Mechanizing Abstract Interpretation

Thesis Defense

David Darais
University of Maryland

Software
Reliability

The Usual Story

Program

Testing

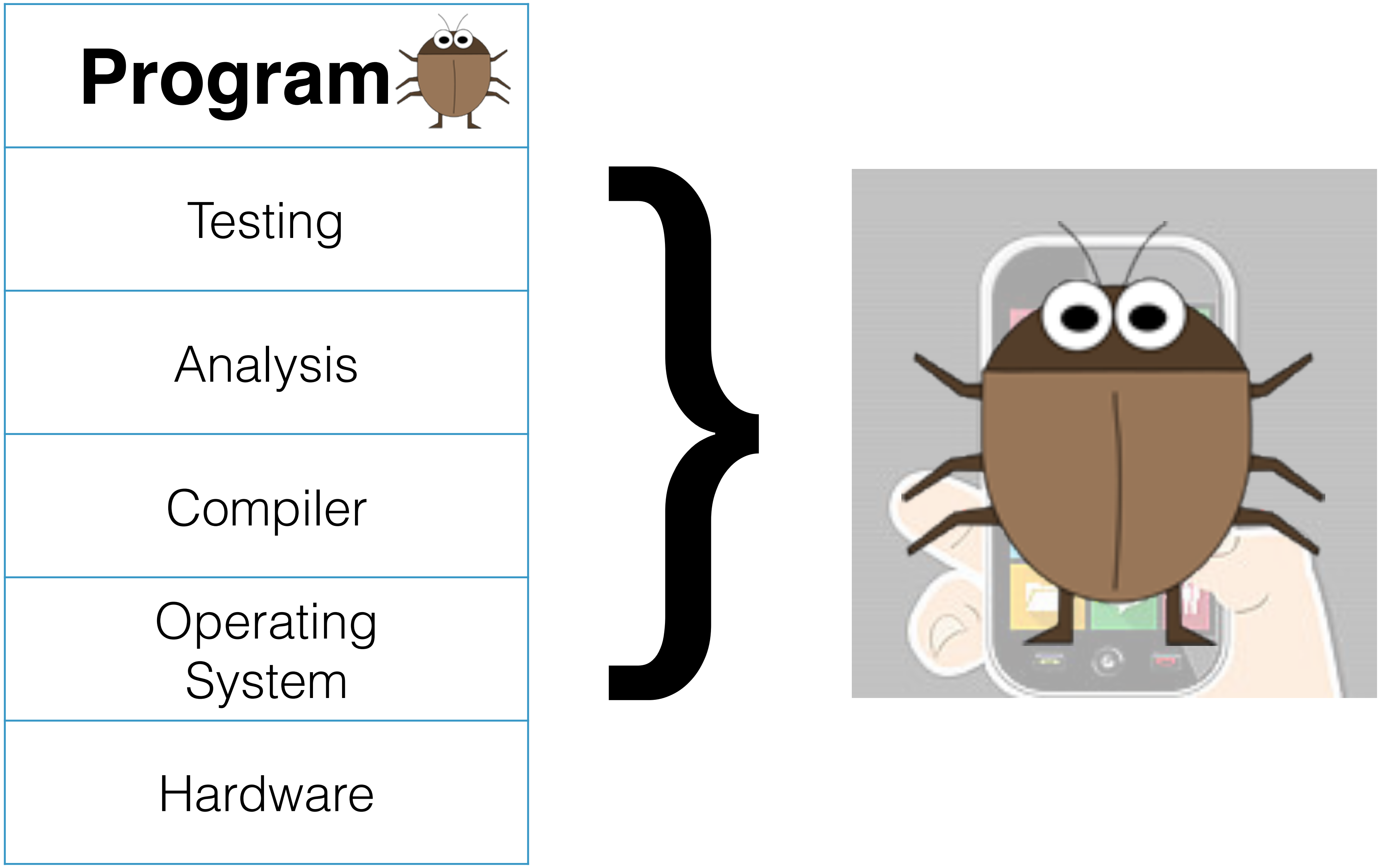
Analysis

Compiler

Operating System

Hardware

The Usual Story



Program

Testing

Analysis

Compiler

Operating System

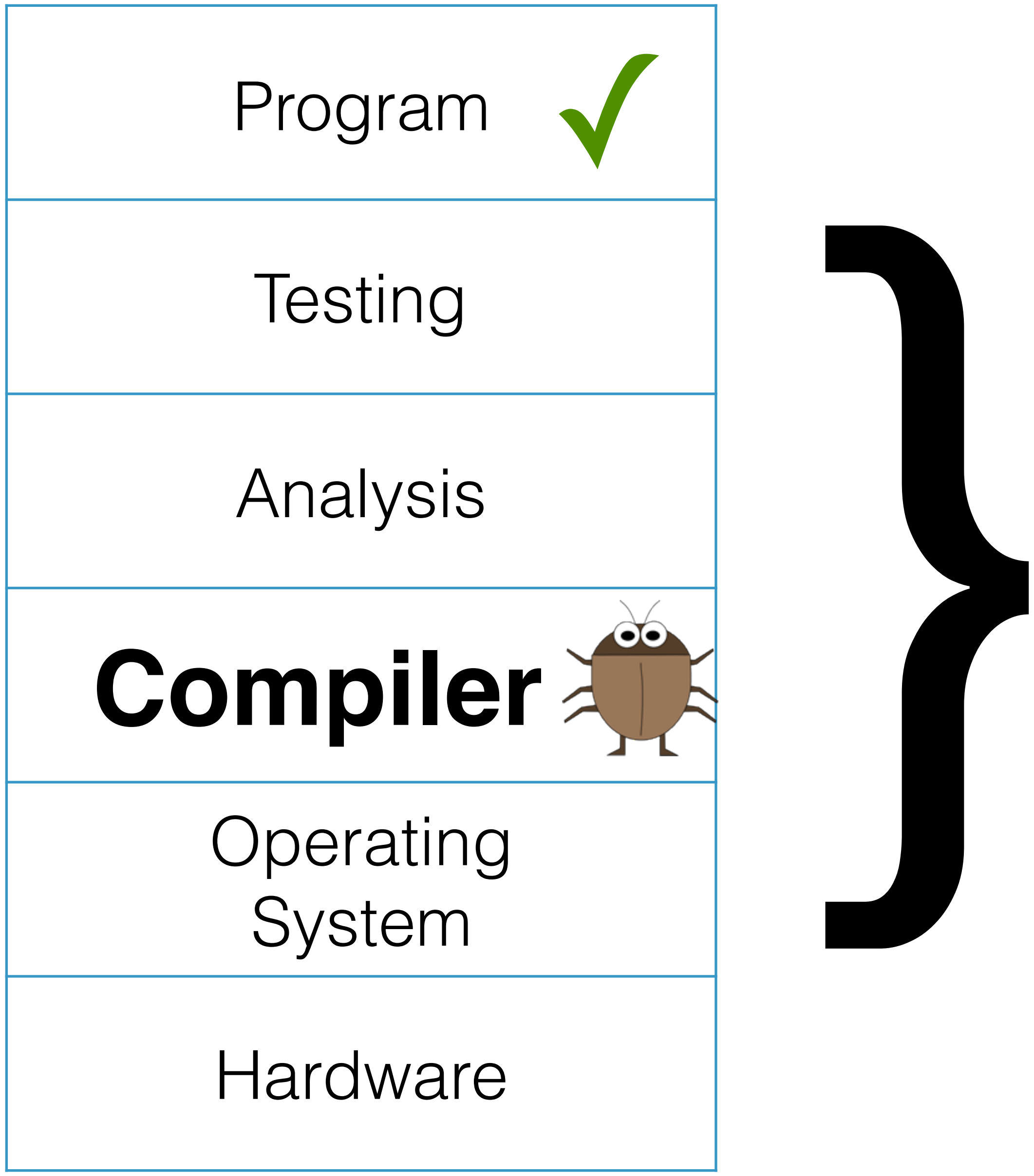
Hardware

The Reality

- Program ✓
- Testing
- Analysis
- Compiler
- Operating System
- Hardware

}

The Reality



Program ✓

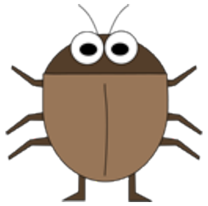
Testing

Analysis

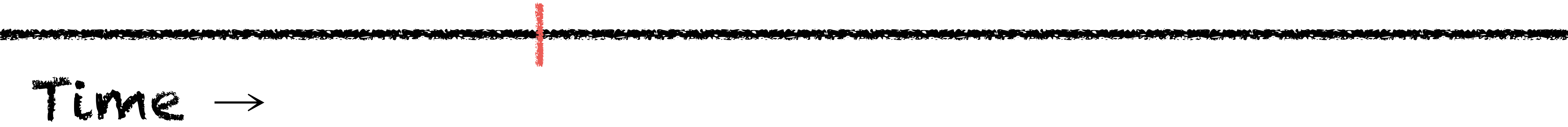
Compiler

Operating System

Hardware



Security Exploit
In Linux Kernel



(2009)

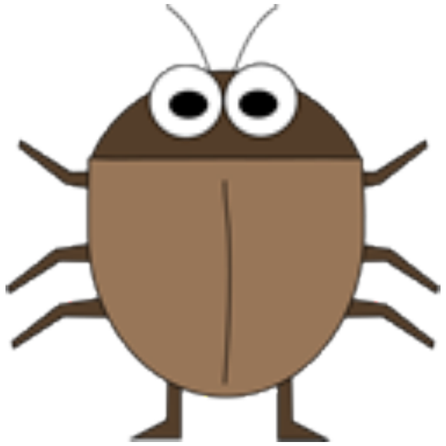
Security Exploit
In Linux Kernel

Time →

(2009)

Kernel Patch to
Fix Exploit

Security Exploit
In Linux Kernel



Security Exploit
In Linux Kernel

Time →

(2009)

Kernel Patch to
Fix Exploit

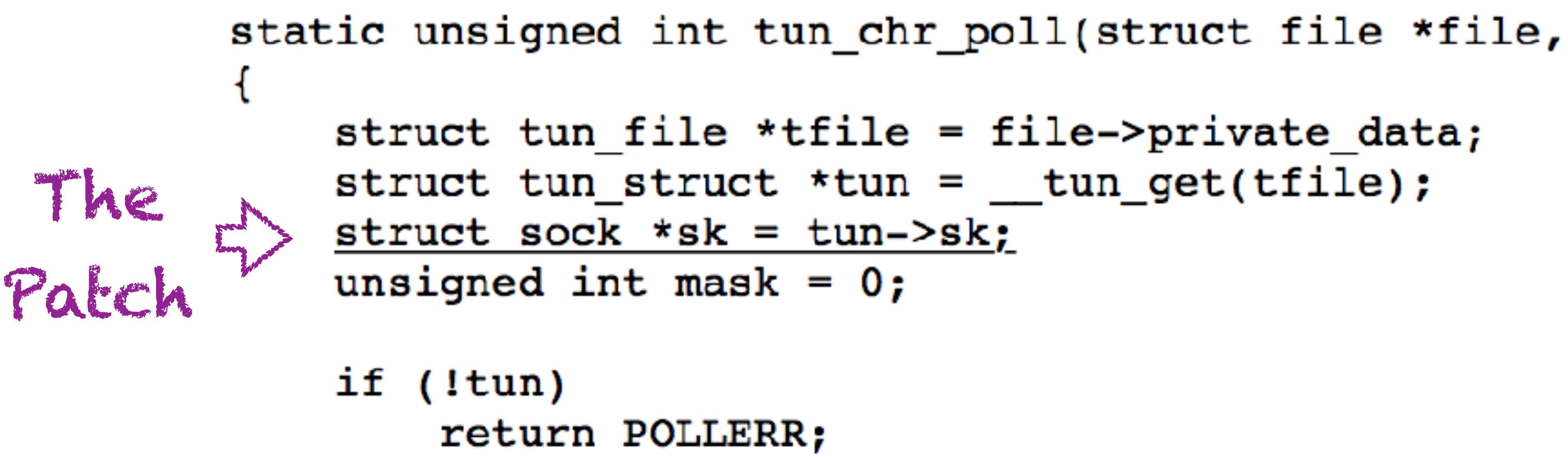
Security Exploit
In Linux Kernel

Security Exploit
In Linux Kernel

Time →
(2009)

Kernel Patch to
Fix Exploit

Story 1: Linux Kernel Exploit

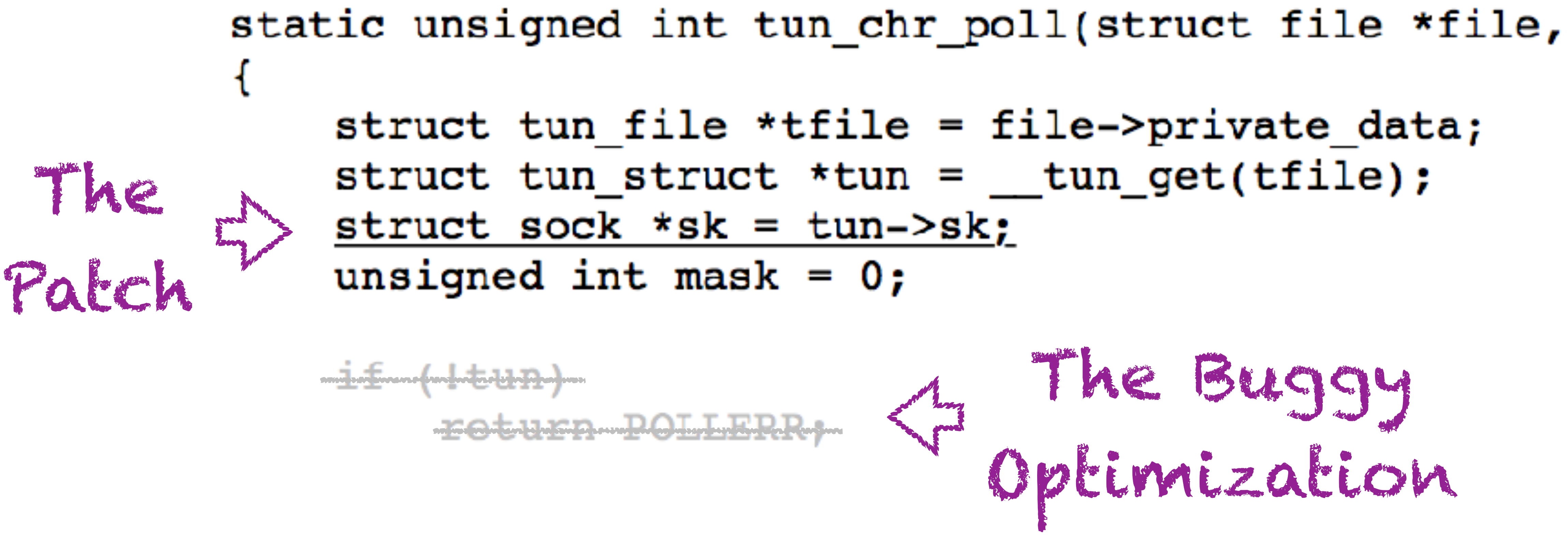


```
static unsigned int tun_chr_poll(struct file *file,
{
    struct tun_file *tfile = file->private_data;
    struct tun_struct *tun = __tun_get(tfile);
    struct sock *sk = tun->sk;
    unsigned int mask = 0;

    if (!tun)
        return POLLERR;
```

–Linux 2.6.30 kernel exploit [2009]

Story 1: Linux Kernel Exploit



```
static unsigned int tun_chr_poll(struct file *file,
{
     struct tun_file *tfile = file->private_data;
     struct tun_struct *tun = __tun_get(tfile);
     struct sock *sk = tun->sk;
     unsigned int mask = 0;
```

~~if (!tun)~~

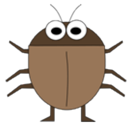
~~return POLLERR;~~

The Patch

The Buggy Optimization

–Linux 2.6.30 kernel exploit [2009]

GCC Compiler

Program


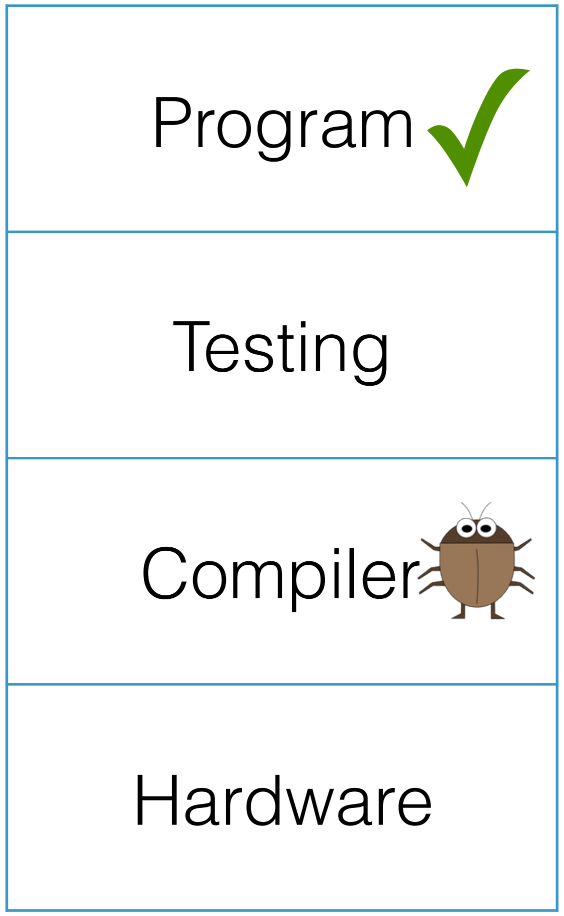
Testing

Compiler

Operating System

Hardware

GCC
Compiler



Program ✓

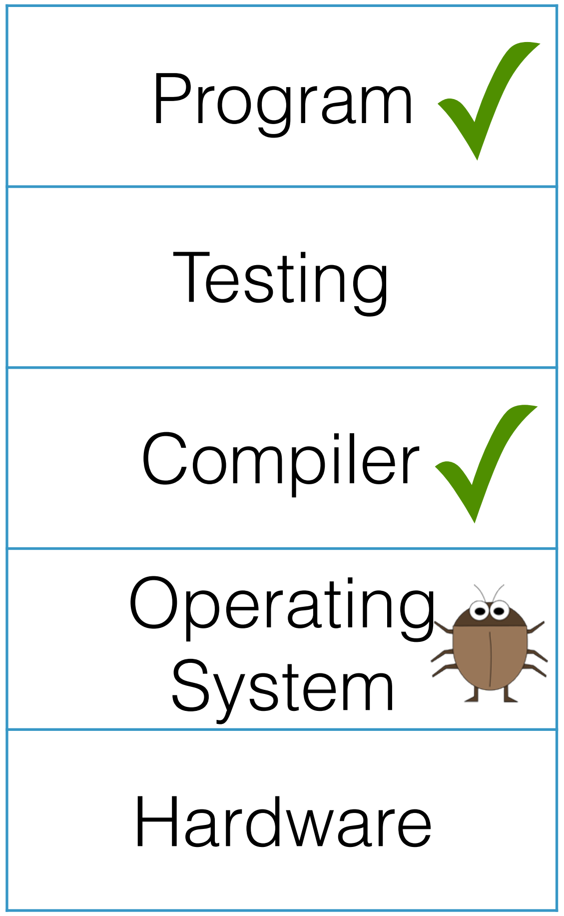
Testing

Compiler

Hardware

Linux OS
Kernel

Linux OS
Kernel



Program ✓

Testing

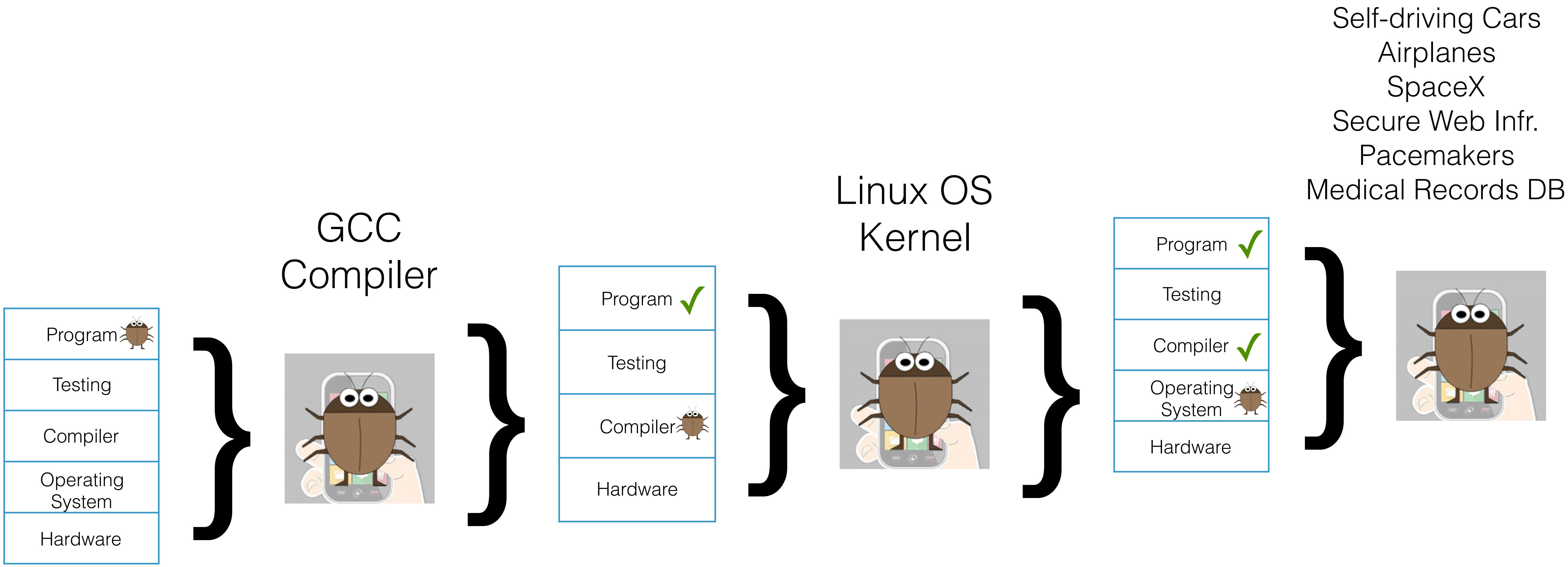
Compiler ✓

Operating System

Hardware

Self-driving Cars
Airplanes
SpaceX
Secure Web Infr.
Pacemakers
Medical Records DB

Trust in Software Runs Deep



- Program 🐞
- Testing
- Compiler
- Operating System
- Hardware

GCC Compiler

- Program ✓
- Testing
- Compiler 🐞
- Hardware

Linux OS Kernel

- Program ✓
- Testing
- Compiler ✓
- Operating System 🐞
- Hardware

Self-driving Cars
Airplanes
SpaceX
Secure Web Infr.
Pacemakers
Medical Records DB

Program ✓

Testing ✓

Analysis ✓

Compiler ✓

Operating System ✓

Hardware ✓

Critical Software Requires Trustworthy Tools

Program	✓
Testing	✓
Analysis	✓
Compiler	✓
Operating System	✓
Hardware	✓

Trustworthy Tools are Critical Software

My Research:
Tools with 0 Bugs

The Tools I Build:
Program Analyzers
(lightweight)

Difficult to Implement Correctly

The Tool I Use:

Mechanized Verification

(heavyweight)

Verify 0 Bugs in Program Analyzers

	Usable	Trustworthy
Program Analyzers	✓	✗

	Usable	Trustworthy
Program Analyzers	✓	✗
Mechanized Verification	✗	✓

	Usable	Trustworthy
Program Analyzers	✓	✗
Mechanized Verification	✗	✓
Mechanically Verified Program Analyzers	✓	✓

My Research

Problem

Building one
verified analyzer is
extremely difficult.

(decades for first compiler)

Assumption

Calculational and *compositional* methods can make analyzers easier to construct.

Research Question

How can we construct
mechanically verified
program analyzers
using calculational and
compositional methods?

Thesis

Constructing mechanically verified program analyzers via calculation and composition is *feasible* using constructive Galois connections and modular abstract interpreters.

Contribution 1

State of the art in program analysis and mechanized verification:

Abstract interpretation: 0 bugs in analyzer design+specification

Mechanized verification: 0 bugs in analyzer implementation

~20 year old problem: how to combine these two techniques

Contribution 1

State of the art in program analysis and mechanized verification:

Abstract interpretation: 0 bugs in analyzer design+specification

Mechanized verification: 0 bugs in analyzer implementation

~20 year old problem: how to combine these two techniques

Result: achieved mechanically verified calculational AI

Idea: new AI framework which supports mechanization

[**Darais** and Van Horn, ICFP '16]

Contribution 2

State of the art in *reusable* program analyzers:

Some features easy to reuse: context and object sens.

Some features had to reuse: path and flow sens.

Challenge: achieve reuse in both implementation and proof

Contribution 2

State of the art in *reusable* program analyzers:

Some features easy to reuse: context and object sens.

Some features had to reuse: path and flow sens.

Challenge: achieve reuse in both implementation and proof

Result: compositional PA components, implementation + proofs

Idea: combine monad transformers and Galois connections

[**Darais,** Might and Van Horn, OOPSLA '15]

Contribution 3

State of the art in *reusable* program analysis:

Control flow abstraction: often too imprecise

Pushdown precision: precise abstraction for control

No technique which supports compositional interpreters

Contribution 3

State of the art in *reusable* program analysis:

Control flow abstraction: often too imprecise

Pushdown precision: precise abstraction for control

No technique which supports compositional interpreters

Result: pushdown precision for definitional interpreters

Idea: inherit precision from defining metalanguage

[**Darais**, Labich, Nguyễn and Van Horn, ICFP ‘17]

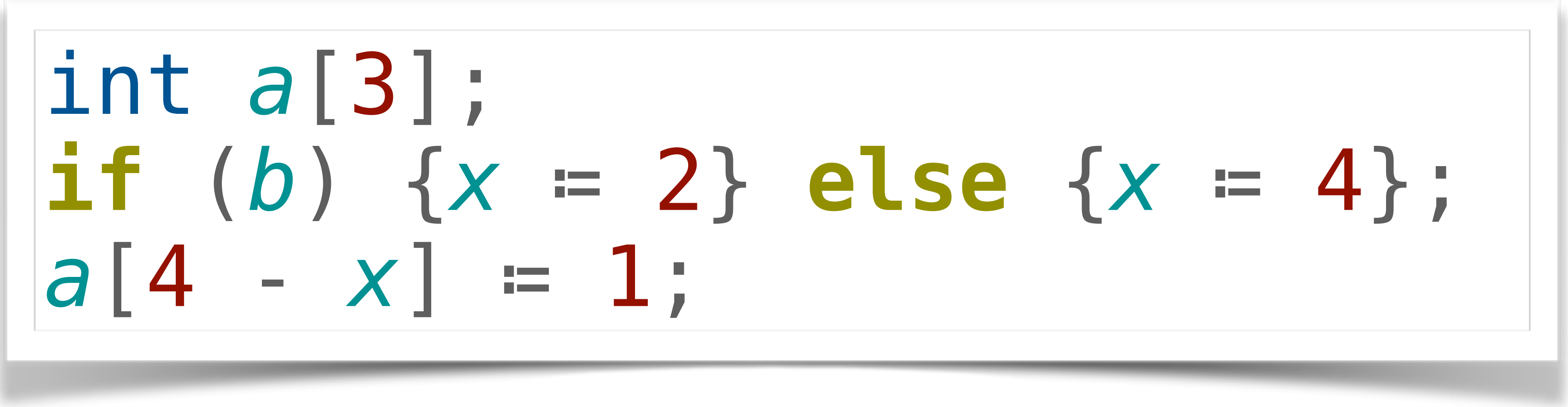
Constructive
Galois
Connections

Galois
Transformers

Abstracting
Definitional
Interpreters

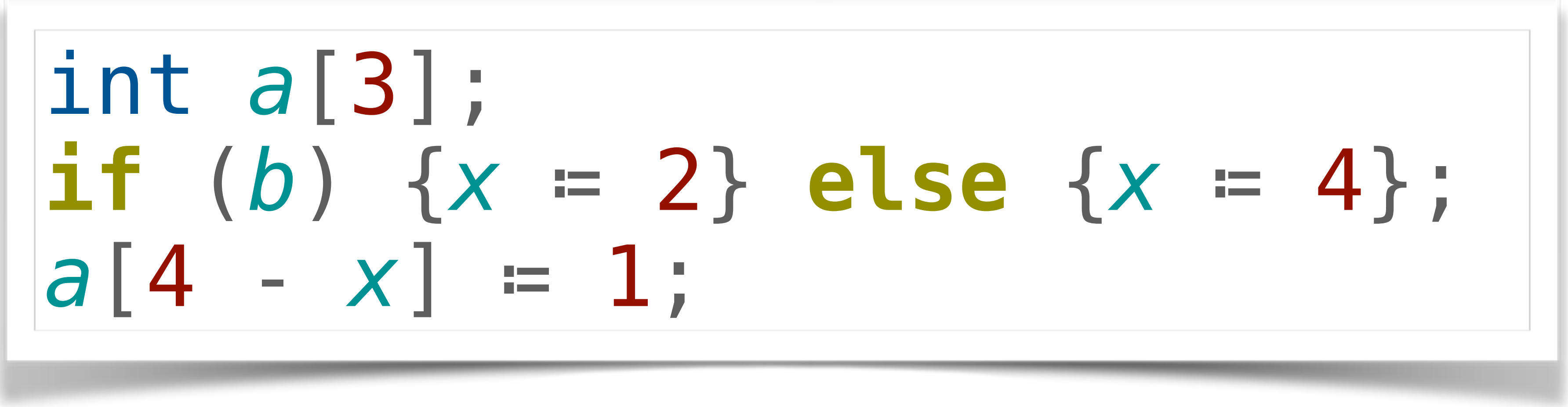
Constructive Galois Connections

Classical Galois Connections



```
int a[3];
if (b) {x ≔ 2} else {x ≔ 4};
a[4 - x] ≔ 1;
```

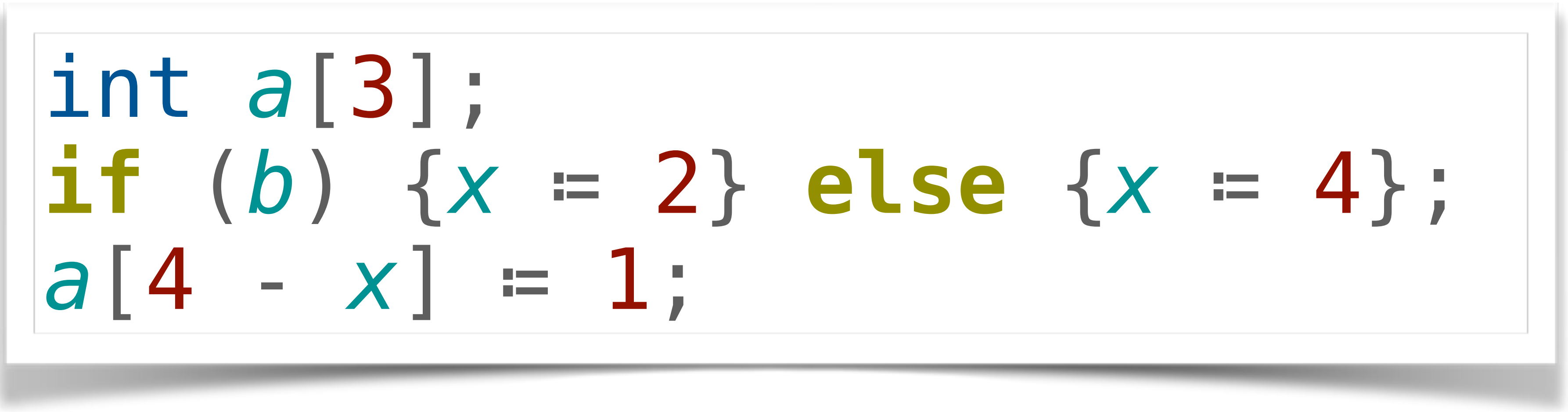
Classical Galois Connections



```
int a[3];
if (b) {x ≔ 2} else {x ≔ 4};
a[4 - x] ≔ 1;
```

`x ∈ {2,4}`

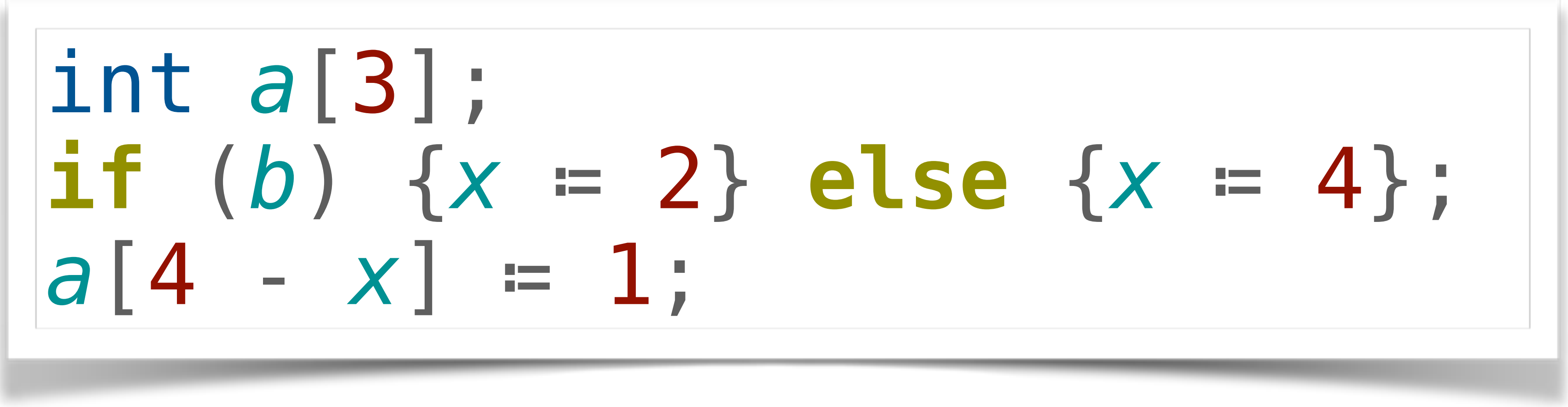
Classical Galois Connections



```
int a[3];
if (b) {x ≔ 2} else {x ≔ 4};
a[4 - x] ≔ 1;
```

$$x \in \{2,4\} \dashrightarrow x \in [2,4]$$

Classical Galois Connections

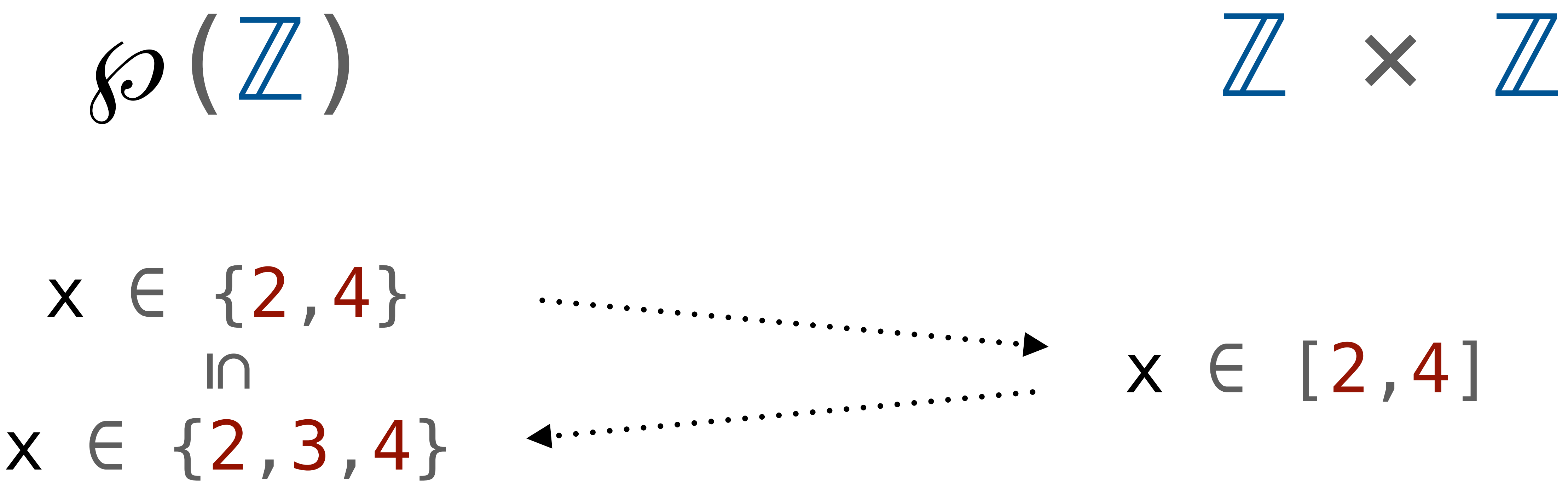


```
int a[3];
if (b) {x ≔ 2} else {x ≔ 4};
a[4 - x] ≔ 1;
```

$$\begin{tikzcd}
x \in \{2,4\} \arrow[rd, dotted] & \\
\rotatebox{-90}{$\subseteq$} & x \in [2,4] \arrow[ld, dotted] \\
x \in \{2,3,4\} &
\end{tikzcd}$$

Classical Galois Connections

```
int a[3];
if (b) {x ≔ 2} else {x ≔ 4};
a[4 - x] ≔ 1;
```



$\wp(\mathbb{Z})$

$\mathbb{Z} \times \mathbb{Z}$

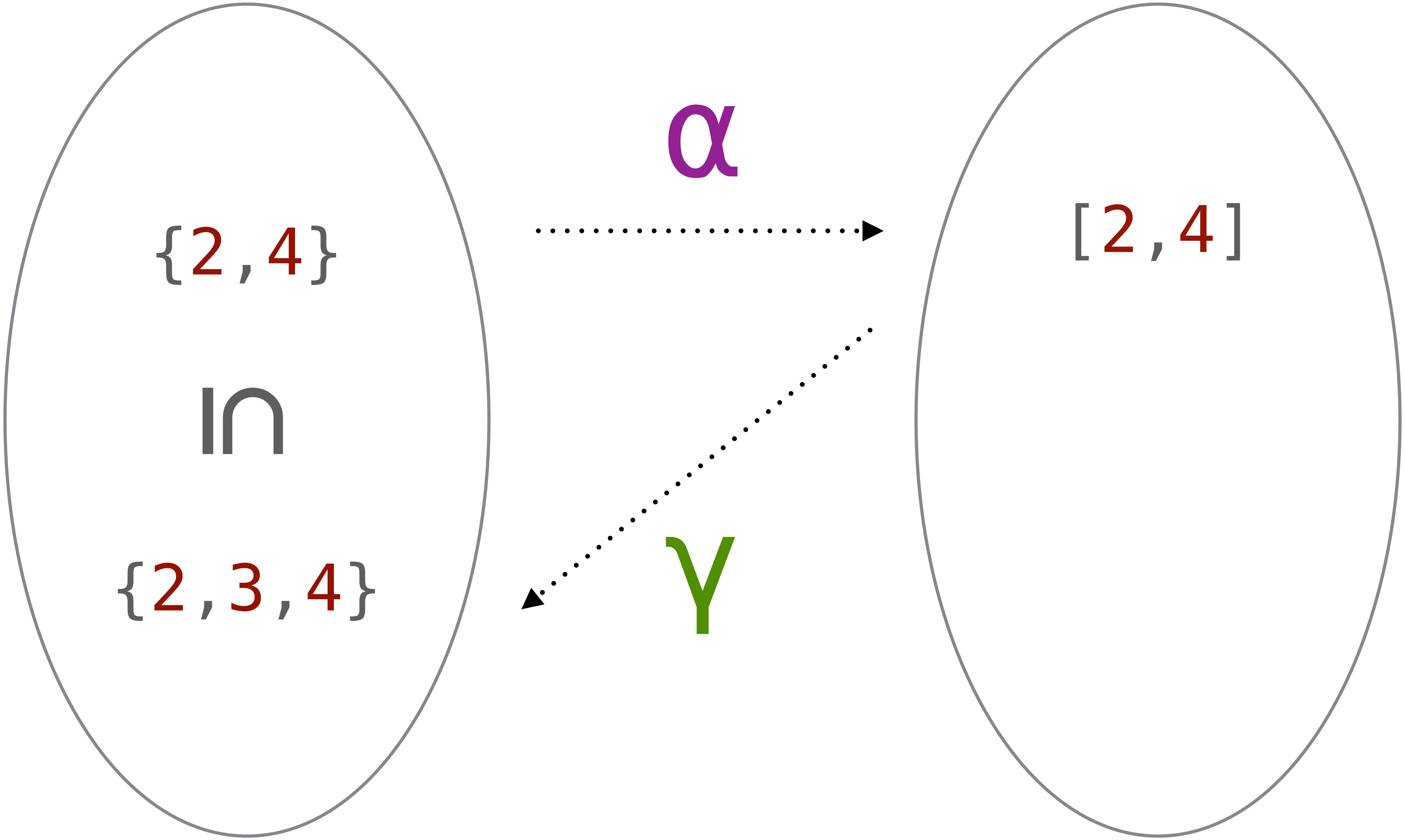
$x \in \{2,4\}$

$\subseteq$

$x \in \{2,3,4\}$

$x \in [2,4]$

$\wp(\mathbb{Z})$ $\mathbb{Z} \times \mathbb{Z}$



$\{2,4\}$

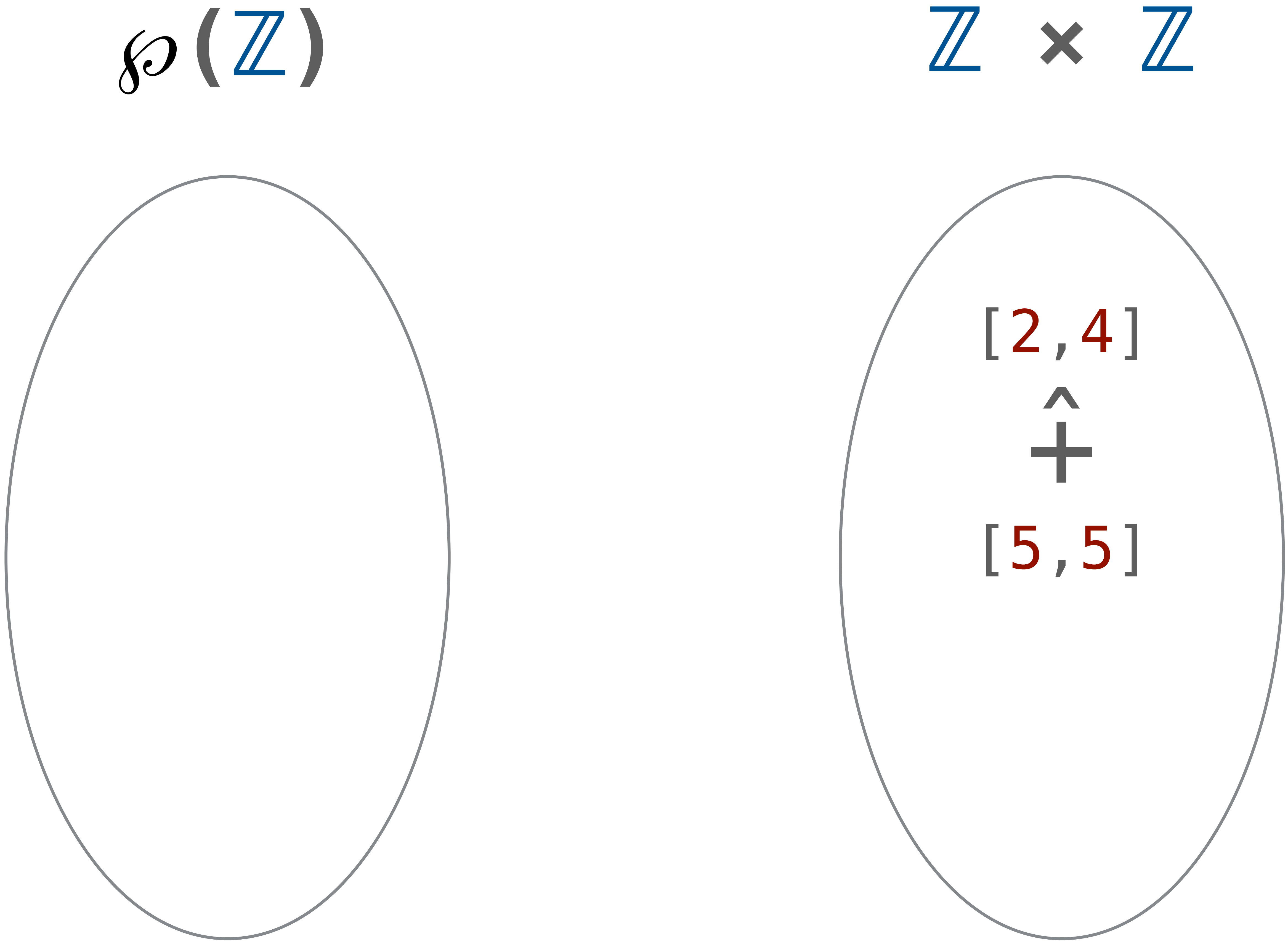
$\subseteq$

$\{2,3,4\}$

α

γ

$[2,4]$



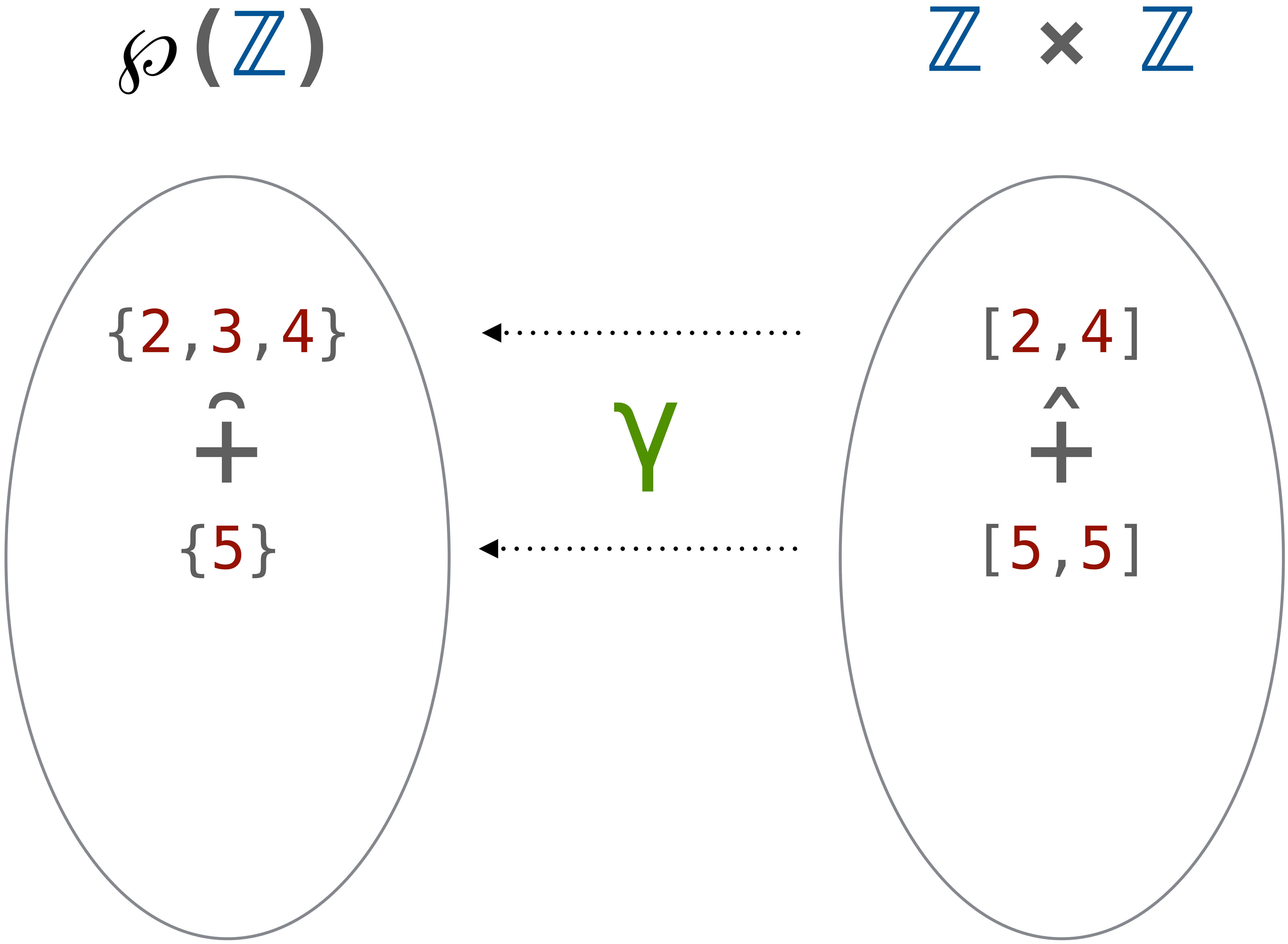
$\wp(\mathbb{Z})$

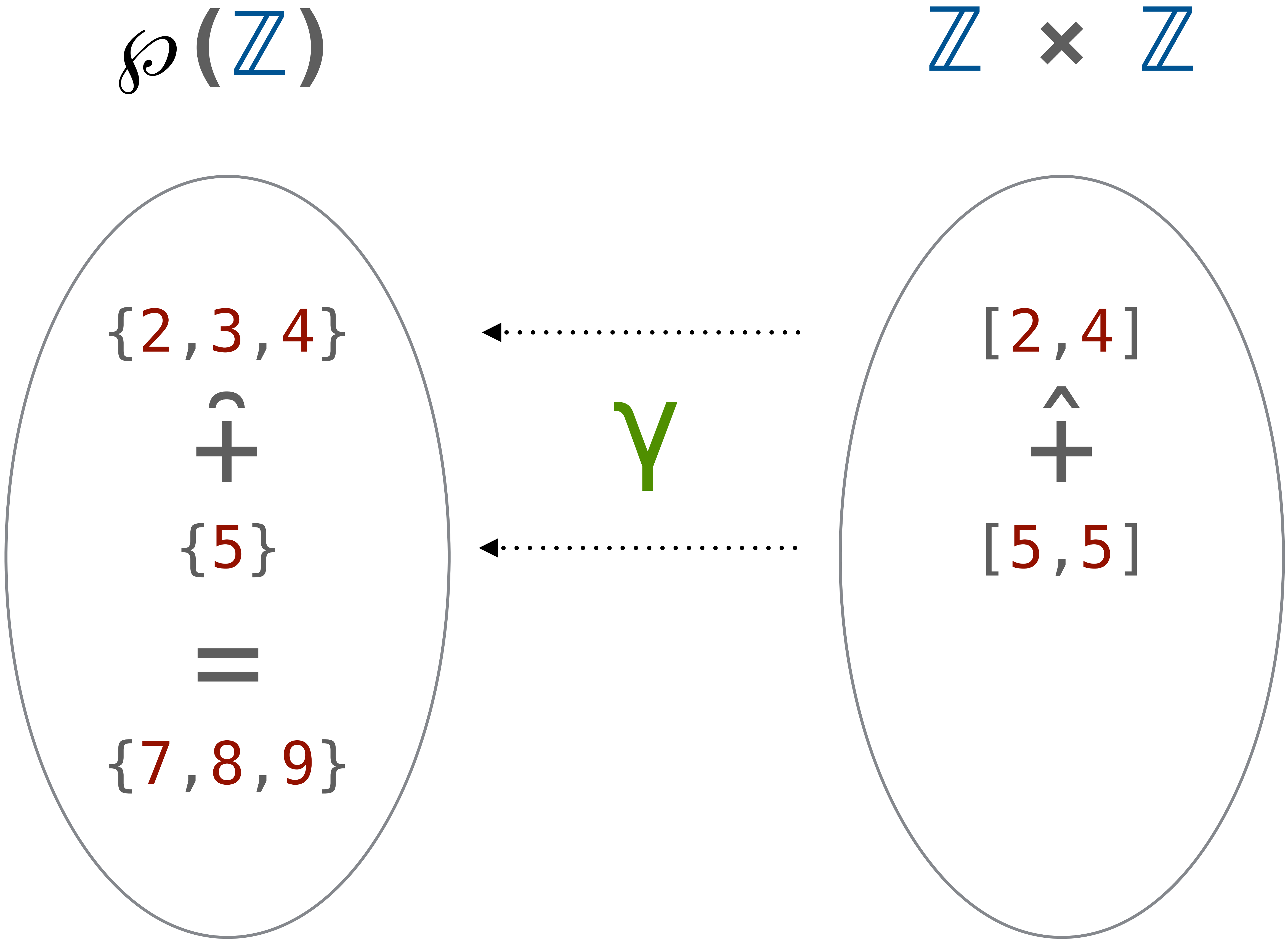
$\mathbb{Z} \times \mathbb{Z}$

$[2,4]$

$\hat{+}$

$[5,5]$


$\wp(\mathbb{Z})$
$\mathbb{Z} \times \mathbb{Z}$
$\{2,3,4\}$
$\overset{\frown}{+}$
$\{5\}$
γ
$[2,4]$
$\hat{+}$
$[5,5]$



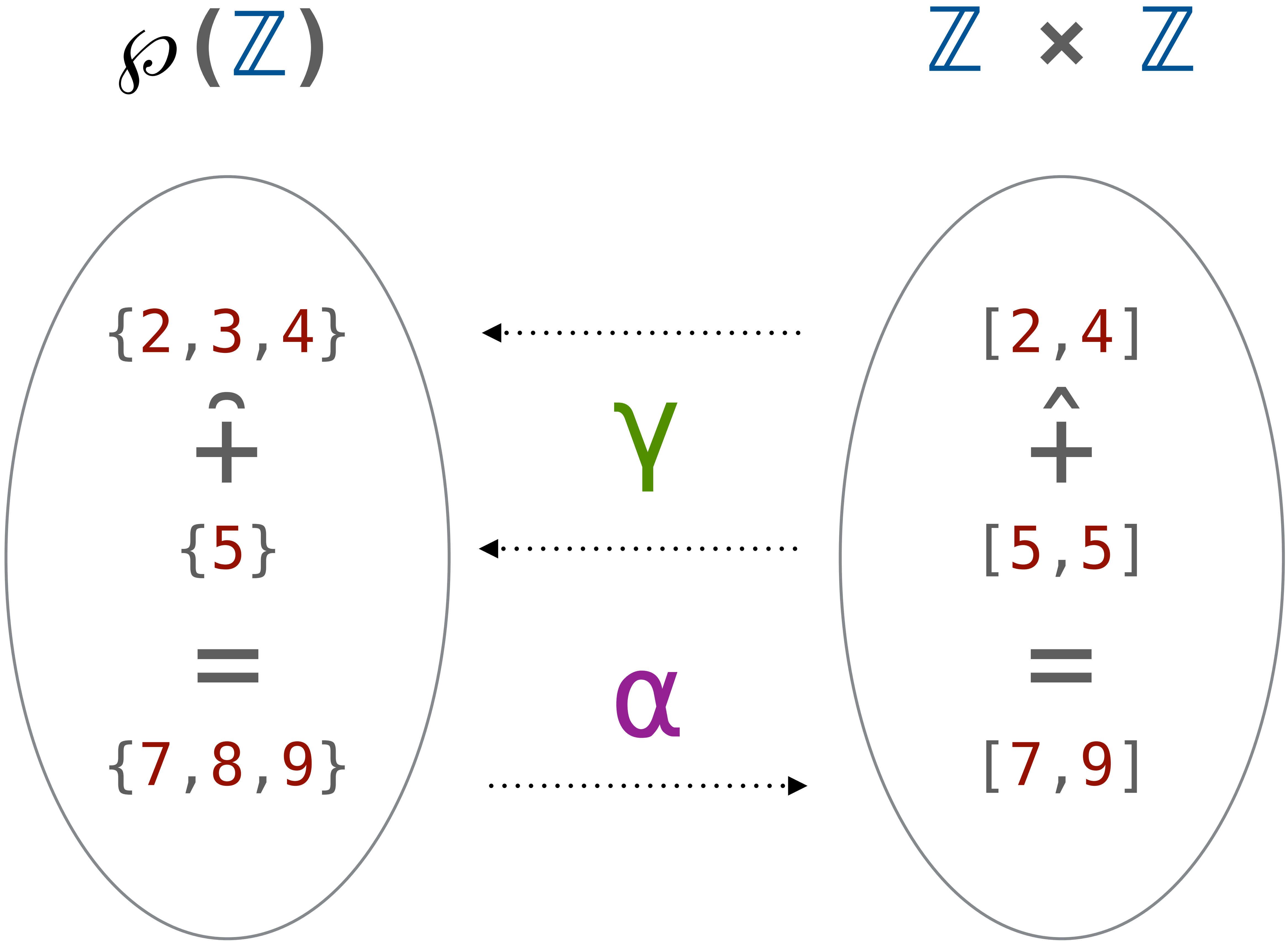
$\wp(\mathbb{Z})$

$\mathbb{Z} \times \mathbb{Z}$

$\{2,3,4\} \overset{\frown}{+} \{5\} = \{7,8,9\}$

$[2,4] \hat{+} [5,5]$

γ


$\wp(\mathbb{Z})$
$\mathbb{Z} \times \mathbb{Z}$
$\{2,3,4\}$
$\overset{\frown}{+}$
$\{5\}$
$=$
$\{7,8,9\}$
γ
α
$[2,4]$
$\hat{+}$
$[5,5]$
$=$
$[7,9]$

$$[2,4] \mathbin{\hat{+}} [5,5]$$

$$=$$

$$\alpha(\gamma([2,4]) \overset{\frown}{+} \gamma([5,5]))$$

$$\alpha(\gamma([2,4]) \overset{\frown}{+} \gamma([5,5]))$$

$$[2,4] \hat{+} [5,5]$$

$$\alpha(\gamma([2,4]) \mathbin{\hat{+}} \gamma([5,5]))$$

$$=$$

$$\alpha(\{\ i + j \mid i \in \gamma([2,4]) \\ \wedge\ j \in \gamma([5,5])\ \})$$

$$[2,4] \mathbin{\hat{+}} [5,5]$$

$$\alpha(\gamma([2,4]) \mathbin{\hat{+}} \gamma([5,5]))$$

$$=$$

$$\begin{aligned}\alpha(\{\ i + j \mid i \in \gamma([2,4]) \\ \land\ j \in \gamma([5,5])\ \})\end{aligned}$$

$$=$$

$$\alpha(\{7,8,9\})$$

$$[2,4] \mathbin{\hat{+}} [5,5]$$

$$\alpha(\gamma([2,4]) \mathbin{\hat{+}} \gamma([5,5]))$$

$$=$$

$$\begin{aligned}\alpha(\{\ i + j \mid i \in \gamma([2,4]) \\ \land\ j \in \gamma([5,5])\ \})\end{aligned}$$

$$=$$

$$\alpha(\{7,8,9\})$$

$$=$$

$$\alpha(\{7\}) \sqcup \alpha(\{8\}) \sqcup \alpha(\{9\})$$

$$[2,4] \mathbin{\hat{+}} [5,5]$$

$$\alpha(\gamma([2,4]) \overset{\frown}{+} \gamma([5,5]))$$

$$=$$

$$\alpha(\{\ i + j \mid i \in \gamma([2,4]) \\ \wedge\ j \in \gamma([5,5])\ \})$$

$$=$$

$$\alpha(\{7,8,9\})$$

$$=$$

$$\alpha(\{7\}) \sqcup \alpha(\{8\}) \sqcup \alpha(\{9\})$$

$$=$$

$$[7,9]$$

$$[2,4] \mathbin{\hat{+}} [5,5]$$

$$\alpha(\gamma([2,4]) \hat{+} \gamma([5,5]))$$

$$=$$

$$\alpha(\{\, i + j \mid i \in \gamma([2,4]) \wedge j \in \gamma([5,5]) \,\})$$

$$=$$

$$\alpha(\{7,8,9\})$$

$$=$$

$$\alpha(\{7\}) \sqcup \alpha(\{8\}) \sqcup \alpha(\{9\})$$

$$=$$

$$[7,9]$$

$$\triangleq$$

$$[2,4] \hat{+} [5,5]$$

$$\alpha(\gamma([w,x]) \overset{\frown}{+} \gamma([y,z]))$$

$$=$$

$$\alpha(\{\ i + j \mid i \in \gamma([w,x]) \\ \land\ j \in \gamma([y,z])\ \})$$

$$=$$

$$\alpha(\{w+y, \ldots, x+z\})$$

$$=$$

$$\alpha(\{w+y\}) \sqcup \cdots \sqcup \alpha(\{x+z\})$$

$$=$$

$$[w+y, x+z]$$

$$\triangleq$$

$$[w,x] \hat{+} [y,z]$$

Spec

$$\alpha(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z]))$$

$$=$$

$$\begin{aligned}\alpha(\{\ i + j \mid\ & i \in \gamma([w,x]) \\ & \wedge\ j \in \gamma([y,z])\ \})\end{aligned}$$

$$=$$

$$\alpha(\{w+y,\dots,x+z\})$$

$$=$$

$$\alpha(\{w+y\}) \sqcup \cdots \sqcup \alpha(\{x+z\})$$

$$=$$

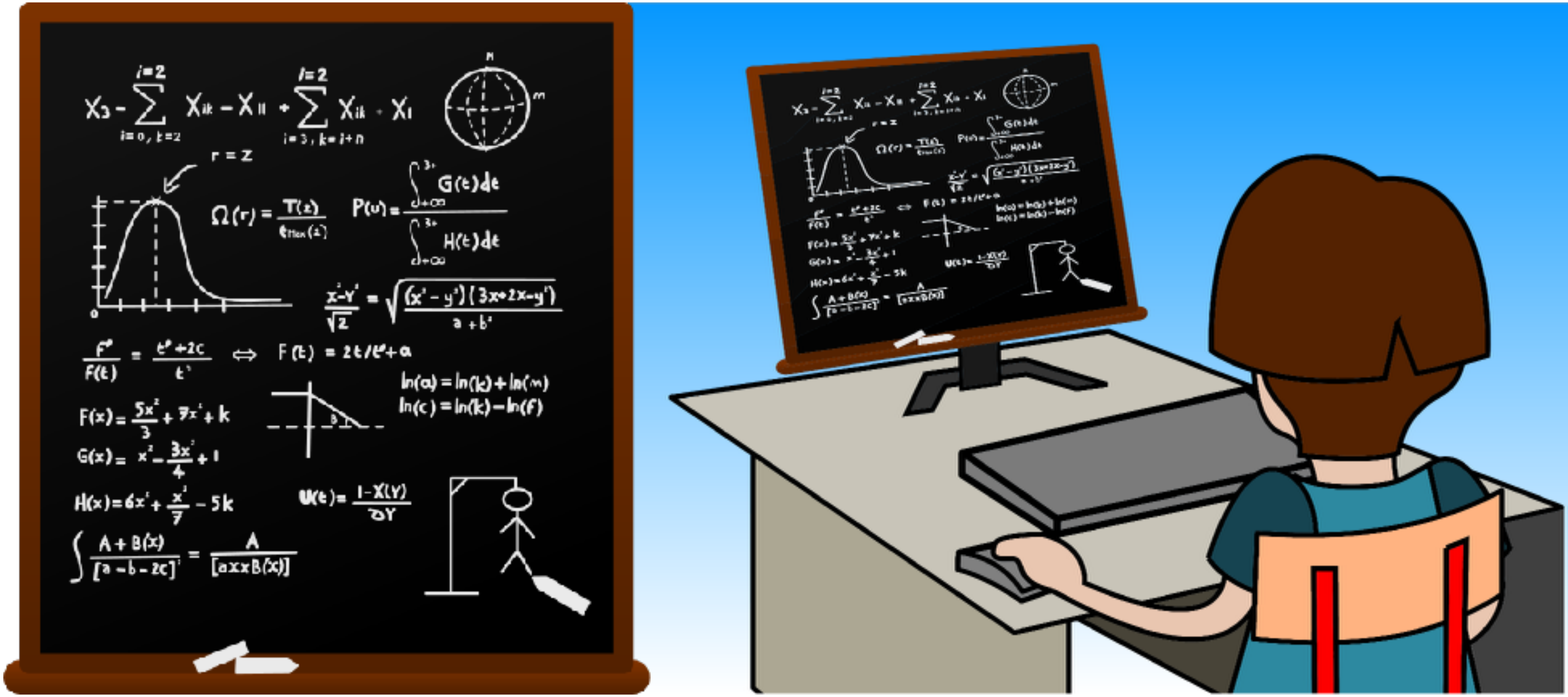
Algorithm

$$[w+y,x+z]$$

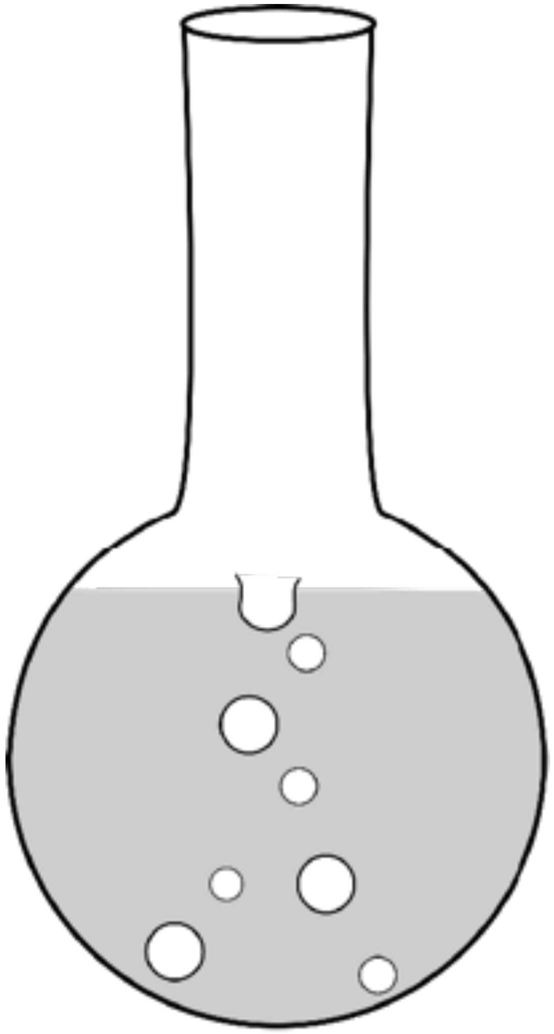
$$\triangleq$$

$$[w,x] \mathbin{\hat{+}} [y,z]$$

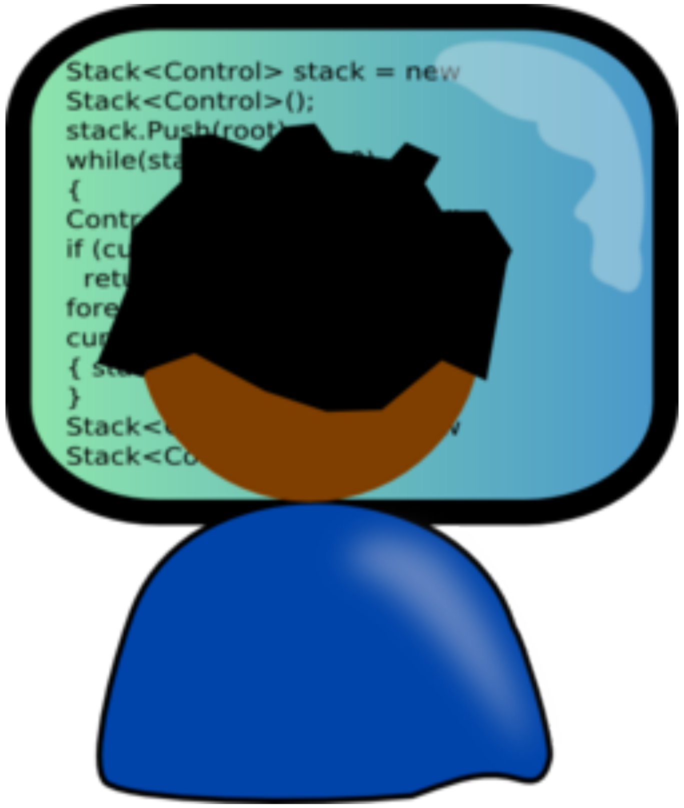
Mechanized Verification (MV)



Traditional Approach

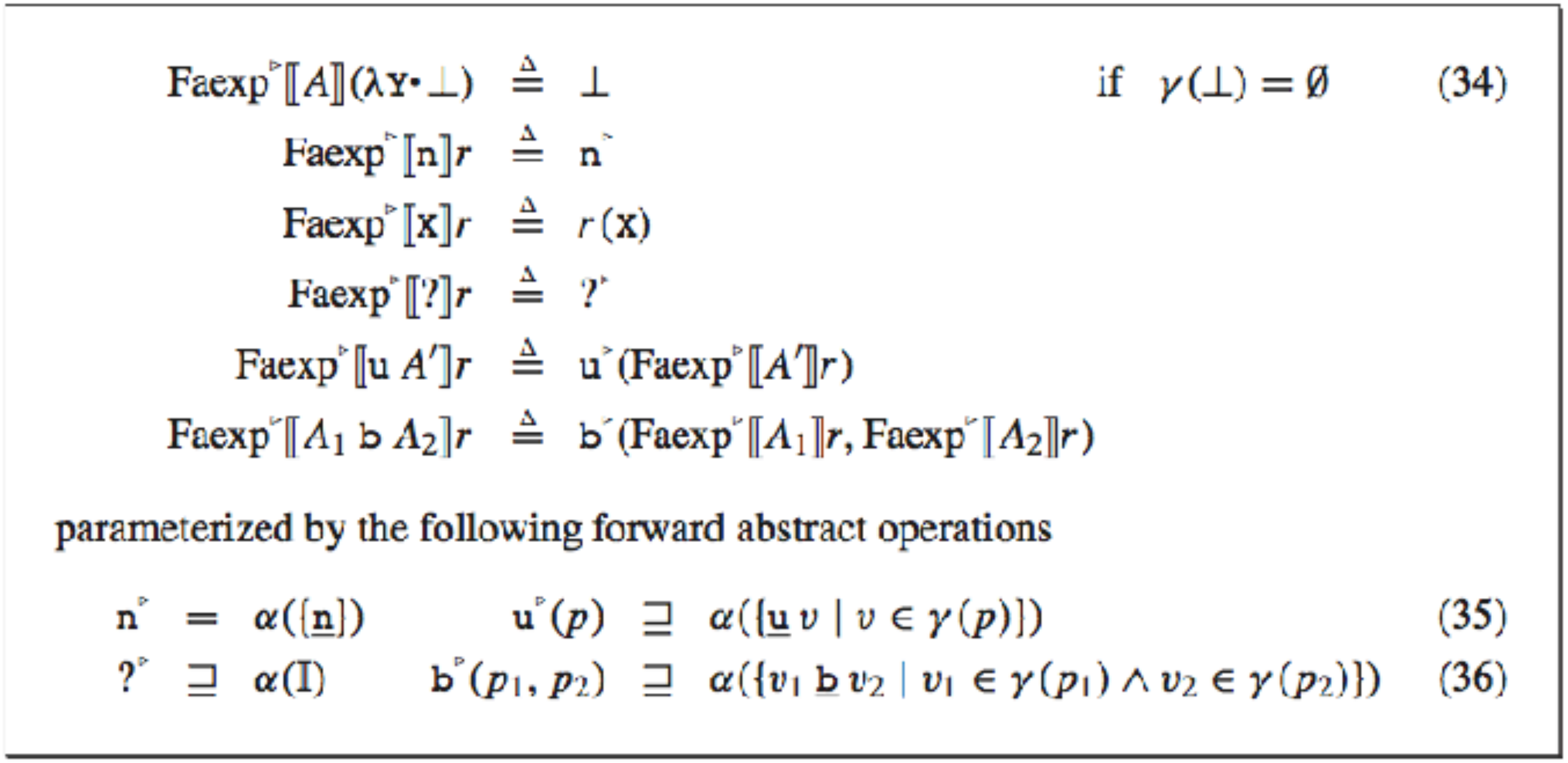


Verified Model



```
Stack<Control> stack = new
Stack<Control>();
stack.Push(root)
while(sta
{
Contr
if (cu
  retu
fore
cu
{ st
}
Stack<
Stack<Co
```

Traditional Approach



$$\mathrm{Faexp}^{\triangleright}[\![A]\!](\lambda \mathrm{Y}\cdot\bot) \triangleq \bot \qquad \text{if} \quad \gamma(\bot)=\emptyset \qquad (34)$$

$$\mathrm{Faexp}^{\triangleright}[\![\mathrm{n}]\!]r \triangleq \mathrm{n}^{\triangleright}$$

$$\mathrm{Faexp}^{\triangleright}[\![\mathrm{X}]\!]r \triangleq r(\mathrm{X})$$

$$\mathrm{Faexp}^{\triangleright}[\![?]\!]r \triangleq ?^{\triangleright}$$

$$\mathrm{Faexp}^{\triangleright}[\![\mathrm{u}\ A']\!]r \triangleq \mathrm{u}^{\triangleright}(\mathrm{Faexp}^{\triangleright}[\![A']\!]r)$$

$$\mathrm{Faexp}^{\triangleright}[\![A_1\ \mathrm{b}\ A_2]\!]r \triangleq \mathrm{b}^{\triangleright}(\mathrm{Faexp}^{\triangleright}[\![A_1]\!]r, \mathrm{Faexp}^{\triangleright}[\![A_2]\!]r)$$

parameterized by the following forward abstract operations

$$\mathrm{n}^{\triangleright} = \alpha(\{\underline{\mathrm{n}}\}) \qquad \mathrm{u}^{\triangleright}(p) \sqsupseteq \alpha(\{\underline{\mathrm{u}}\, v \mid v \in \gamma(p)\}) \qquad (35)$$

$$?^{\triangleright} \sqsupseteq \alpha(\mathbb{I}) \qquad \mathrm{b}^{\triangleright}(p_1, p_2) \sqsupseteq \alpha(\{v_1\ \underline{\mathrm{b}}\ v_2 \mid v_1 \in \gamma(p_1) \wedge v_2 \in \gamma(p_2)\}) \qquad (36)$$

Figure 6: Forward abstract interpretation of arithmetic expressions

–The Calculational Design of a Generic Abstract Interpreter
[Cousot, 1998]

Traditional Approach

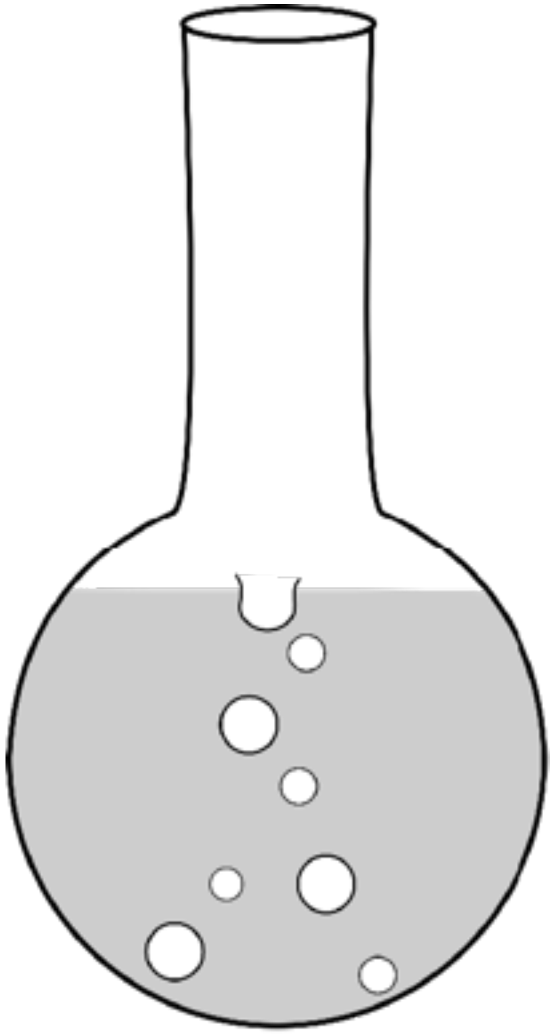
```
...
(*** bug corrected on 02/09/2000 ***)
(*  let b_unary b_uop r x =        *)
    let b_unary b_uop x r =
...
(*** bug corrected on 02/09/2000 ***)
(*  let b_binary b_bop r x y =     *)
    let b_binary b_bop x y r =
...
```

–CDGAI Errata [Cousot, 2000]

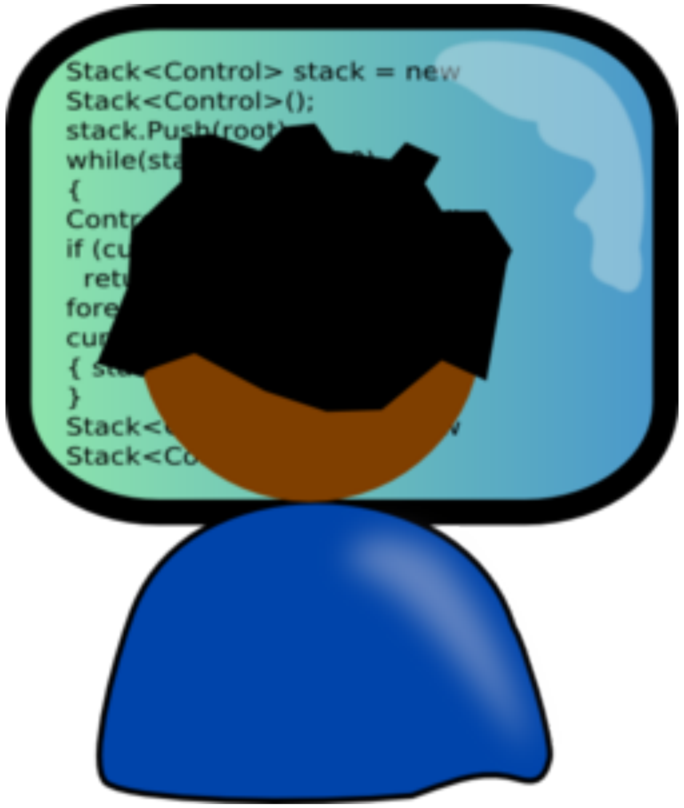
Traditional Approach

> “Beware of bugs in the above code;
> I have only proved it correct, not tried it.”
>
> –Donald Knuth

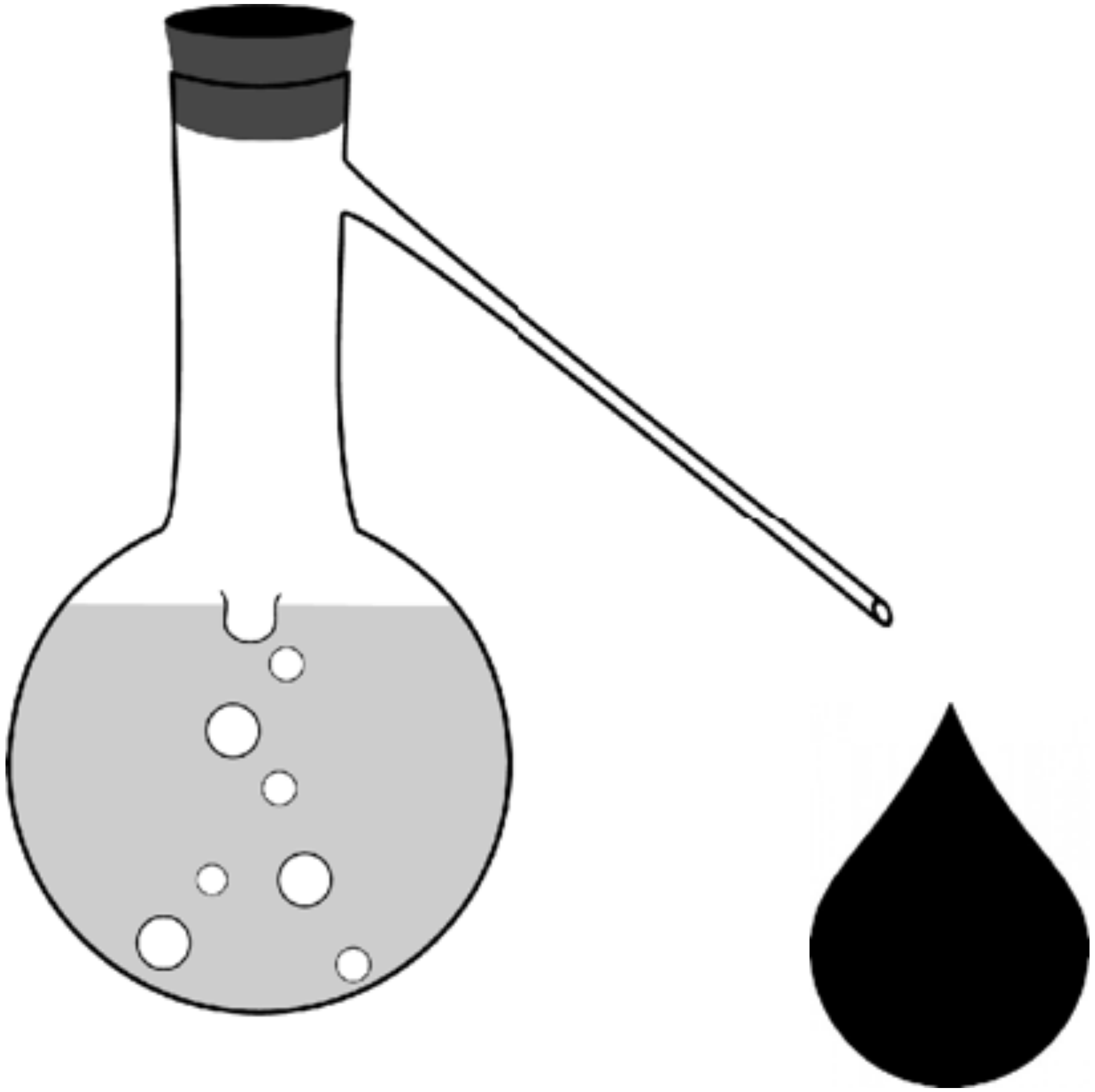
Traditional Approach



Verified Model



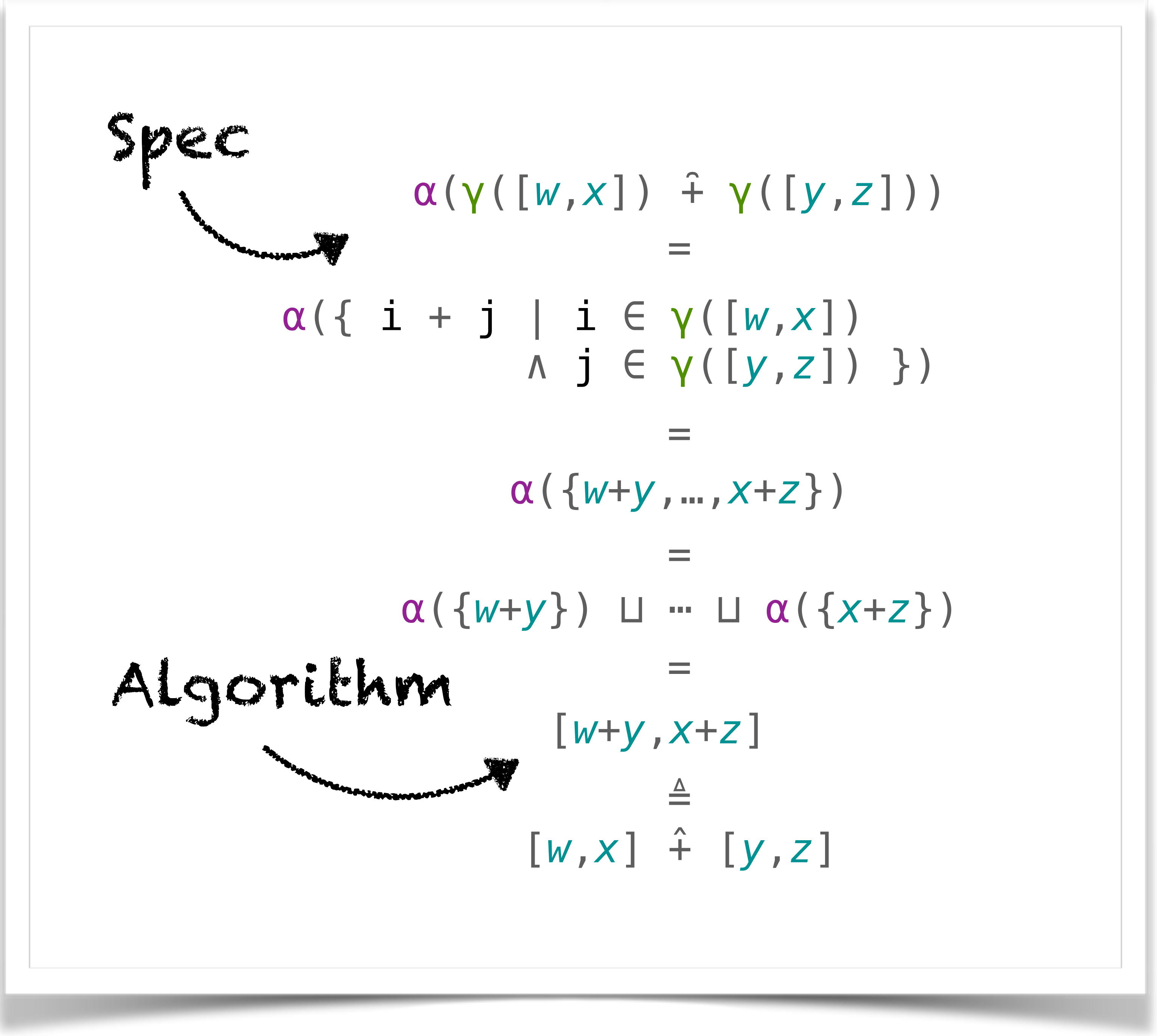
Mechanized Verification



Verified Model

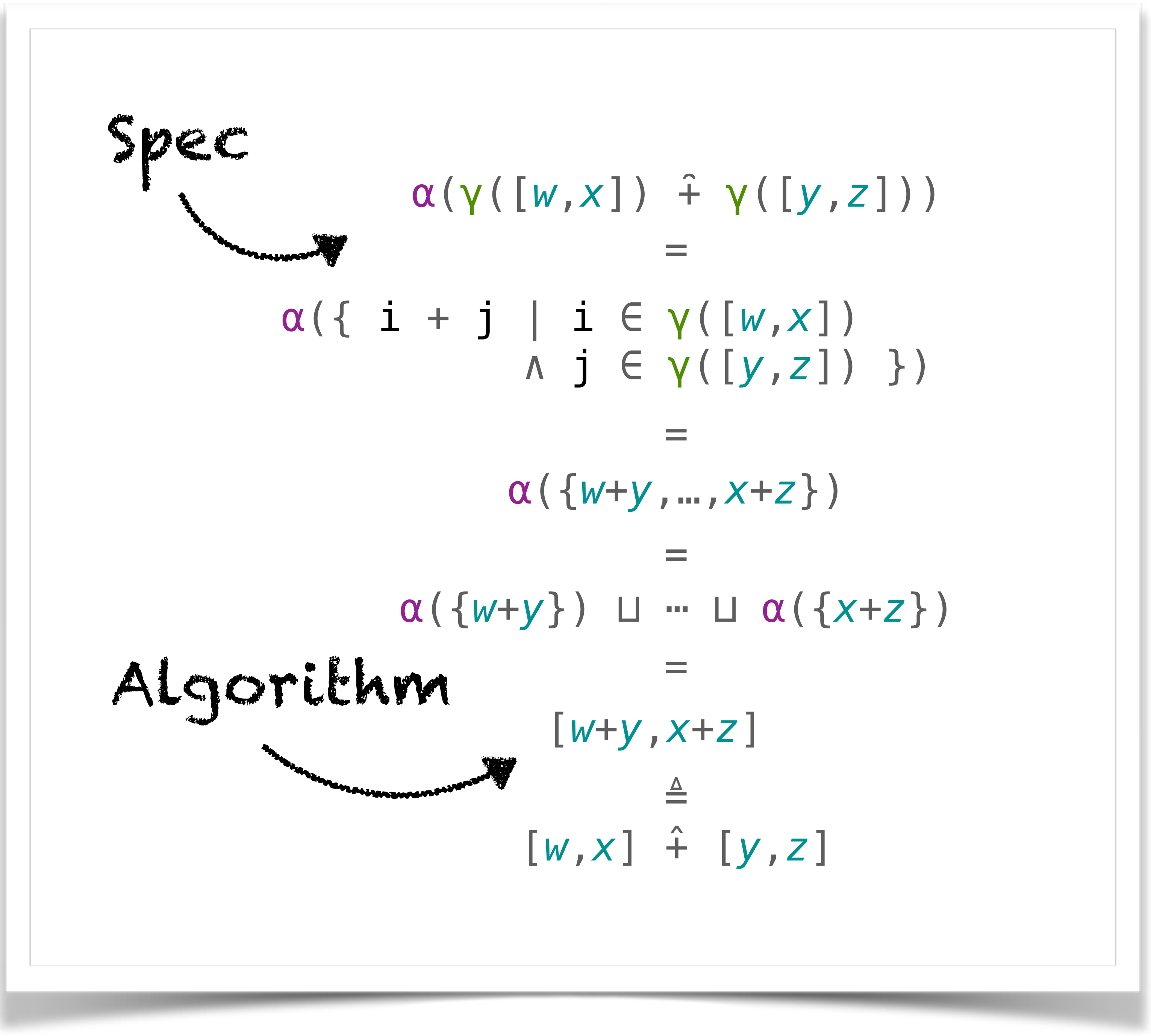
Certified Implementation

The Plan



$$
\begin{gathered}
\alpha(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z])) \\
= \\
\alpha(\{\ i + j \mid i \in \gamma([w,x]) \\
\qquad\qquad \land\ j \in \gamma([y,z])\ \}) \\
= \\
\alpha(\{w+y,\ldots,x+z\}) \\
= \\
\alpha(\{w+y\}) \sqcup \cdots \sqcup \alpha(\{x+z\}) \\
= \\
[w+y,x+z] \\
\triangleq \\
[w,x] \mathbin{\hat{+}} [y,z]
\end{gathered}
$$

The Plan



Spec

$$\begin{aligned}
&\alpha(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z])) \\
&= \\
&\alpha(\{\, i + j \mid i \in \gamma([w,x]) \\
&\qquad \wedge j \in \gamma([y,z]) \,\}) \\
&= \\
&\alpha(\{w+y,\dots,x+z\}) \\
&= \\
&\alpha(\{w+y\}) \sqcup \cdots \sqcup \alpha(\{x+z\}) \\
&= \\
&[w+y,x+z] \\
&\triangleq \\
&[w,x] \mathbin{\hat{+}} [y,z]
\end{aligned}$$

Algorithm

Step 1:
Check These Calculations
Using a Proof Assistant

The Plan

Spec

$$\alpha(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z]))$$

$$=$$

$$\alpha(\{\ i + j \mid i \in \gamma([w,x]) \land j \in \gamma([y,z])\ \})$$

$$=$$

$$\alpha(\{w+y,\dots,x+z\})$$

$$=$$

$$\alpha(\{w+y\}) \sqcup \cdots \sqcup \alpha(\{x+z\})$$

$$=$$

Algorithm

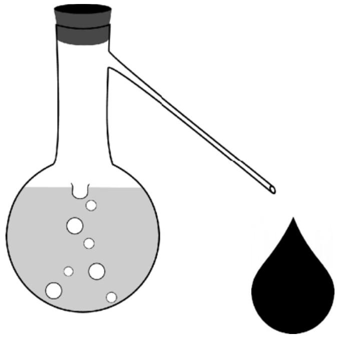
$$[w+y,x+z]$$

$$\triangleq$$

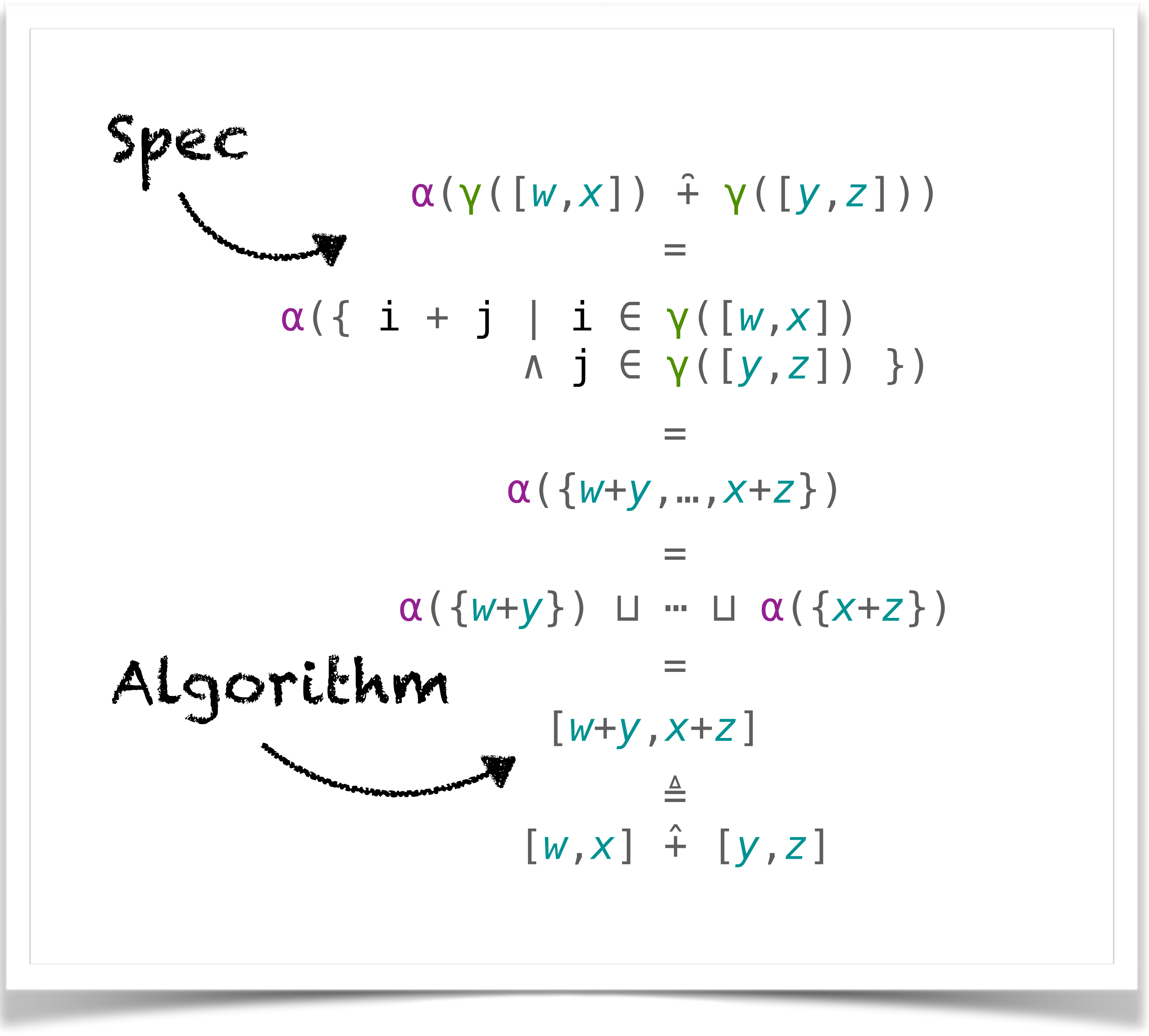
$$[w,x] \mathbin{\hat{+}} [y,z]$$

Step 1:
Check These Calculations
Using a Proof Assistant

Step 2:
Extract a Certified
Implementation



The Plan



Spec

$$\alpha(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z]))$$

$$=$$

$$\alpha(\{\ i + j \mid i \in \gamma([w,x]) \wedge j \in \gamma([y,z])\ \})$$

$$=$$

$$\alpha(\{w+y,\ldots,x+z\})$$

$$=$$

$$\alpha(\{w+y\}) \sqcup \cdots \sqcup \alpha(\{x+z\})$$

$$=$$

Algorithm

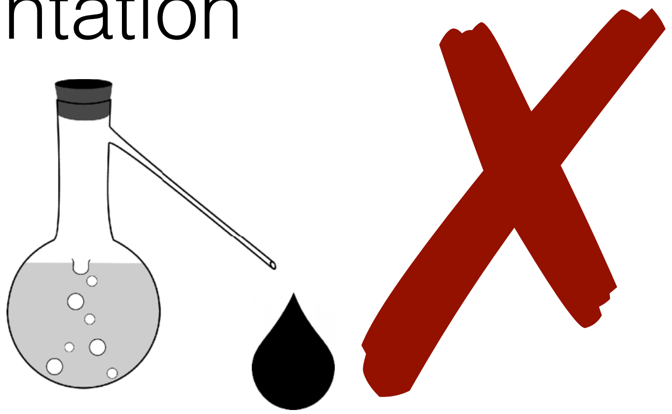
$$[w+y,x+z]$$

$$\triangleq$$

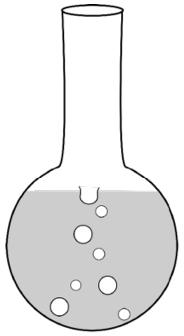
$$[w,x] \mathbin{\hat{+}} [y,z]$$

Step 1:
Check These Calculations
Using a Proof Assistant

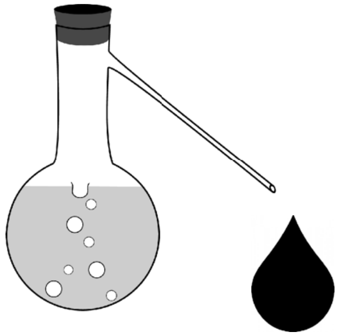
Step 2:
Extract a Certified
Implementation



“This looks like an algorithm”
(to a human)

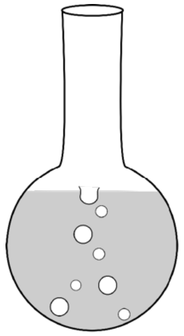


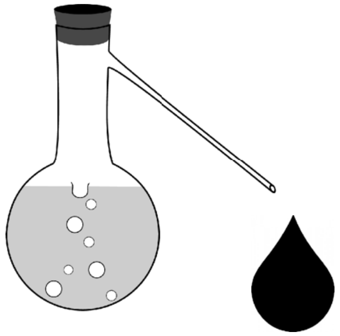
“I know how to execute this”
(to a machine)

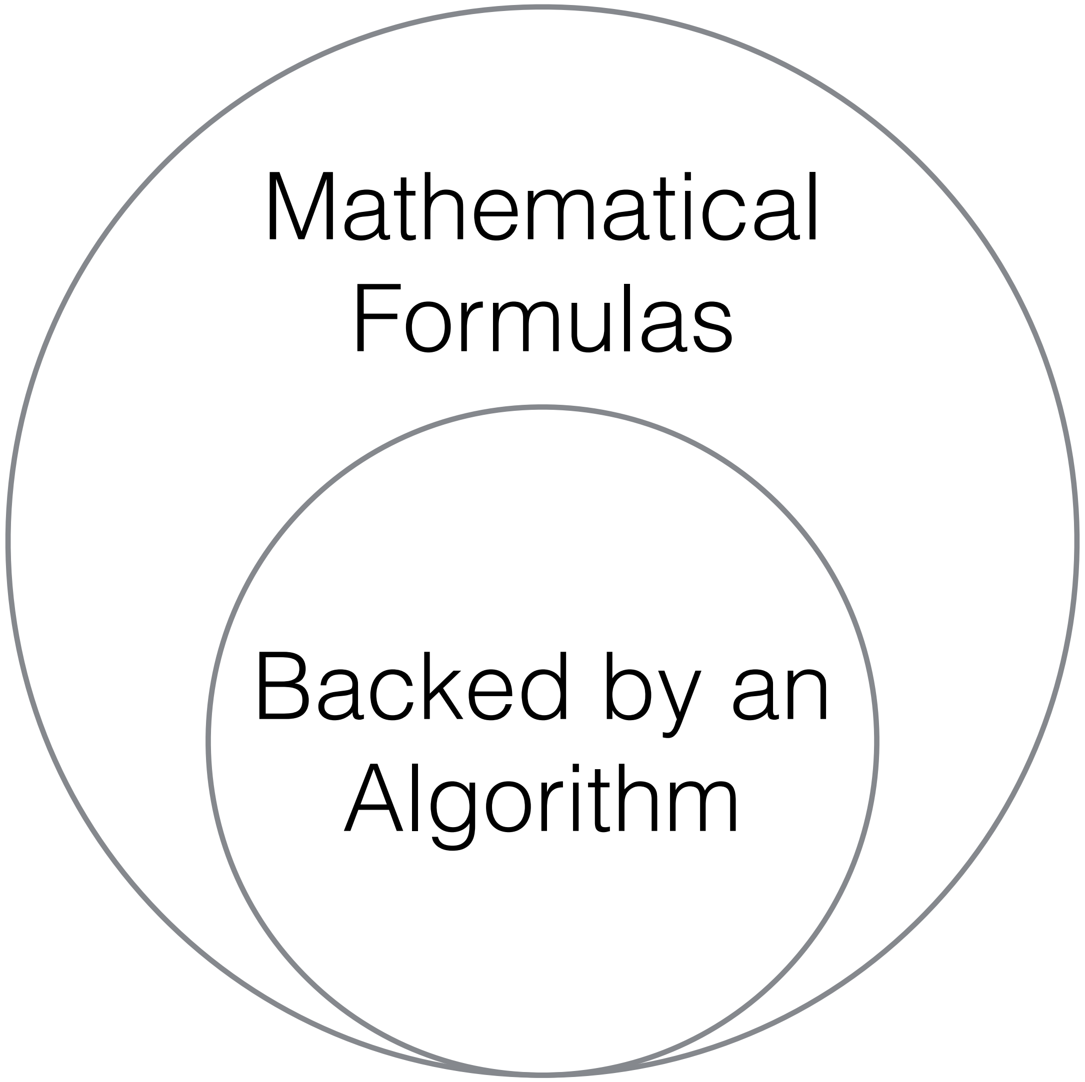


“This looks like an algorithm”
(to a human)

“I know how to execute this”
(to a machine)



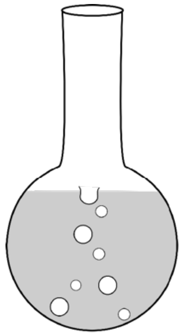




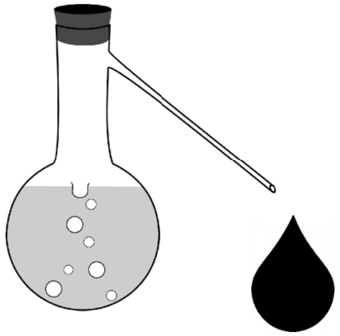
Mathematical Formulas

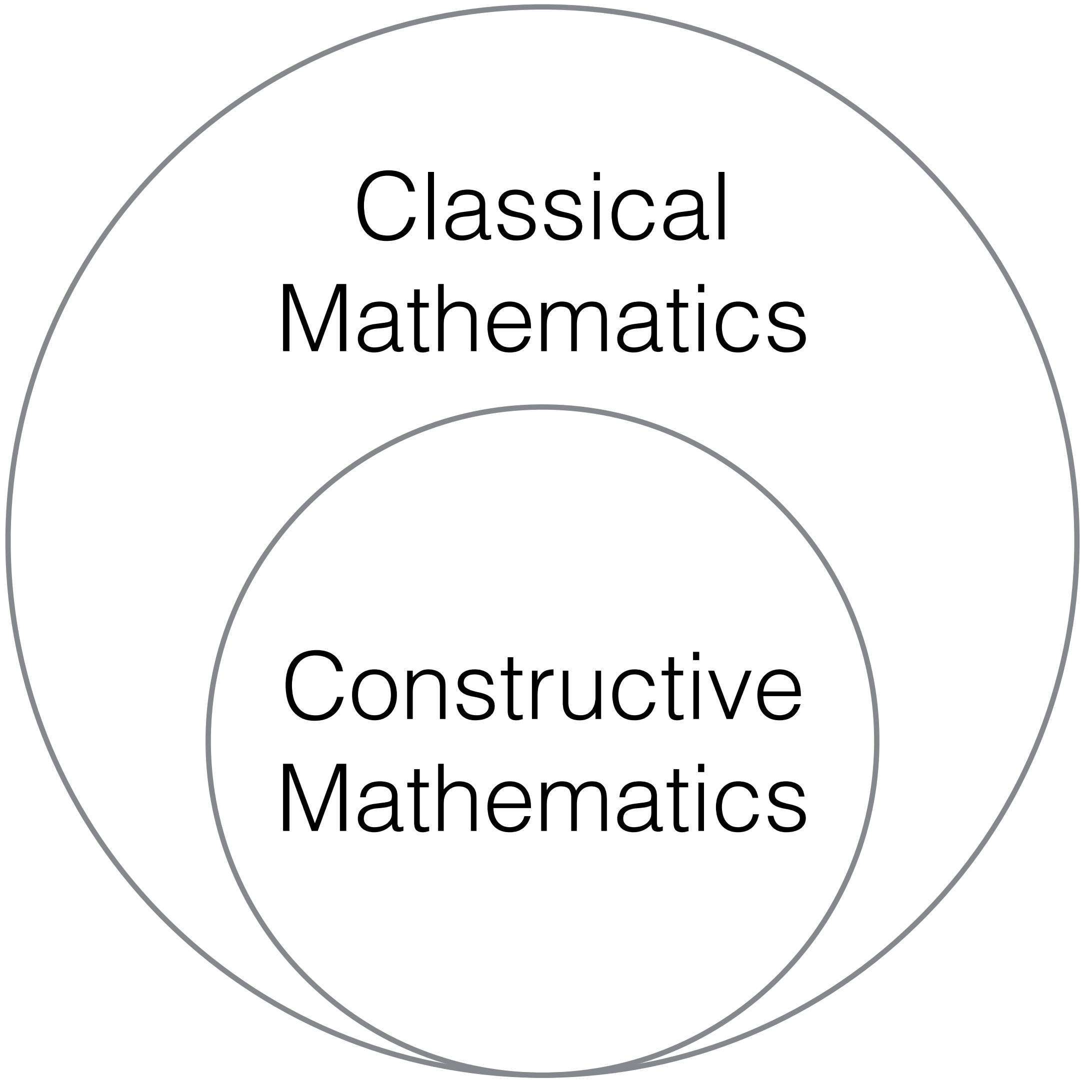
Backed by an Algorithm

“This looks like an algorithm”
(to a human)



“I know how to execute this”
(to a machine)

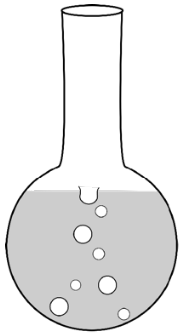




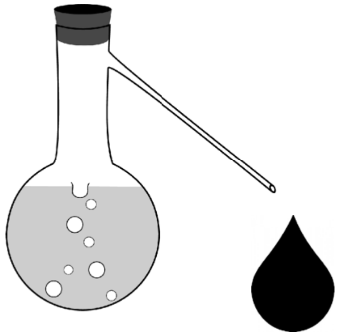
Classical
Mathematics

Constructive
Mathematics

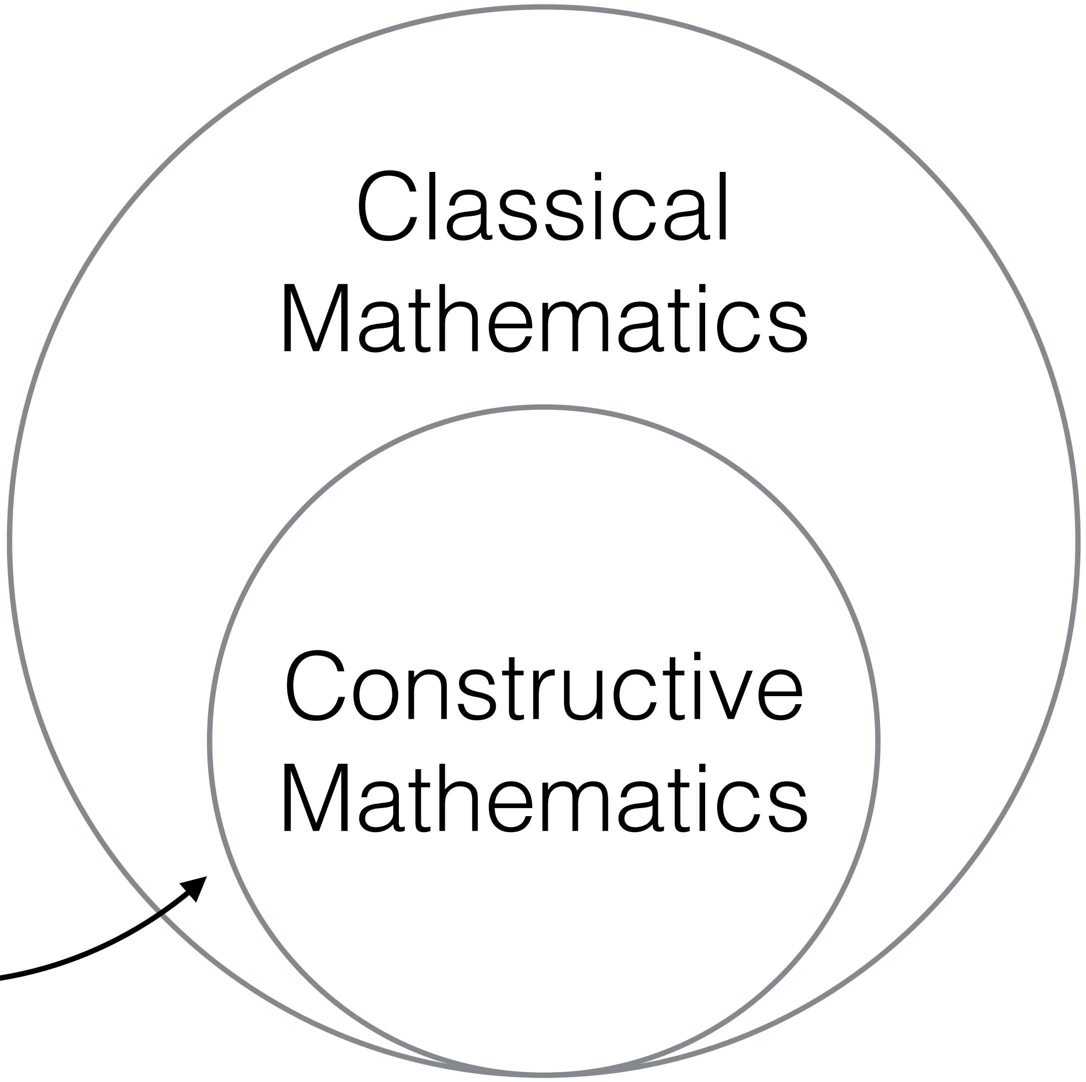
"This looks like an algorithm"
(to a human)



"I know how to execute this"
(to a machine)



PROBLEM:
how to know when boundary is crossed



Classical Mathematics

Constructive Mathematics

SOLUTION:
explicitly account for algorithmic content

Classical Galois Connections

$$\wp(\mathbb{Z}) \xrightarrow{\alpha} \mathbb{Z} \times \mathbb{Z}$$

Constructive Galois Connections

$$\mathbb{Z} \xrightarrow{\eta} \mathbb{Z} \times \mathbb{Z}$$

Constructive Galois Connections

$$\mathbb{Z} \xrightarrow{\eta} \mathbb{Z} \times \mathbb{Z}$$

defn

$$\eta(i) := [i,i]$$

algorithmic content of abstraction

Constructive Galois Connections

$$\mathbb{Z} \xrightarrow{\eta} \mathbb{Z} \times \mathbb{Z}$$

defn

$$\eta(i) \coloneqq [i,i]$$

Law 1

$$\alpha = \langle\eta\rangle$$

embedding algorithms

Constructive Galois Connections

$$\mathbb{Z} \xrightarrow{\eta} \mathbb{Z} \times \mathbb{Z}$$

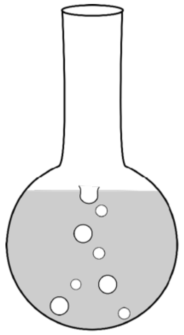
defn $\eta(i) \coloneqq [i,i]$

Law 1 $\alpha = \langle \eta \rangle$

Law 2 $\langle \eta \rangle(\{x\}) = \langle \eta(x) \rangle$

singleton
powersets
compute

$$
\begin{gathered}
\alpha(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z])) \\
= \\
\alpha(\{\ \mathtt{i} + \mathtt{j} \mid \mathtt{i} \in \gamma([w,x]) \\
\qquad\qquad \wedge\ \mathtt{j} \in \gamma([y,z])\ \}) \\
= \\
\alpha(\{w+y,\dots,x+z\}) \\
= \\
\alpha(\{w+y\}) \sqcup \cdots \sqcup \alpha(\{x+z\}) \\
= \\
[w+y,x+z] \\
\triangleq \\
[w,x] \mathbin{\hat{+}} [y,z]
\end{gathered}
$$



$$\alpha(\gamma([w, x]) \overset{\frown}{+} \gamma([y, z]))$$

$$\alpha(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z]))$$

Law 1

$$\alpha = \langle \eta \rangle$$

$$\alpha(\gamma([w,x]) \overset{\frown}{+} \gamma([y,z]))$$

$$\underline{\langle \eta \rangle}(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z]))$$

$$\langle\eta\rangle(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z]))$$

$$=$$

$$\langle\eta\rangle(\{\ i + j \mid i \in \gamma([w,x]) \\ \land\ j \in \gamma([y,z])\ \})$$

$$=$$

$$\langle\eta\rangle(\{w+y,\ldots,x+z\})$$

$$=$$

$$\langle\eta\rangle(\{w+y\}) \sqcup \cdots \sqcup \langle\eta\rangle(\{x+z\})$$

$$\langle\eta\rangle(\gamma([w,x]) \hat{+} \gamma([y,z]))$$

$$=$$

$$\langle\eta\rangle(\{\ i + j \mid i \in \gamma([w,x]) \wedge j \in \gamma([y,z])\ \})$$

$$=$$

$$\langle\eta\rangle(\{w+y,\ldots,x+z\})$$

$$=$$

$$\langle\eta\rangle(\{w+y\}) \sqcup \cdots \sqcup \langle\eta\rangle(\{x+z\})$$

Law 2

$$\langle\eta\rangle(\{x\}) = \langle\eta(x)\rangle$$

$$\langle\eta\rangle(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z]))$$

$$=$$

$$\langle\eta\rangle(\{\ i + j \mid i \in \gamma([w,x]) \\ \land\ j \in \gamma([y,z])\ \})$$

$$=$$

$$\langle\eta\rangle(\{w+y,\ldots,x+z\})$$

$$=$$

$$\langle\eta\rangle(\{w+y\}) \sqcup \cdots \sqcup \langle\eta\rangle(\{x+z\})$$

$$\langle\eta\rangle(\gamma([w,x]) \mathbin{\hat{+}} \gamma([y,z]))$$

$$=$$

$$\begin{aligned}\langle\eta\rangle(\{\ i + j \mid i \in \gamma([w,x]) \\ \land\ j \in \gamma([y,z])\ \})\end{aligned}$$

$$=$$

$$\langle\eta\rangle(\{w+y,\ldots,x+z\})$$

$$=$$

$$\underline{\langle\eta(w+y)\rangle} \sqcup \cdots \sqcup \underline{\langle\eta(x+z)\rangle}$$

$$\langle\eta\rangle(\gamma([w,x]) \overset{\frown}{+} \gamma([y,z]))$$

$$=$$

$$\langle\eta\rangle(\{\ i + j \mid i \in \gamma([w,x]) \\ \wedge\ j \in \gamma([y,z])\ \})$$

$$=$$

$$\langle\eta\rangle(\{w+y,\ldots,x+z\})$$

$$=$$

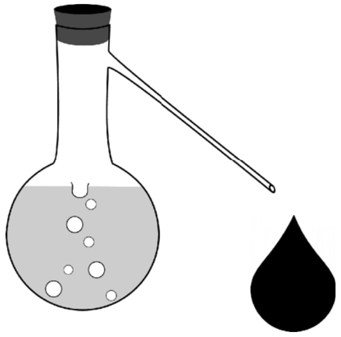
$$\langle\eta(w+y)\rangle \sqcup \cdots \sqcup \langle\eta(x+z)\rangle$$

$$=$$

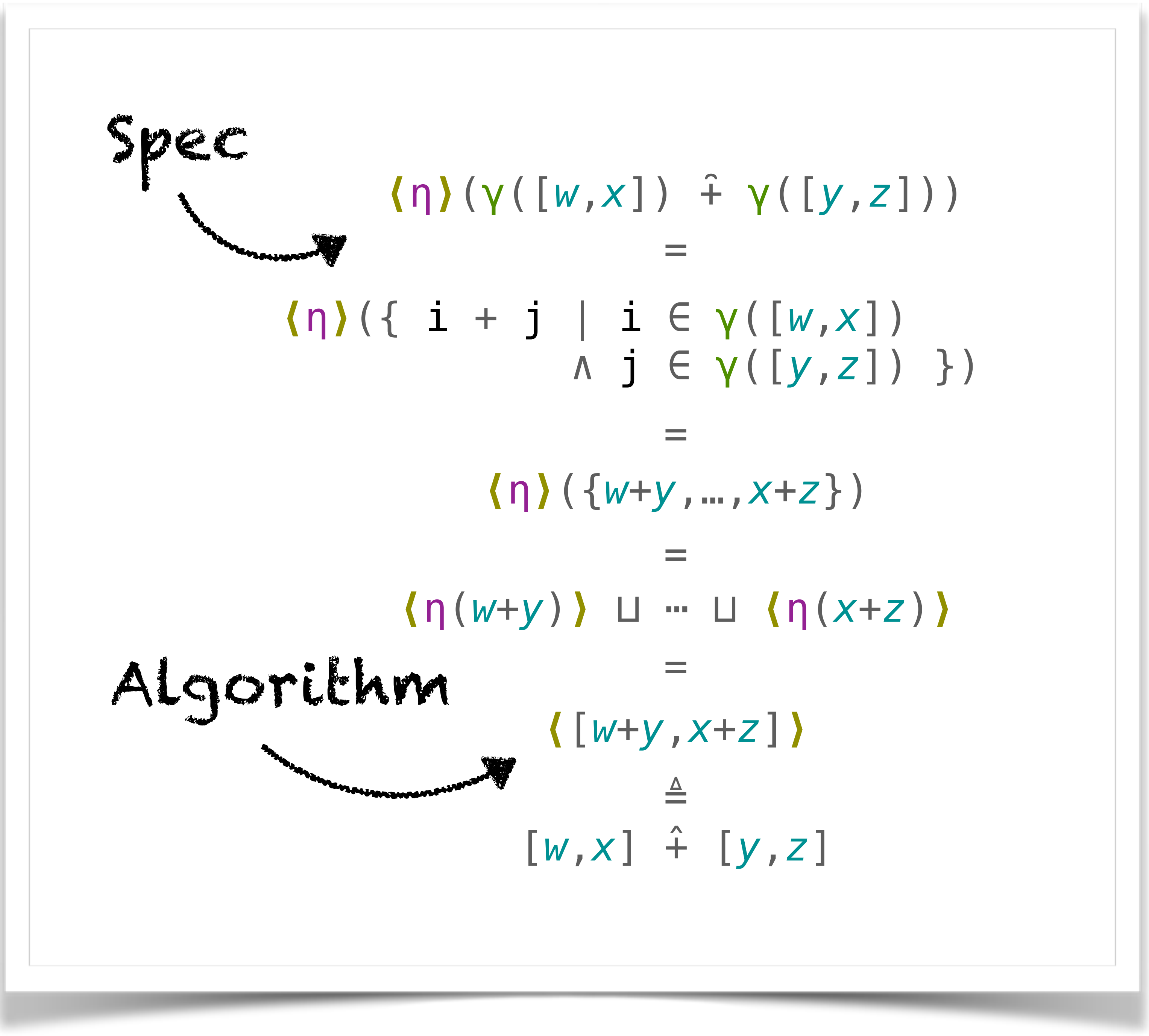
$$\langle[w+y,x+z]\rangle$$

$$\triangleq$$

$$[w,x] \mathbin{\hat{+}} [y,z]$$



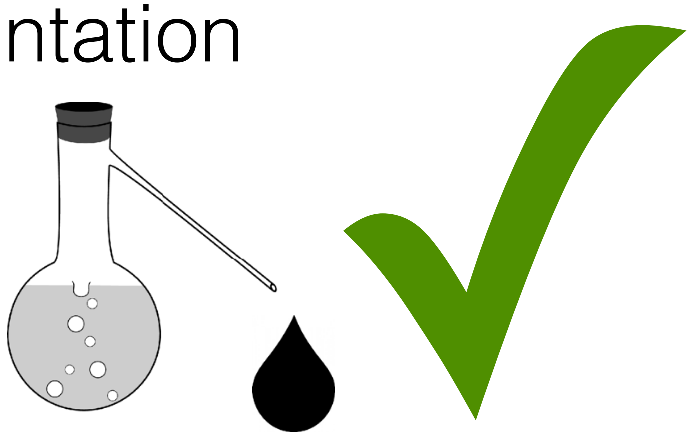
The Plan

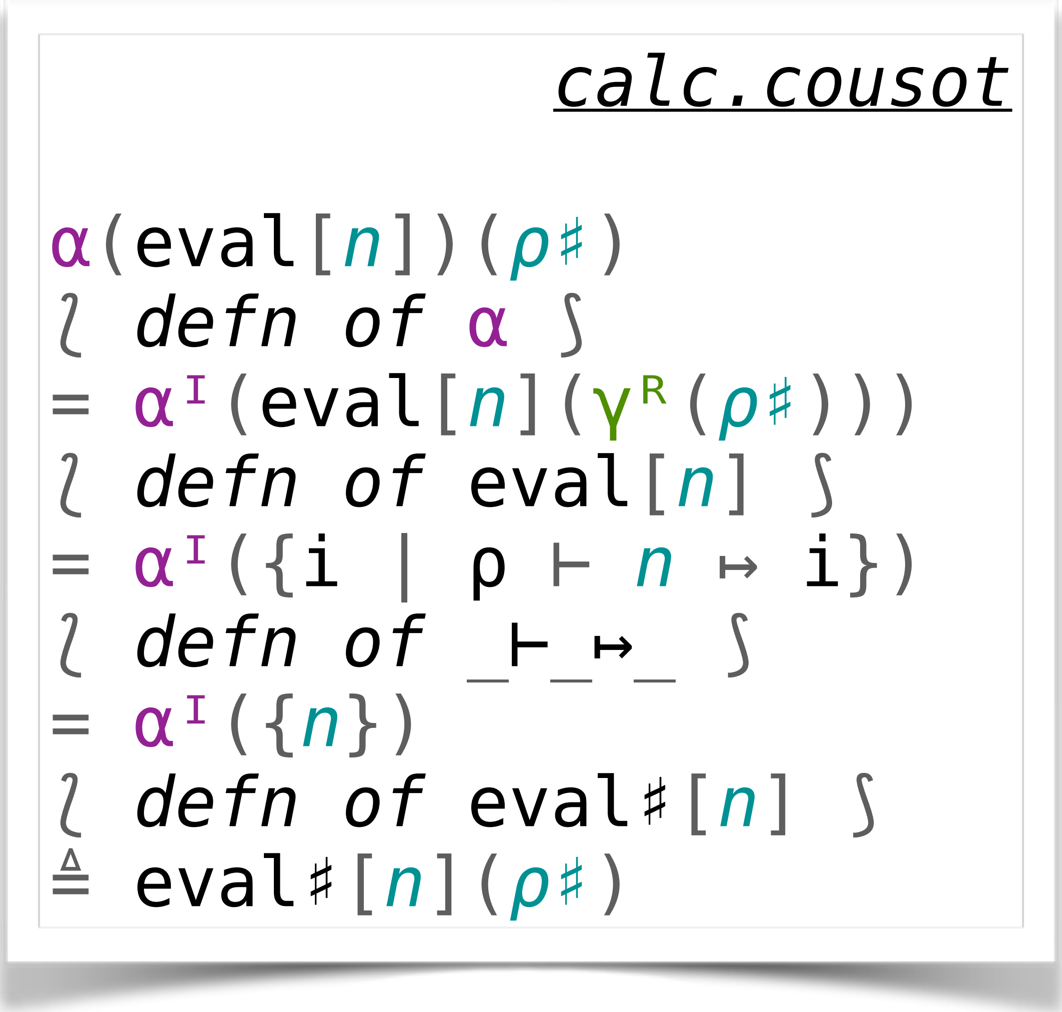


```
⟨η⟩(γ([w,x]) +̂ γ([y,z]))
          =
⟨η⟩({ i + j | i ∈ γ([w,x])
            ∧ j ∈ γ([y,z]) })
          =
⟨η⟩({w+y,…,x+z})
          =
⟨η(w+y)⟩ ⊔ ⋯ ⊔ ⟨η(x+z)⟩
          =
⟨[w+y,x+z]⟩
          ≜
[w,x] +̂ [y,z]
```

Step 1:
Check These **Calculations**
Using a Proof Assistant

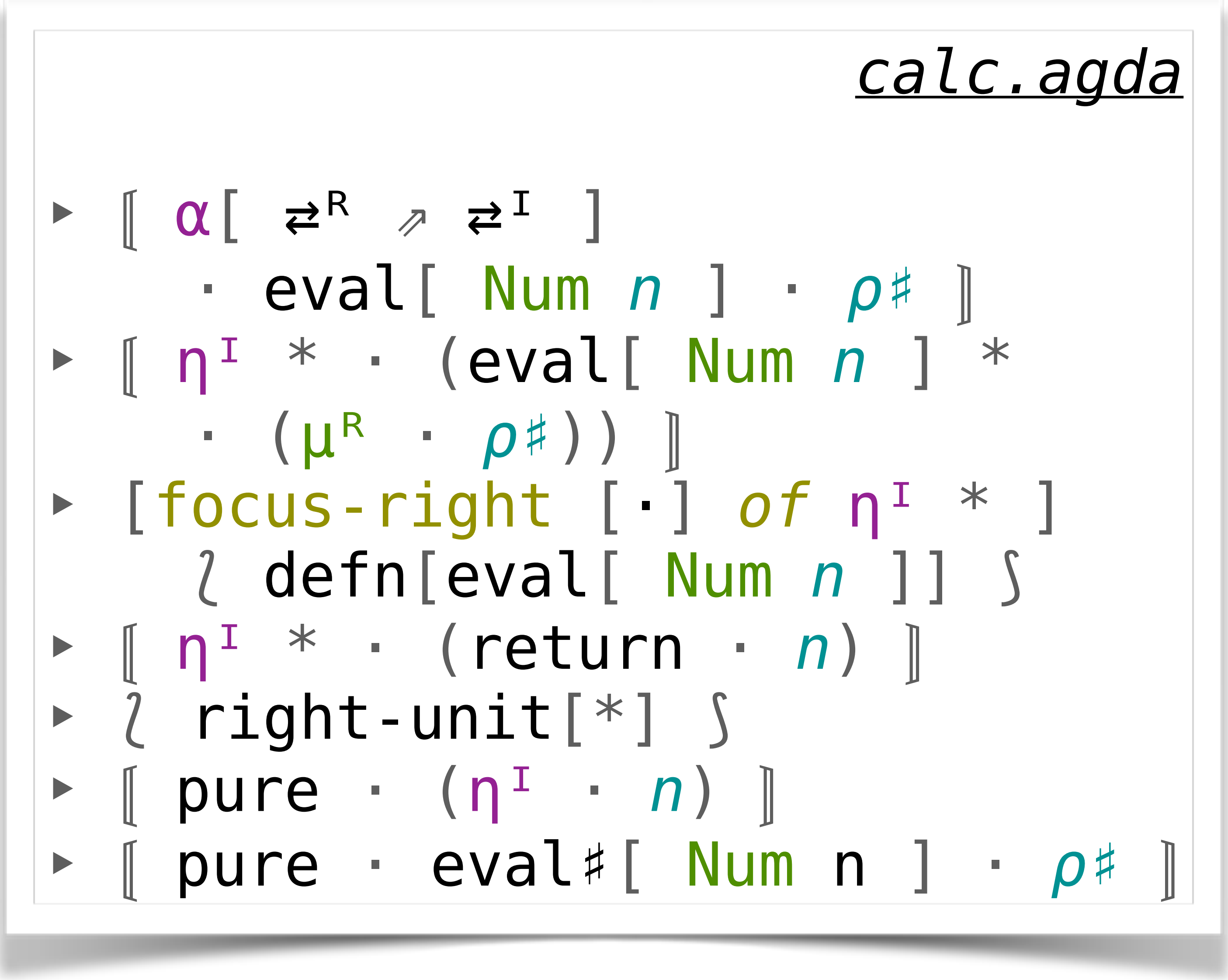
Step 2:
Extract a Certified
Implementation





calc.cousot

```
α(eval[n])(ρ♯)
⟅ defn of α ⟆
= αᴵ(eval[n](γᴿ(ρ♯)))
⟅ defn of eval[n] ⟆
= αᴵ({i | ρ ⊢ n ↦ i})
⟅ defn of _⊢_↦_ ⟆
= αᴵ({n})
⟅ defn of eval♯[n] ⟆
≜ eval♯[n](ρ♯)
```



calc.agda

```
▸ ⟦ α[ ⇄ᴿ ⇗ ⇄ᴵ ]
    · eval[ Num n ] · ρ♯ ⟧
▸ ⟦ ηᴵ * · (eval[ Num n ] *
    · (μᴿ · ρ♯)) ⟧
▸ [focus-right [·] of ηᴵ * ]
    ⟅ defn[eval[ Num n ]] ⟆
▸ ⟦ ηᴵ * · (return · n) ⟧
▸ ⟅ right-unit[*] ⟆
▸ ⟦ pure · (ηᴵ · n) ⟧
▸ ⟦ pure · eval♯[ Num n ] · ρ♯ ⟧
```

Classical GCs

```
A : poset          α : A ↗ B

B : poset          γ : B ↗ A


x ⊑ γ(α(x))  ∧  α(γ(y)) ⊑ y
===========================
   x ⊑ γ(y) ⟺ α(x) ⊑ y
```

Classical GCs

$A : \mathtt{poset}$ $\qquad$ $\alpha : A \nearrow B$

$B : \mathtt{poset}$ $\qquad$ $\gamma : B \nearrow A$

$$\begin{array}{c} \mathrm{id} \sqsubseteq \gamma\circ\alpha \quad\wedge\quad \alpha\circ\gamma \sqsubseteq \mathrm{id} \\ \hline\hline \mathrm{id}(x) \sqsubseteq \gamma(y) \iff \alpha(x) \sqsubseteq \mathrm{id}(y) \end{array}$$

Constructive GCs

$\mathtt{A} : \mathtt{poset} \qquad \eta : \mathtt{A} \nearrow \wp(\mathtt{B})$

$\mathtt{B} : \mathtt{poset} \qquad \mu : \mathtt{B} \nearrow \wp(\mathtt{A})$

$$\dfrac{\mathtt{ret} \sqsubseteq \mu \circledast \eta \quad \wedge \quad \eta \circledast \mu \sqsubseteq \mathtt{ret}}{\mathtt{ret}(n) \subseteq \mu^{*}(r) \iff \eta^{*}(n) \subseteq \mathtt{ret}(r)}$$

Constructive GCs

$$A : \mathrm{poset} \qquad \eta : A \nearrow B$$

$$B : \mathrm{poset} \qquad \mu : B \nearrow \wp(A)$$

$$\begin{array}{c} \mathrm{ret} \sqsubseteq \mu \circledast \langle\eta\rangle \quad \wedge \quad \langle\eta\rangle \circledast \mu \sqsubseteq \mathrm{ret} \\ \hline\hline \mathrm{ret}(n) \subseteq \mu^{*}(r) \iff \langle\eta\rangle^{*}(n) \subseteq \mathrm{ret}(r) \end{array}$$

Constructive GCs

$\mathtt{A} : \mathtt{poset}$

$\mathtt{B} : \mathtt{poset}$

$\eta : \mathtt{A} \nearrow \mathtt{B}$

$\mu : \mathtt{B} \nearrow \wp(\mathtt{A})$

$$\begin{array}{c}
\mathtt{ret} \sqsubseteq \underline{\mu} \circledast \underline{\langle \eta \rangle} \quad \wedge \quad \underline{\langle \eta \rangle} \circledast \underline{\mu} \sqsubseteq \mathtt{ret} \\
\hline\hline
\mathtt{ret}(n) \subseteq \underline{\mu}^{*}(r) \iff \underline{\langle \eta \rangle}^{*}(n) \subseteq \mathtt{ret}(r)
\end{array}$$

Constructive GCs

$\mathtt{A} : \mathtt{poset}$

$\mathtt{B} : \mathtt{poset}$

$\eta : \mathtt{A} \nearrow \mathtt{B}$

$\mu : \mathtt{B} \nearrow \wp(\mathtt{A})$

$$\begin{array}{c} \underline{\mathtt{ret}} \sqsubseteq \mu\underline{\circledast}\langle\eta\rangle \quad\wedge\quad \langle\eta\rangle\underline{\circledast}\mu \sqsubseteq \underline{\mathtt{ret}} \\ \hline\hline \underline{\mathtt{ret}}(n) \subseteq \mu^{\underline{*}}(r) \iff \langle\eta\rangle^{\underline{*}}(n) \subseteq \underline{\mathtt{ret}}(r) \end{array}$$

Classical GCs

=

adjunction in

category of posets

(adjoints are mono. functions)

Constructive GCs

=

biadjunction in

category of posets enriched over $\wp$-Kleisli

(adjoints are mono. $\wp$-monadic functions)

Constructive Galois Connections

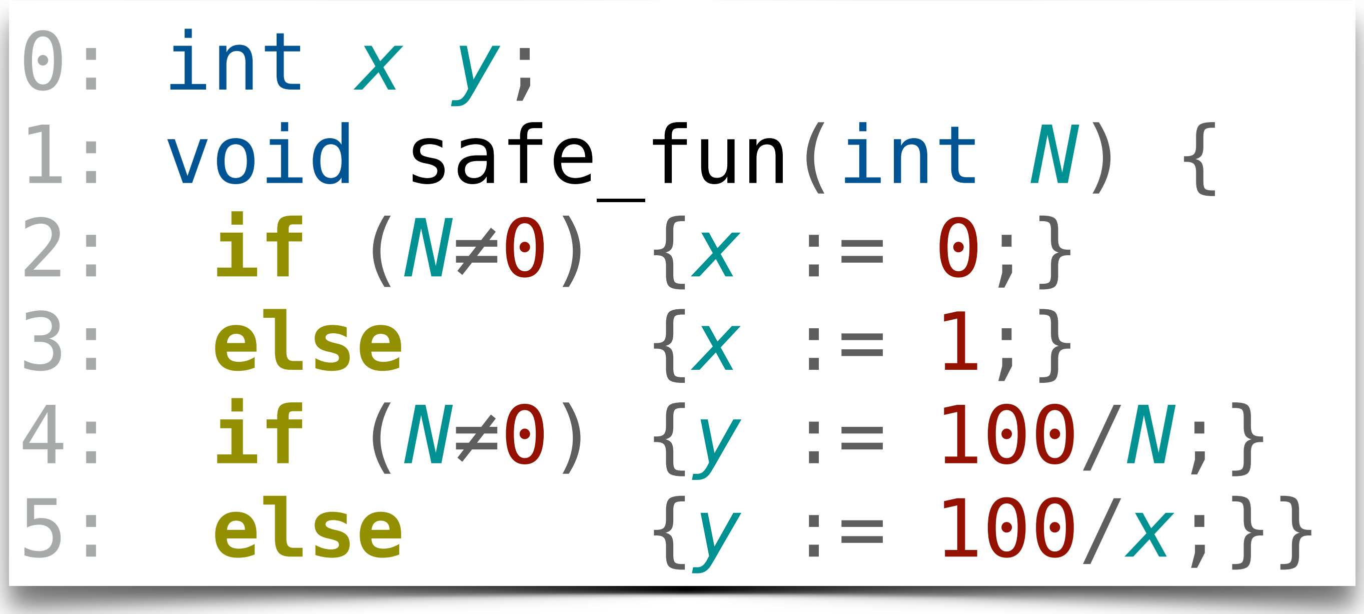
- ✓ First theory to support both calculation and extraction
- ✓ Soundness and completeness w.r.t. classical GCs
- ✓ Two case studies: calculational AI and gradual typing
- ✗ Only (constr.) equivalent to subset of classical GCs
- ✗ Same limitations as classical GCs ($\nexists\alpha$ for some γ)

Constructive
Galois
Connections

Galois
Transformers

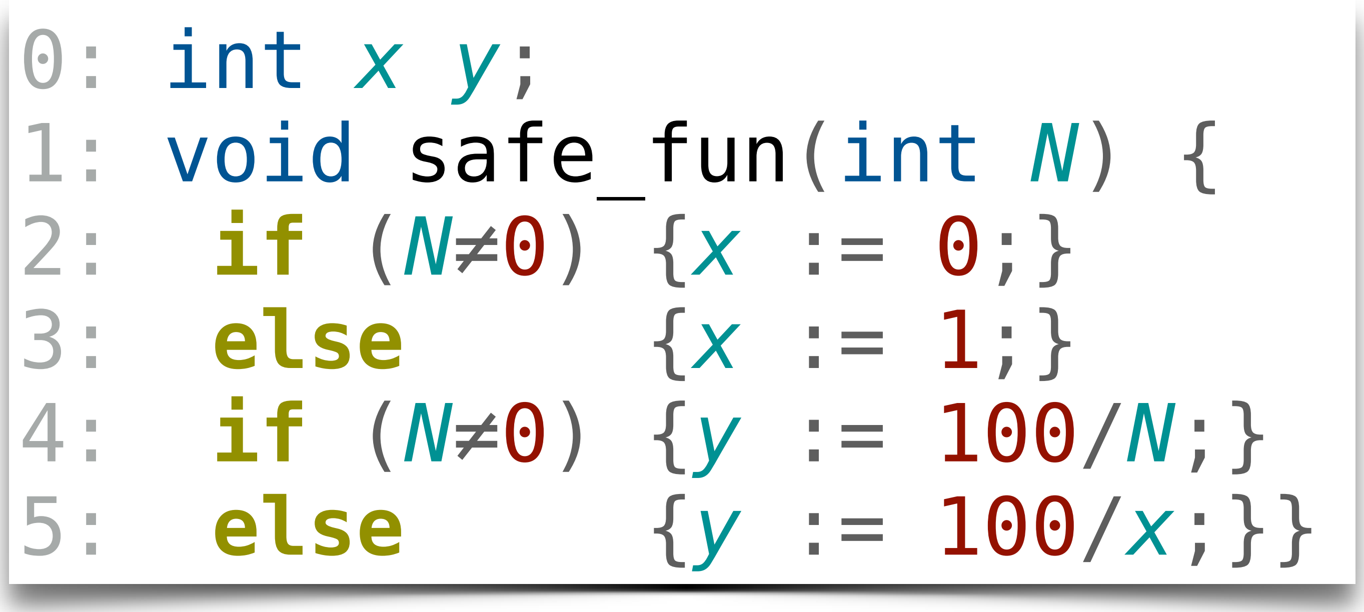
Abstracting
Definitional
Interpreters

Galois Transformers



```
int x y;
void safe_fun(int N) {
 if (N≠0) {x := 0;}
 else     {x := 1;}
 if (N≠0) {y := 100/N;}
 else     {y := 100/x;}}
```

Galois Transformers



```
int x y;
void safe_fun(int N) {
 if (N≠0) {x := 0;}
 else     {x := 1;}
 if (N≠0) {y := 100/N;}
 else     {y := 100/x;}}
```

Flow-insensitive

$$\texttt{results} : \texttt{var} \mapsto \wp(\{\texttt{-},\texttt{0},\texttt{+}\})$$

Galois Transformers

```
int x y;
void safe_fun(int N) {
 if (N≠0) {x := 0;}
 else     {x := 1;}
 if (N≠0) {y := 100/N;}
 else     {y := 100/x;}}
```

```
N ∈ {-,0,+}
x ∈ {0,+}
y ∈ {-,0,+}

UNSAFE: {100/N}
UNSAFE: {100/x}
```

Flow-insensitive

$$\mathtt{results} : \mathtt{var} \mapsto \wp(\{-,0,+\})$$

Galois Transformers

```
0: int x y;
1: void safe_fun(int N) {
2:  if (N≠0) {x := 0;}
3:  else      {x := 1;}
4:  if (N≠0) {y := 100/N;}
5:  else      {y := 100/x;}}
```

```
4:     x ∈ {0,+}
4.T: N ∈ {-,+}
5.F: x ∈ {0,+}

N,y ∈ {-,0,+}

UNSAFE: {100/x}
```

Flow-sensitive

`results : loc ↦ (var ↦ ℘({-,0,+}))`

Galois Transformers

```
0: int x y;
1: void safe_fun(int N) {
2:  if (N≠0) {x := 0;}
3:  else      {x := 1;}
4:  if (N≠0) {y := 100/N;}
5:  else      {y := 100/x;}}
```

```
4: N∈{-,+},x∈{0}
4: N∈{0}  ,x∈{+}

N∈{-,+},y∈{-,0,+}
N∈{0}  ,y∈{0,+}

SAFE
```

Path-sensitive

$$\mathtt{results} : \mathtt{loc} \mapsto \underline{\wp}(\mathtt{var} \mapsto \wp(\{-,0,+\}))$$

Precision ↯ *Performance*

Insight

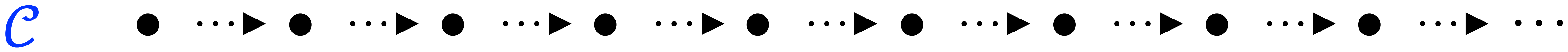
$$\texttt{results} : \texttt{var} \mapsto \wp(\{\texttt{-},\texttt{0},\texttt{+}\})$$

$$\texttt{results} : \texttt{loc} \mapsto (\texttt{var} \mapsto \wp(\{\texttt{-},\texttt{0},\texttt{+}\}))$$

$$\texttt{results} : \texttt{loc} \mapsto \wp(\texttt{var} \mapsto \wp(\{\texttt{-},\texttt{0},\texttt{+}\}))$$

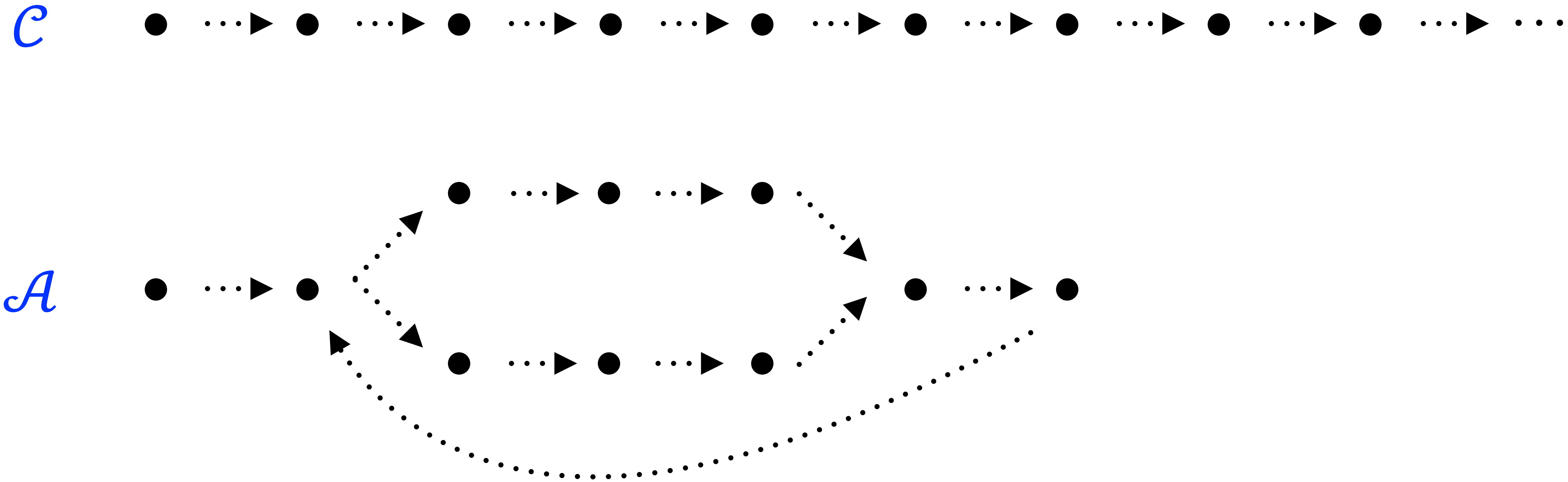
Single Analyzer + **Monadic Abstraction** + **FI, FS and PS Monads**

$$\mathcal{C} : \texttt{loc} \times \texttt{store} \to \texttt{loc} \times \texttt{store}$$



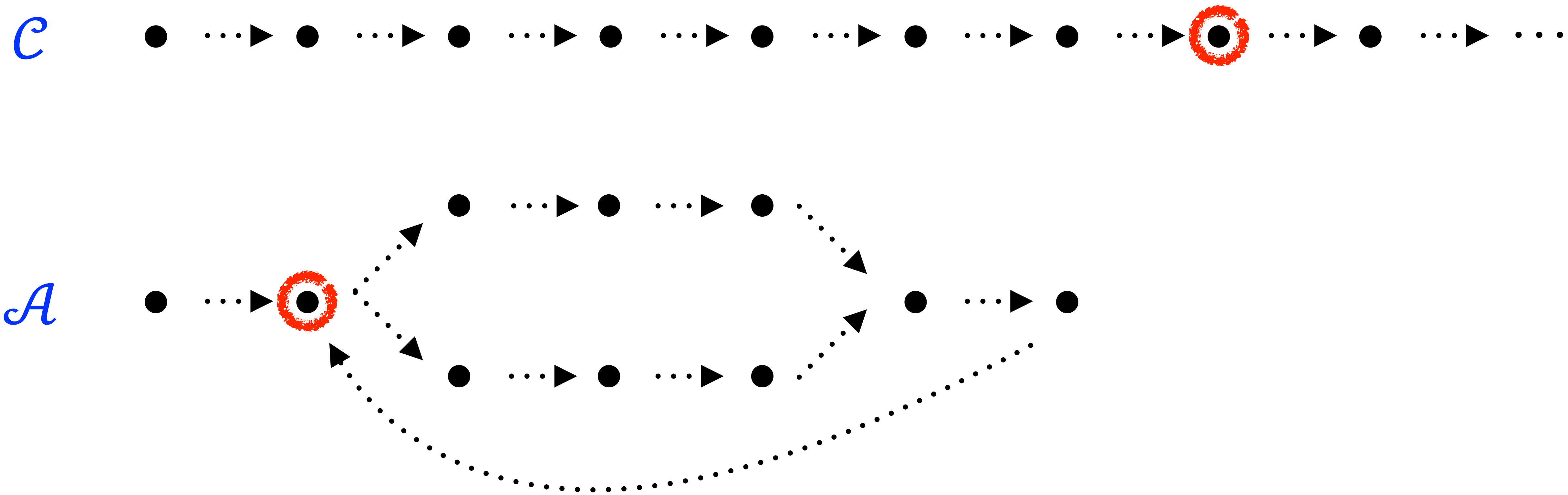
$$\mathcal{C} : \texttt{loc} \times \texttt{store} \to \texttt{loc} \times \texttt{store}$$

$$\mathcal{A} : \texttt{loc} \times \texttt{store}\sharp \to \wp(\texttt{loc} \times \texttt{store}\sharp)$$



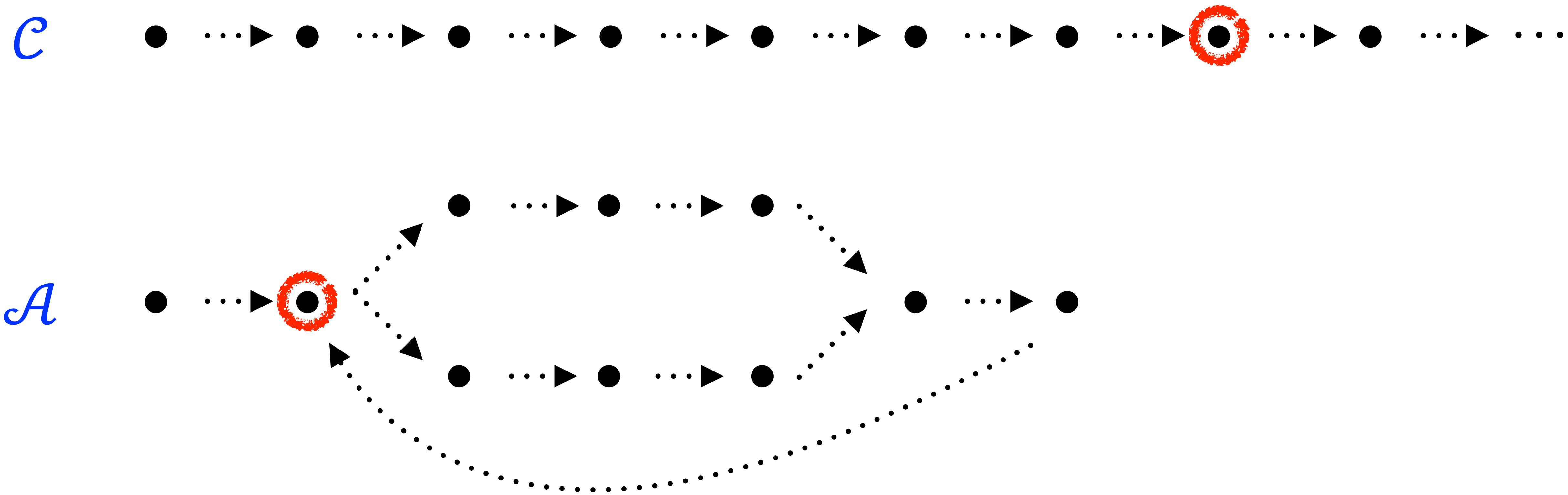
$$\mathcal{C} : \mathtt{loc} \times \mathtt{store} \to \mathtt{loc} \times \mathtt{store}$$

$$\mathcal{A} : \mathtt{loc} \times \mathtt{store}\sharp \to \wp(\mathtt{loc} \times \mathtt{store}\sharp)$$



$\mathcal{C}$: loc × store → loc × store

$\mathcal{A}$: loc × store♯ → $\wp$(loc × store♯)



analyzer ≔ **lfp** $X.\ X \cup \mathcal{A}^{*}(X) \cup \{\langle l_0, \bot \rangle\}$

$$\mathcal{A} : \underbrace{\wp(\mathtt{loc} \times \mathtt{store}\sharp)}_{\Sigma} \rightarrow \underbrace{\wp(\mathtt{loc} \times \mathtt{store}\sharp)}_{\Sigma}$$

$$\mathcal{A} : \underbrace{\wp(\mathtt{loc} \times \mathtt{store}^\sharp)}_{\Sigma} \rightarrow \underbrace{\wp(\mathtt{loc} \times \mathtt{store}^\sharp)}_{\Sigma}$$

Path Sensitive	$\Sigma \coloneqq \mathtt{loc} \mapsto \wp(\mathtt{store}^\sharp)$ $\approx \wp(\mathtt{loc} \times \mathtt{store}^\sharp)$
Flow Sensitive	$\Sigma \coloneqq \mathtt{loc} \mapsto \mathtt{store}^\sharp$
Flow Insensitive	$\Sigma \coloneqq \wp(\mathtt{loc}) \times \mathtt{store}^\sharp$

$$\mathcal{M} : \texttt{loc} \rightarrow m(\texttt{loc})$$

$$\mathcal{M} : \mathtt{loc} \to m(\mathtt{loc})$$

Path Sensitive	$m(A) \coloneqq \mathtt{store}\sharp \to A \mapsto \wp(\mathtt{store}\sharp)$
Flow Sensitive	$m(A) \coloneqq \mathtt{store}\sharp \to A \mapsto \mathtt{store}\sharp$
Flow Insensitive	$m(A) \coloneqq \mathtt{store}\sharp \to \wp(A) \times \mathtt{store}\sharp$

$$\mathcal{M} : \texttt{loc} \rightarrow m(\texttt{loc})$$

Path Sensitive $\quad m(A) \coloneqq (S[\texttt{store}\sharp] \circ ND)(\texttt{ID})(A)$

Flow Sensitive $\quad m(A) \coloneqq FS[\texttt{store}\sharp](\texttt{ID})(A)$

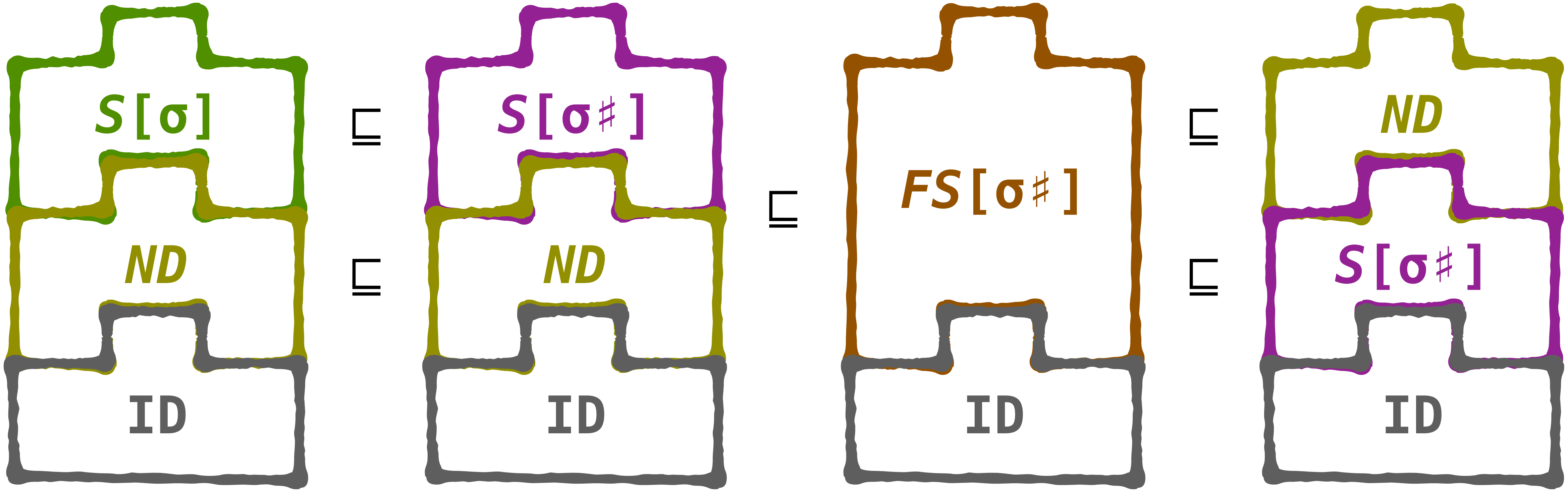
Flow Insensitive $\quad m(A) \coloneqq (ND \circ S[\texttt{store}\sharp])(\texttt{ID})(A)$

Collecting Semantics

Path Sensitive

Flow Sensitive

Flow Insensitive



$S[\sigma]$

ND

ID

$\sqsubseteq$

$\sqsubseteq$

$S[\sigma\sharp]$

ND

ID

$\sqsubseteq$

$FS[\sigma\sharp]$

ID

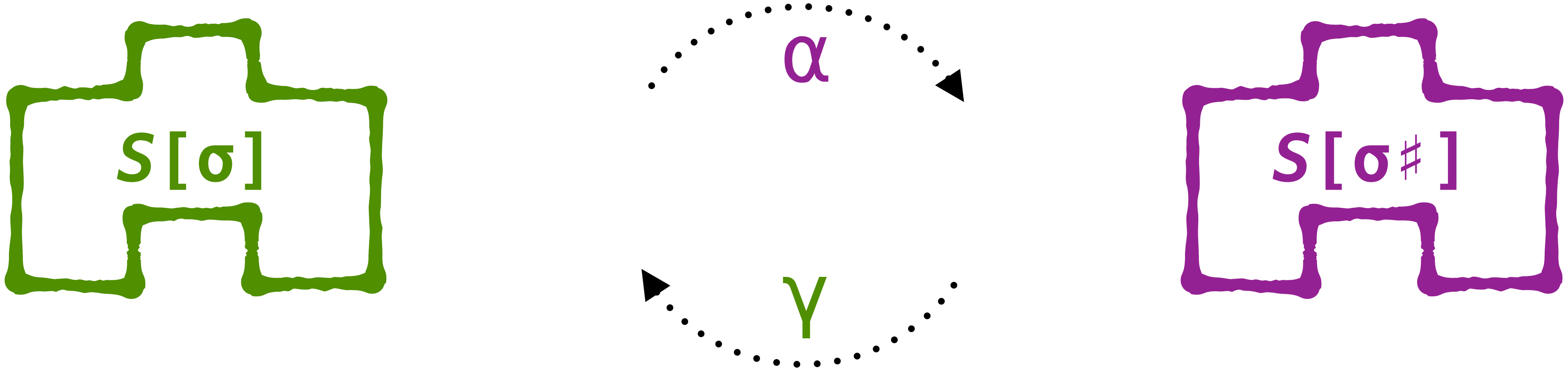
$\sqsubseteq$

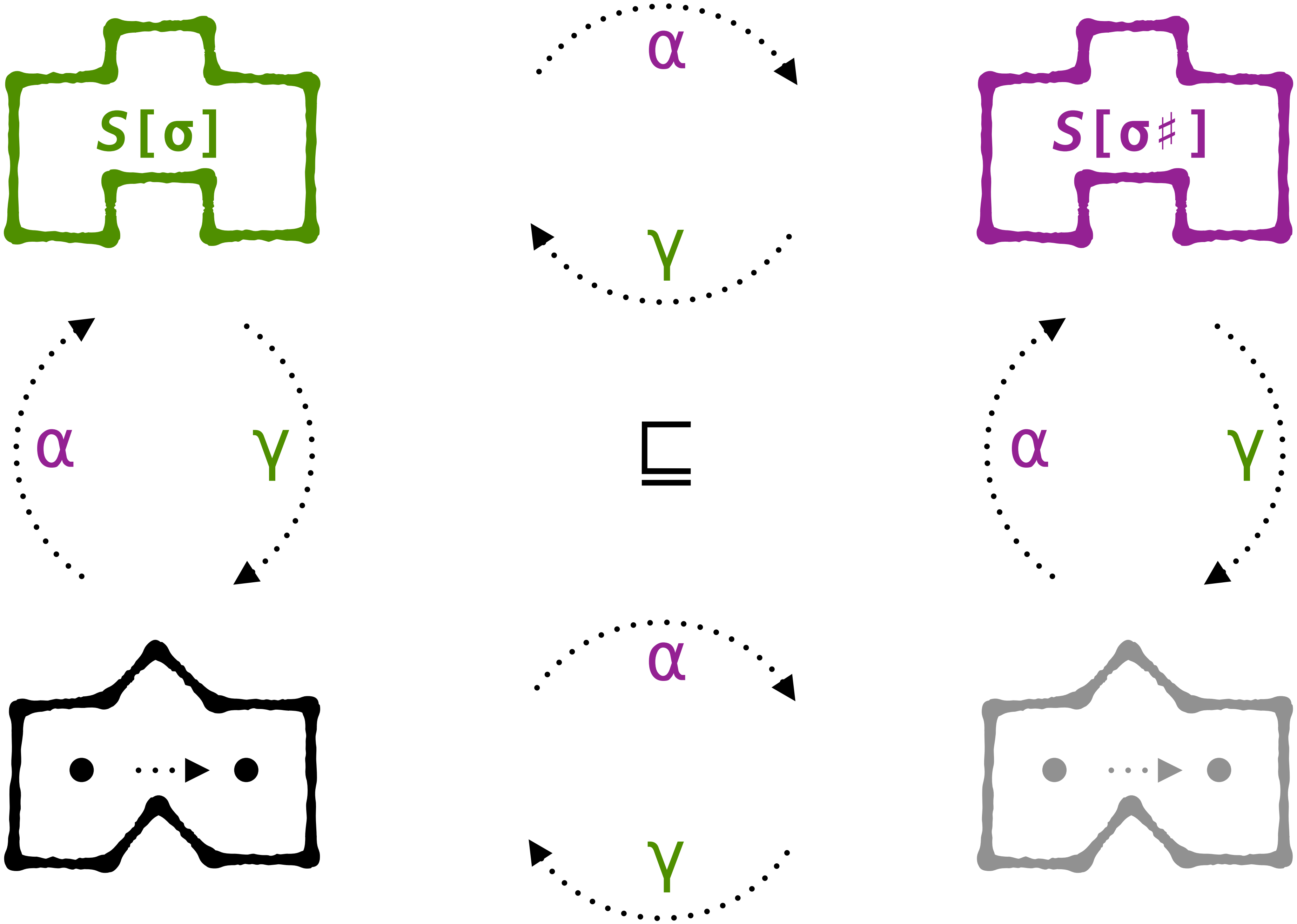
$\sqsubseteq$

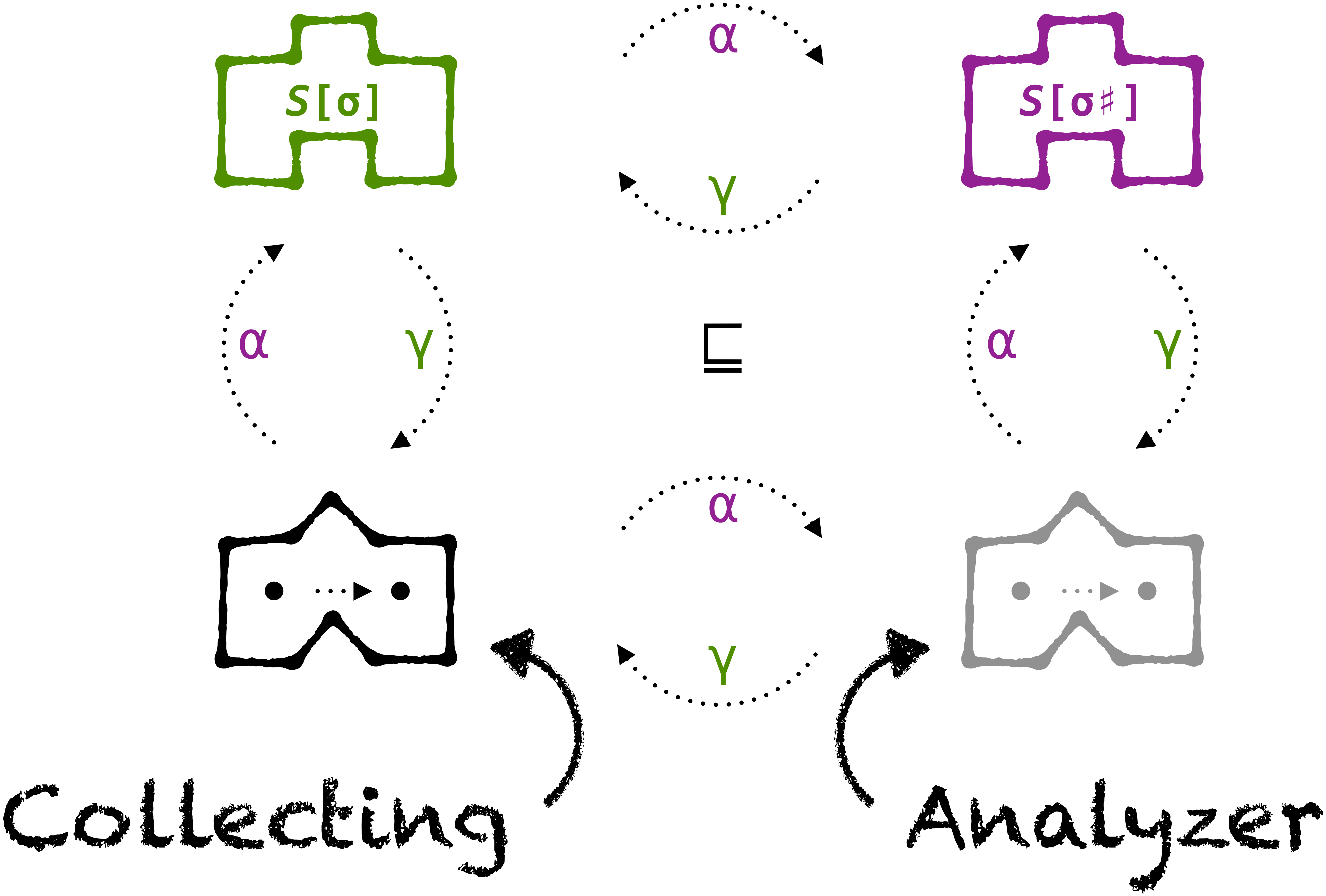
ND

$S[\sigma\sharp]$

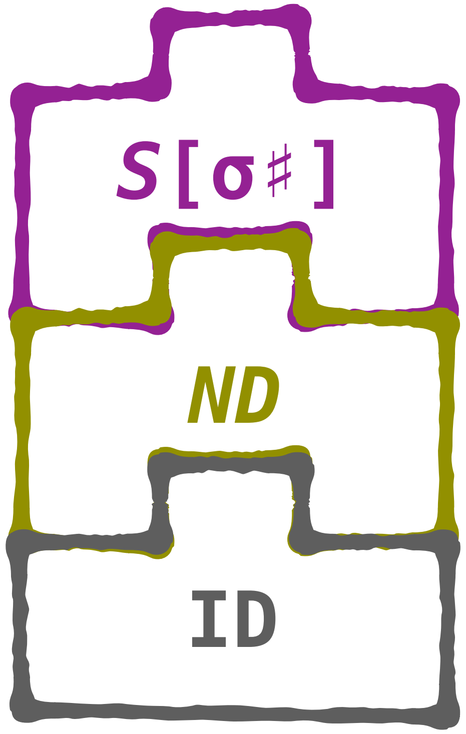
ID

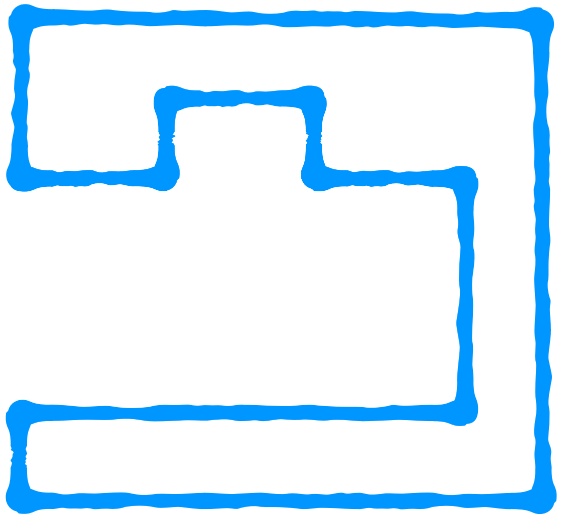

S[σ]
α
γ
S[σ♯]


S[σ]
α
γ
S[σ♯]
α
γ
α
γ
⊑
α
γ
α
γ


S[σ]
α
γ
S[σ♯]
α
γ
α
γ
⊑
α
γ
Collecting
Analyzer

Analyzer = Monad Stack + Monadic Interpreter





One Monadic Interpreter

Must Be Monotonic

Must Recover Collecting Semantics

Galois Transformers

✓ Flow sensitive and path sensitive precision

✓ Compositional end-to-end correctness proofs

✓ Implemented in Haskell and available on Github

✗ Not whole story for path-sensitive refinement

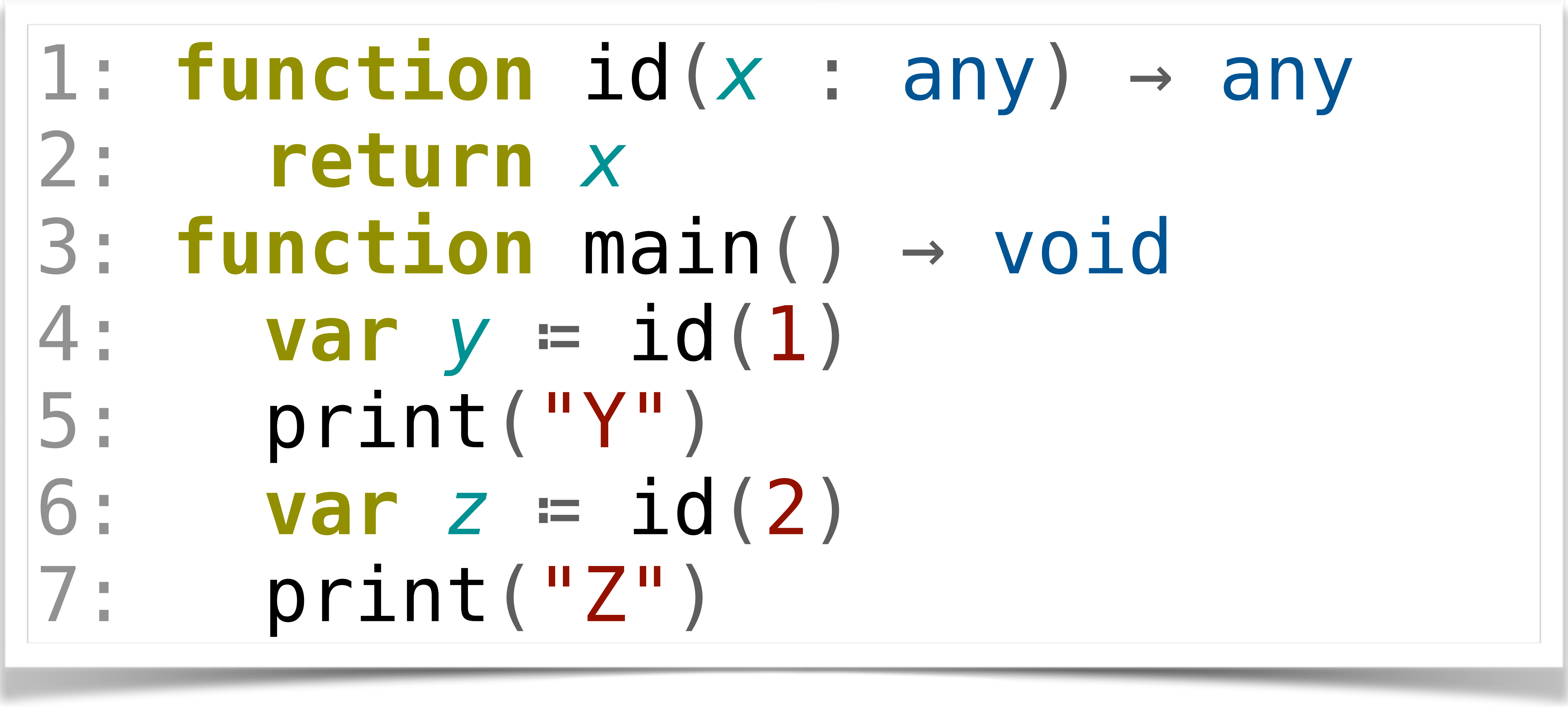
✗ Naive fixpoint strategies

Constructive
Galois
Connections

Galois
Tranformers

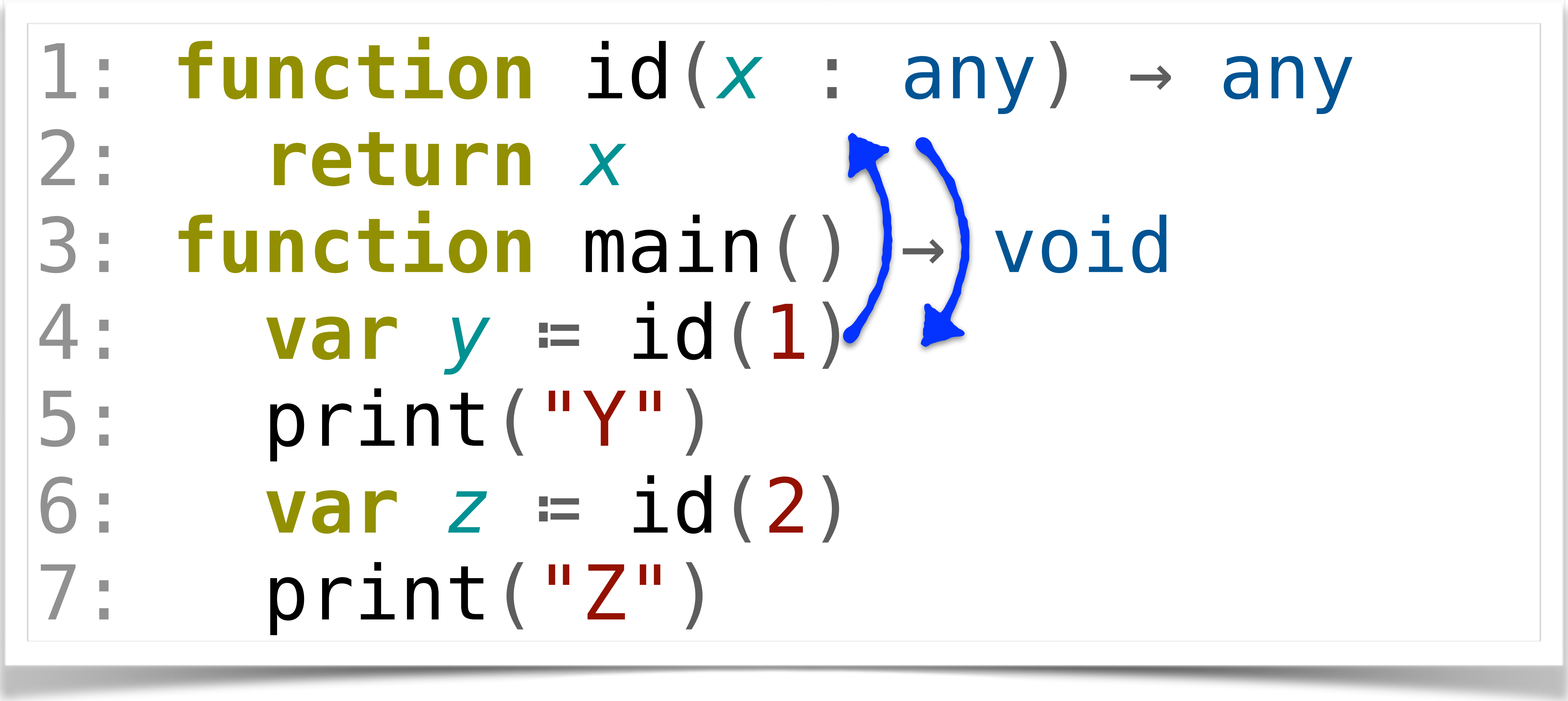
Abstracting
Definitional
Interpreters

Abstracting Definitional Interpreters



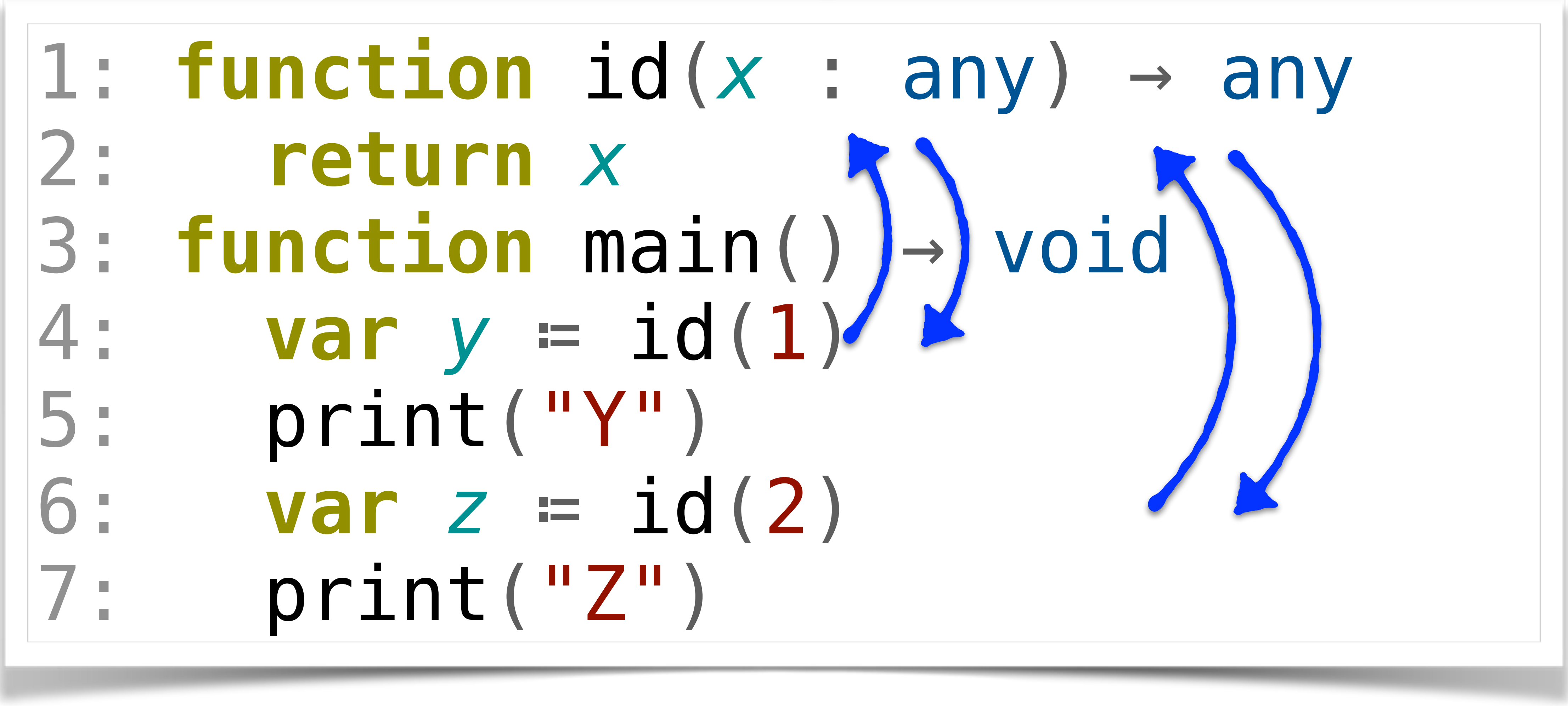
```
function id(x : any) → any
  return x
function main() → void
  var y ≔ id(1)
  print("Y")
  var z ≔ id(2)
  print("Z")
```

Abstracting Definitional Interpreters



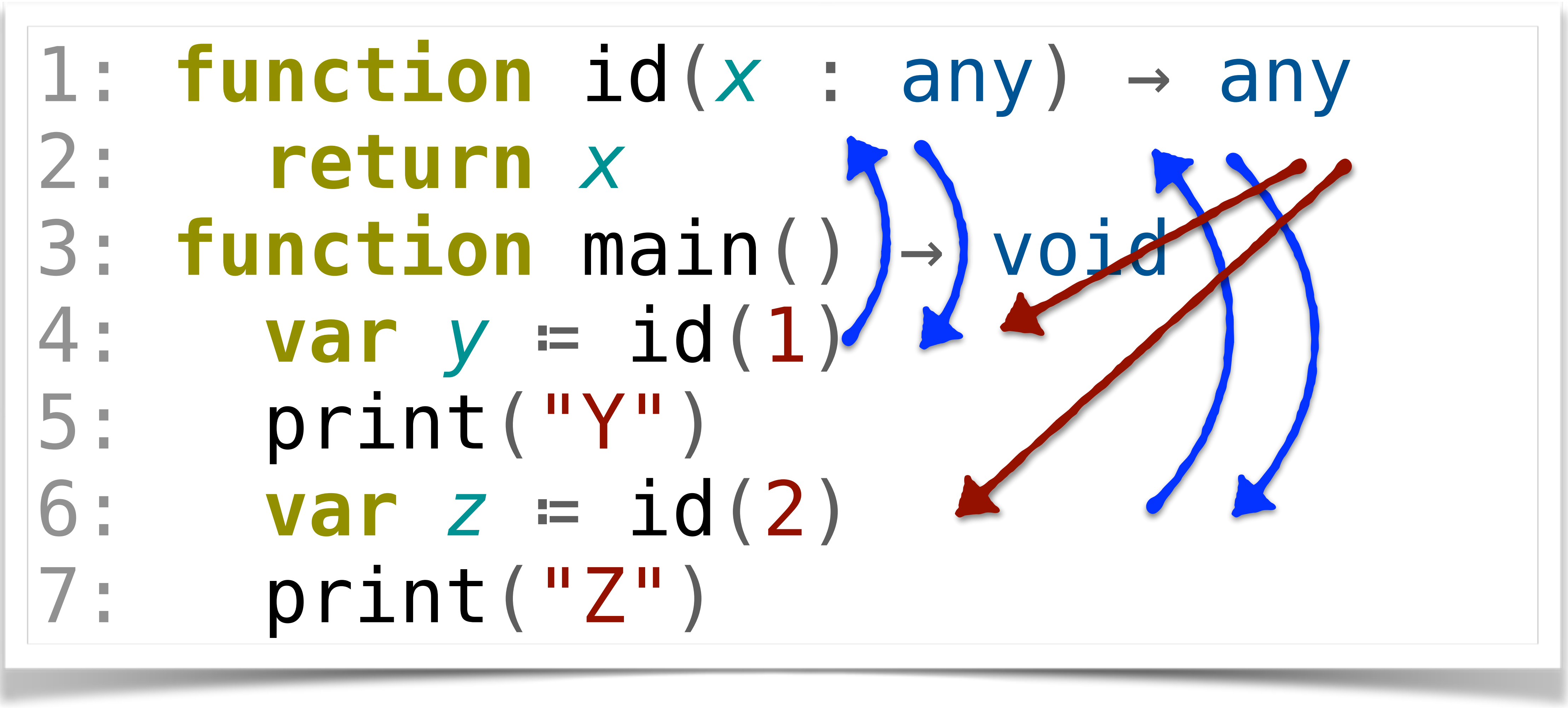
```
function id(x : any) → any
  return x
function main() → void
  var y ≔ id(1)
  print("Y")
  var z ≔ id(2)
  print("Z")
```

Abstracting Definitional Interpreters



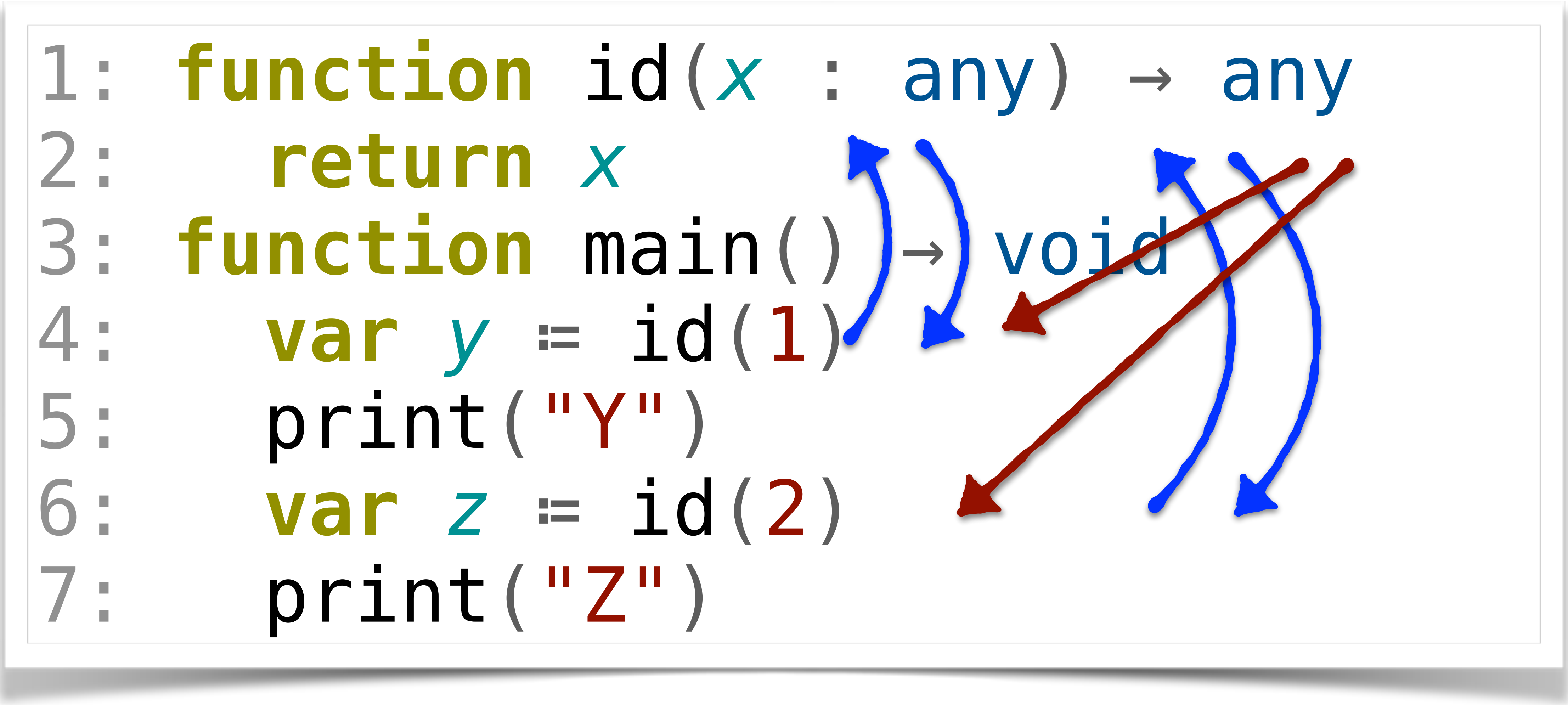
```
function id(x : any) → any
  return x
function main() → void
  var y ≔ id(1)
  print("Y")
  var z ≔ id(2)
  print("Z")
```

Abstracting Definitional Interpreters



```
function id(x : any) → any
  return x
function main() → void
  var y ≔ id(1)
  print("Y")
  var z ≔ id(2)
  print("Z")
```

Abstracting Definitional Interpreters



```
function id(x : any) → any
  return x
function main() → void
  var y ≔ id(1)
  print("Y")
  var z ≔ id(2)
  print("Z")
```

```
> Z
> YYYYYYZ
```

Pushdown Precision

Reps *et al* 1995	Doesn’t support HO control
Earl Diss 2012	Dyck State Graphs
Vardoulakis Diss 2012	“Big”CFA
Johnson and Van Horn 2014	Instrumented AAM
Gilray *et al* 2016	Instrumented AAM

Definitional Interpreters

- Modeled features vs inherited features
- (e.g., Reynolds' inherited CBV and CBN)
- Things often modeled in Abstract Interpreters
  - Control (continuations)
  - Fixpoints

Definitional Interpreters

- Modeled features vs inherited features
- (e.g., Reynolds' inherited CBV and CBN)
- Things often modeled in Abstract Interpreters
    - Control (continuations)
    - Fixpoints

Idea:
Inherit from
metalanguage

$$\mathcal{E}[\![\cdot]\!] : \texttt{exp} \rightarrow \texttt{env} \times \texttt{store} \rightarrow (\texttt{val} \times \texttt{env} \times \texttt{store})$$

$$\mathcal{E}\llbracket \cdot \rrbracket : \mathtt{exp} \to \mathtt{env} \times \mathtt{store} \to (\mathtt{val} \times \mathtt{env} \times \mathtt{store})$$

…

$$\begin{aligned}
&\mathcal{E}\llbracket \mathbf{if}(e_1)\{e_2\}\{e_3\} \rrbracket(\rho,\sigma) \coloneqq \mathbf{match}\ \mathcal{E}\llbracket e_1 \rrbracket(\rho,\sigma) \\
&\quad |\ \langle \mathtt{true}\ ,\sigma' \rangle \Rightarrow \mathcal{E}\llbracket e_2 \rrbracket(\rho,\sigma') \\
&\quad |\ \langle \mathtt{false},\sigma' \rangle \Rightarrow \mathcal{E}\llbracket e_3 \rrbracket(\rho,\sigma')
\end{aligned}$$

…

```
ℰ⟦·⟧ : exp → env × store → (val × env × store)
```

```
…
ℰ⟦if(e₁){e₂}{e₃}⟧(ρ,σ) ≔ match ℰ⟦e₁⟧(ρ,σ)
  | ⟨true ,σ′⟩ ⇒ ℰ⟦e₂⟧(ρ,σ′)
  | ⟨false,σ′⟩ ⇒ ℰ⟦e₃⟧(ρ,σ′)
…
```

No explicit model for control (continuations).
It’s inherited from the metalanguage.

$$\mathcal{E}[\![\cdot]\!] : \texttt{exp} \to m(\texttt{val})$$

Step 1
Monadic Interpreter

$$\mathcal{E}[\![\cdot]\!] : \mathtt{exp} \rightarrow m(\mathtt{val})$$

```
…
$\mathcal{E}[\![$if($e_1$){$e_2$}{$e_3$}$]\!]$ ≔ do
   $v$ ← $\mathcal{E}[\![e_1]\!]$
   match $v$ | true  ⇒ $\mathcal{E}[\![e_2]\!]$
             | false ⇒ $\mathcal{E}[\![e_3]\!]$
…
```

Step 1
Monadic Interpreter

$$\mathcal{E}[\![\cdot]\!] : \mathtt{exp} \rightarrow (\mathtt{exp} \rightarrow m(\mathtt{val})) \rightarrow m(\mathtt{val})$$

Step 2
Unfixed Recursion

$$\mathcal{E}[\![\cdot]\!] : \mathtt{exp} \to (\mathtt{exp} \to m(\mathtt{val})) \to m(\mathtt{val})$$

```
…
$\mathcal{E}[\![\mathbf{if}(e_1)\{e_2\}\{e_3\}]\!](\mathcal{E}')$ ≔ do
    $v \leftarrow \mathcal{E}'[\![e_1]\!]$
    match $v$ | true  ⇒ $\mathcal{E}'[\![e_2]\!]$
              | false ⇒ $\mathcal{E}'[\![e_3]\!]$
…
```

Step 2
Unfixed Recursion

$$\mathcal{E}[\![\cdot]\!] : \mathtt{exp} \rightarrow (\mathtt{exp} \rightarrow m\sharp(\mathtt{val})) \rightarrow m\sharp(\mathtt{val})$$

```
…
$\mathcal{E}[\![$if($e_1$){$e_2$}{$e_3$}$]\!]$($\mathcal{E}'$) ≔ do
  $v \leftarrow \mathcal{E}'[\![e_1]\!]$
  match $v$ | true  ⇒ $\mathcal{E}'[\![e_2]\!]$
            | false ⇒ $\mathcal{E}'[\![e_3]\!]$
…
```

Step 3

Abstract Monad

$$\mathcal{E}[\![\cdot]\!] : \texttt{exp} \rightarrow (\texttt{exp} \rightarrow m\sharp(\texttt{val})) \rightarrow m\sharp(\texttt{val})$$

$$Y(\lambda \mathcal{E}'.\lambda e.\mathcal{E}[\![e]\!](\mathcal{E}'))$$

Abstract Evaluator
(Doesn’t Terminate)

$$\mathcal{E}[\![\cdot]\!] : \mathtt{exp} \to (\mathtt{exp} \to m\sharp(\mathtt{val})) \to m\sharp(\mathtt{val})$$

$$\mathtt{Y}(\lambda\, \mathcal{E}'.\lambda\, e.\mathcal{E}[\![e]\!](\mathcal{E}'))$$

Abstract Evaluator
(Doesn't Terminate)

$$\underbrace{\mathtt{CY}(\lambda\, \mathcal{E}'.\lambda\, e.\mathcal{E}[\![e]\!](\mathcal{E}'))}_{\text{Pushdown Precision}}$$

Caching Evaluator
(Terminates)

Formalism

$\rho, \tau \vdash e, \sigma \Downarrow v, \sigma$ Evaluation

$\rho, \tau \vdash e, \sigma \Uparrow \langle e, \rho, \tau, \sigma \rangle$ Reachability

Formalism

$\rho,\tau \vdash e,\sigma \Downarrow v,\sigma$ Evaluation

$\rho,\tau \vdash e,\sigma \Uparrow \langle e,\rho,\tau,\sigma\rangle$ Reachability

$$
\begin{aligned}
\llbracket e\rrbracket(\rho,\tau,\sigma) \coloneqq\ & \\
\{\langle v,\sigma''\rangle \mid\ & \rho\ ,\tau\ \vdash e\ ,\sigma\ \Uparrow \langle e',\rho',\tau',\sigma'\rangle \\
\wedge\ & \rho',\tau' \vdash e',\sigma' \Downarrow \langle v,\sigma''\rangle\}
\end{aligned}
$$

Definitional Abstract Interpreters

- ✓ Compositional program analyzers
- ✓ Formalized w.r.t. big-step reachability semantics
- ✓ Pushdown precision inherited from metalanguage
- ✓ Implemented in Racket and available on Github
- ✗ Naive caching algorithm (could be improved)
- ✗ Monadic, open-recursive interpreters

	Usable	Trustworthy
Program Analysis	✓	✗
Mechanized Verification	✗	✓
MVPA	✓	✓

My Research

Thesis

Constructing mechanically verified program analyzers via calculation and composition is *feasible* using constructive Galois connections and modular abstract interpreters.

Constructive Galois Connections

Mechanization
+
Calculation

Galois Tranformers

Compositional
Path-sens.
+
Flow-sens.

Abstracting Definitional Interpreters

Compositional
Interpreters
+
Pushdown
Precision