ::= \ldots \mid (\texttt{if0}\ e\ e\ e)$$

Reduction:

$$(\texttt{if0}\ 0\ e_1\ e_2) \rightarrow e_1$$

$$(\texttt{if0}\ n\ e_1\ e_2) \rightarrow e_2 \text{ where } n \neq 0$$

$$(\texttt{if0}\ \texttt{int}\ e_1\ e_2) \rightarrow e_1$$

$$(\texttt{if0}\ \texttt{int}\ e_1\ e_2) \rightarrow e_2$$

Eval. Contexts:

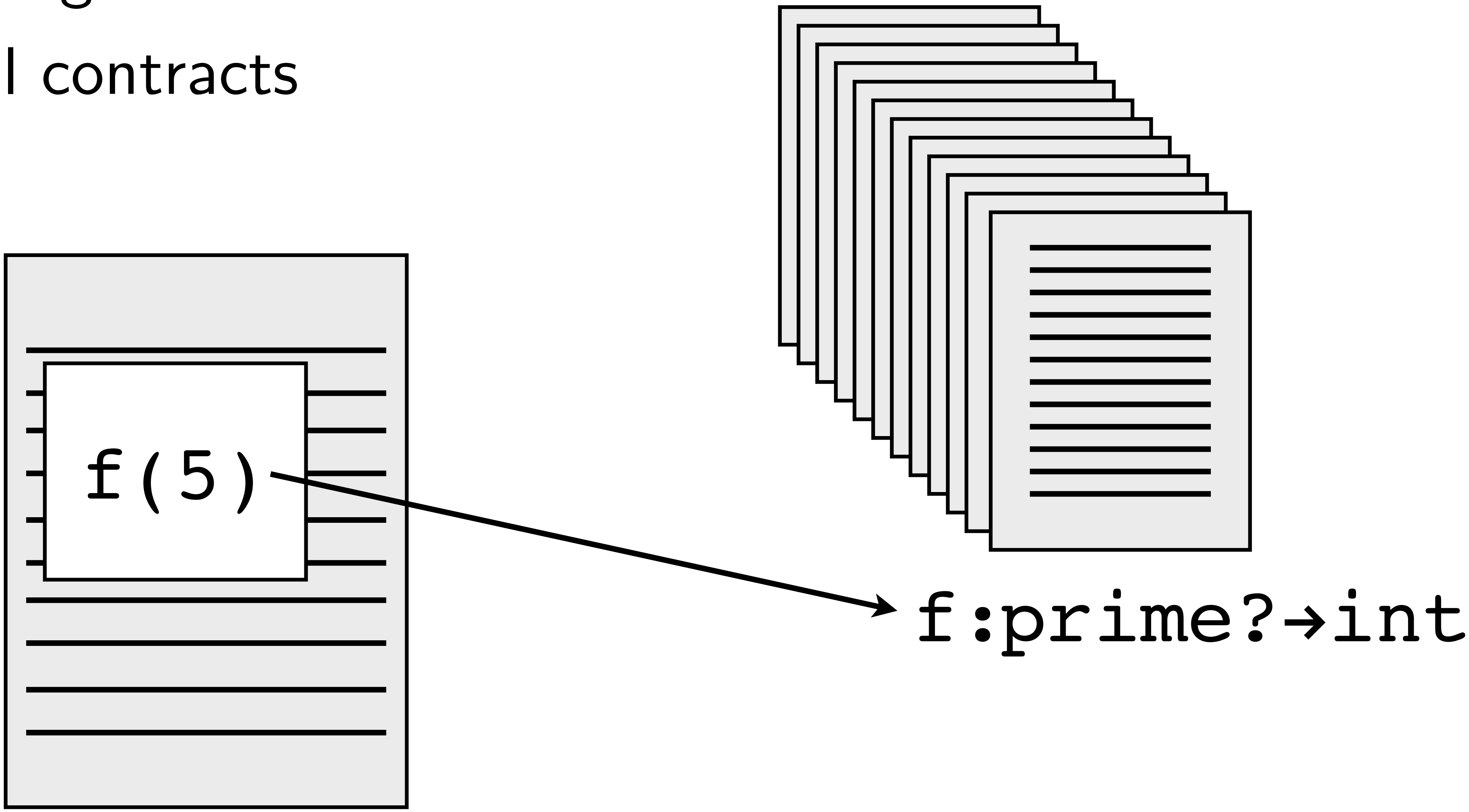
$$E ::= [\,] \mid (E\texttt{+}e) \mid (v\texttt{+}E) \mid (\texttt{if0}\ E\ e\ e)$$

Key idea:

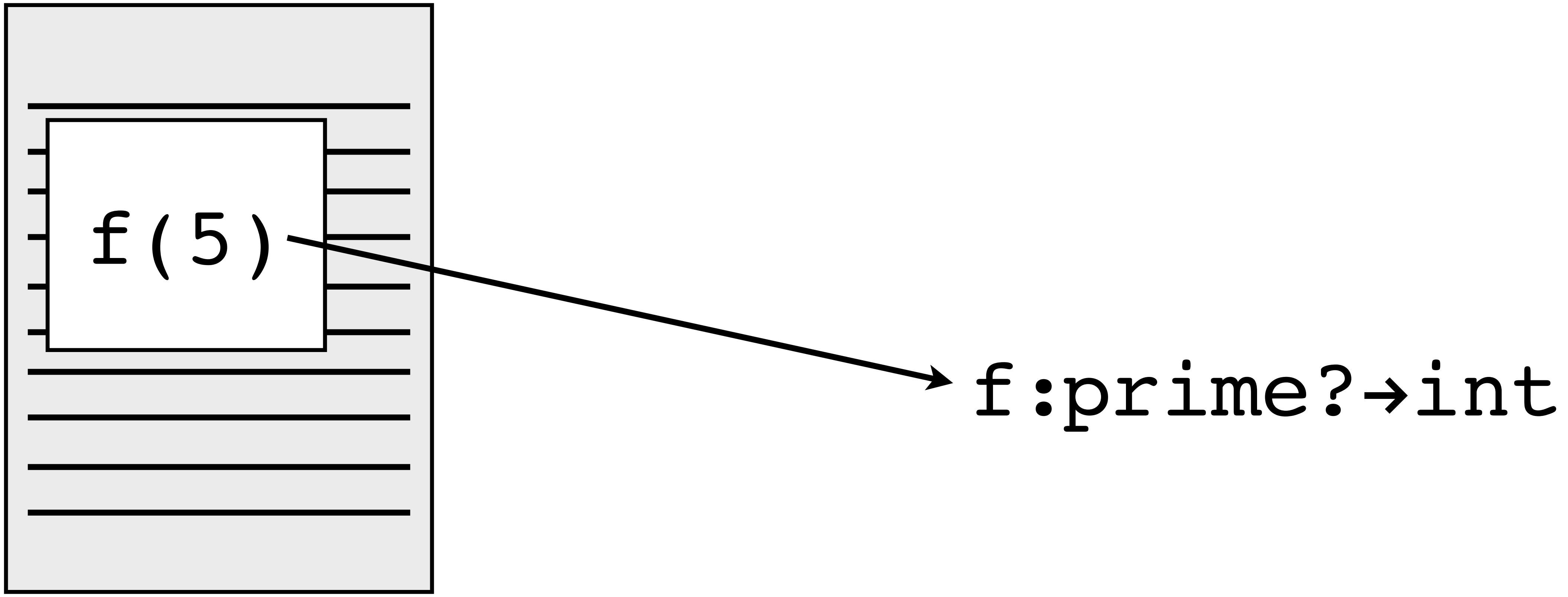
Non-deterministic state transition system with an infinite state space.

Non-deterministic state transition system with a finite state space.

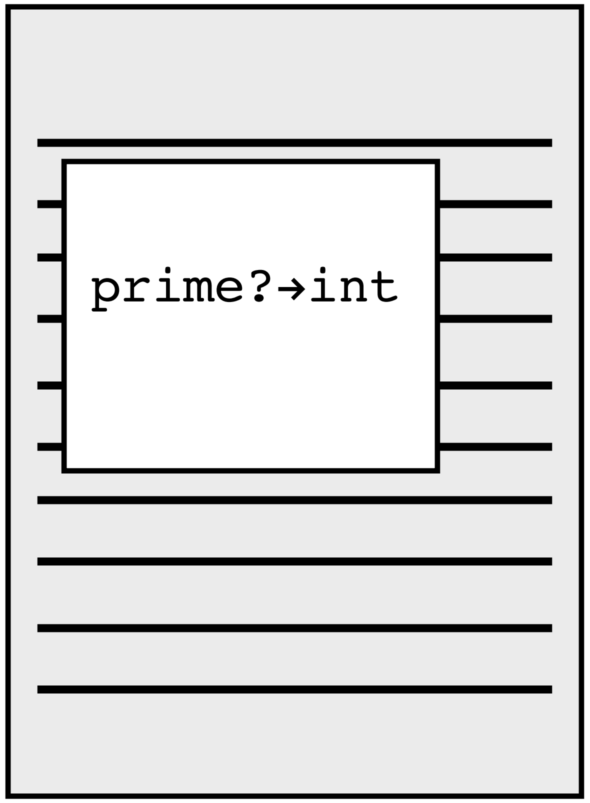
Scales to higher-order behavioral contracts



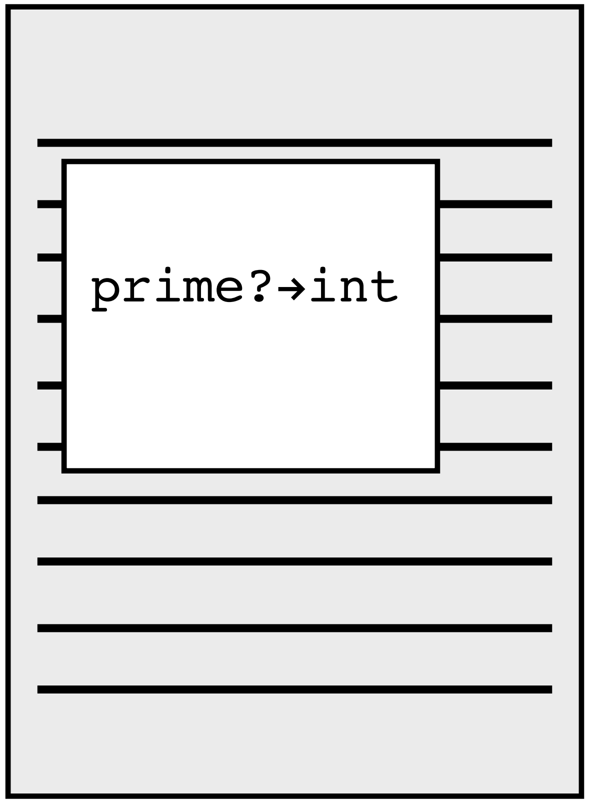
Scales to higher-order behavioral contracts



Scales to higher-order behavioral contracts

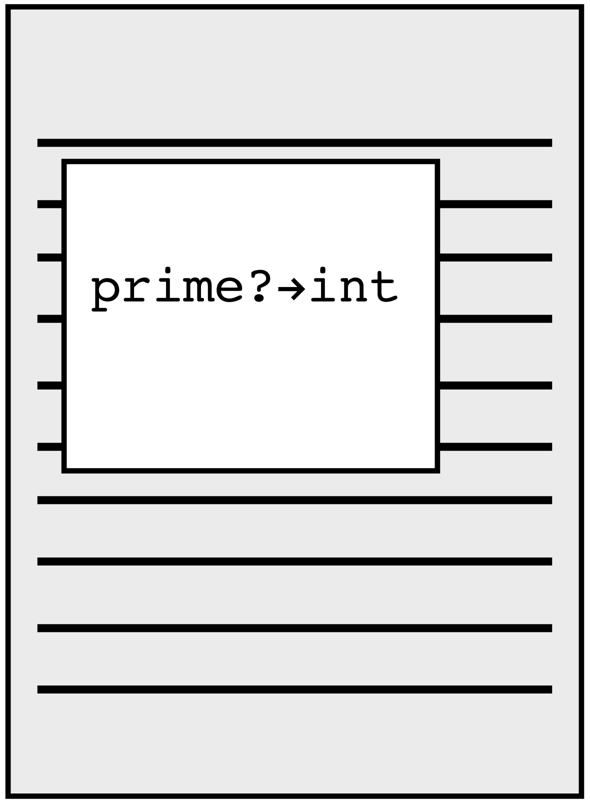


Scales to higher-order behavioral contracts



$$\texttt{f} \rightarrow \texttt{(prime?→int)}$$

Scales to higher-order behavioral contracts



$$\texttt{f} \rightarrow \texttt{(prime?→int)}$$

$$\texttt{(prime?→int)(5)} \rightarrow^* \texttt{int}$$

A Way Forward

$$\text{Scalability} \left\{ \begin{array}{l} \text{Complexity} \\ \text{Maintenance} \\ \text{Verification} \\ \text{Expressivity} \\ \text{Modularity} \end{array} \right.$$

Past

Complexity:

- ICFP'07: PTIME of context-insensitive CFA
- SAS'08: PTIME of sub- 0CFAs
- ICFP'08: EXPTIME of context-sensitive
- HOSC'11: Subcubic bottleneck broken

A Way Forward

Expressive, maintainable, verifiable, modular, performant:

- ICFP'10, CACM'11: Systematic approach analysis
- PLDI'10: Object-oriented, functional bridge
- SFP'10: Pushdown machine analysis
- 2011 (in prep): Modular reduction for modular analysis

Future

$$\text{Scalability} \left\{ \begin{array}{l} \text{Compositional} \\ \text{Componential} \\ \text{Modular} \\ \text{Parallel} \\ \text{Applied} \end{array} \right.$$

Compositional

Composing analyses for mutual benefit

Componential analyses for separate analysis

Modular

Beyond types and contracts as specifications

Parallel

May happen in parallel for H.O. + threads

Futures and imperative H.O. languages

Context-sensitive analysis on a GPU

Applied

Scripts to programs via analysis

Analysis of the Racket Machine, X10

Contract verification of .5MLOC

vision

Understand higher-order program analysis

vision

Systematic approach that scales

vision

Systematic approach that scales

vision

Tools for reasoning about large-scale software written in expressive, modern languages.

vision

Tools for reasoning about large-scale software written in expressive, modern languages.

Thank you