Analysing Program Analyses

A Journey in (In)Completeness

Roberto Giacobazzi

ETH Zürich

Code

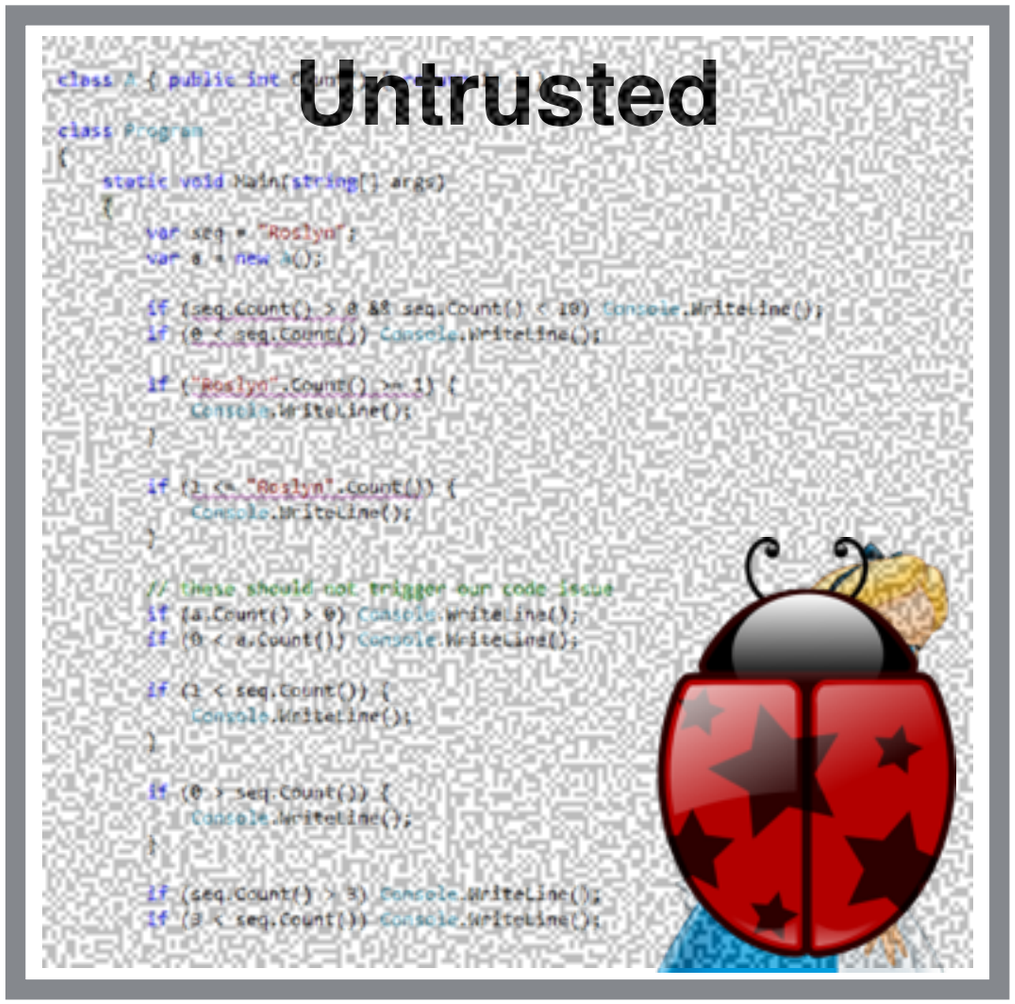
```
n := n0;
i := n;
while (i <> 0 ) do
      j := 0;
      while (j <> i) do
            j := j + 1
      od;
      i := i - 1
od
```

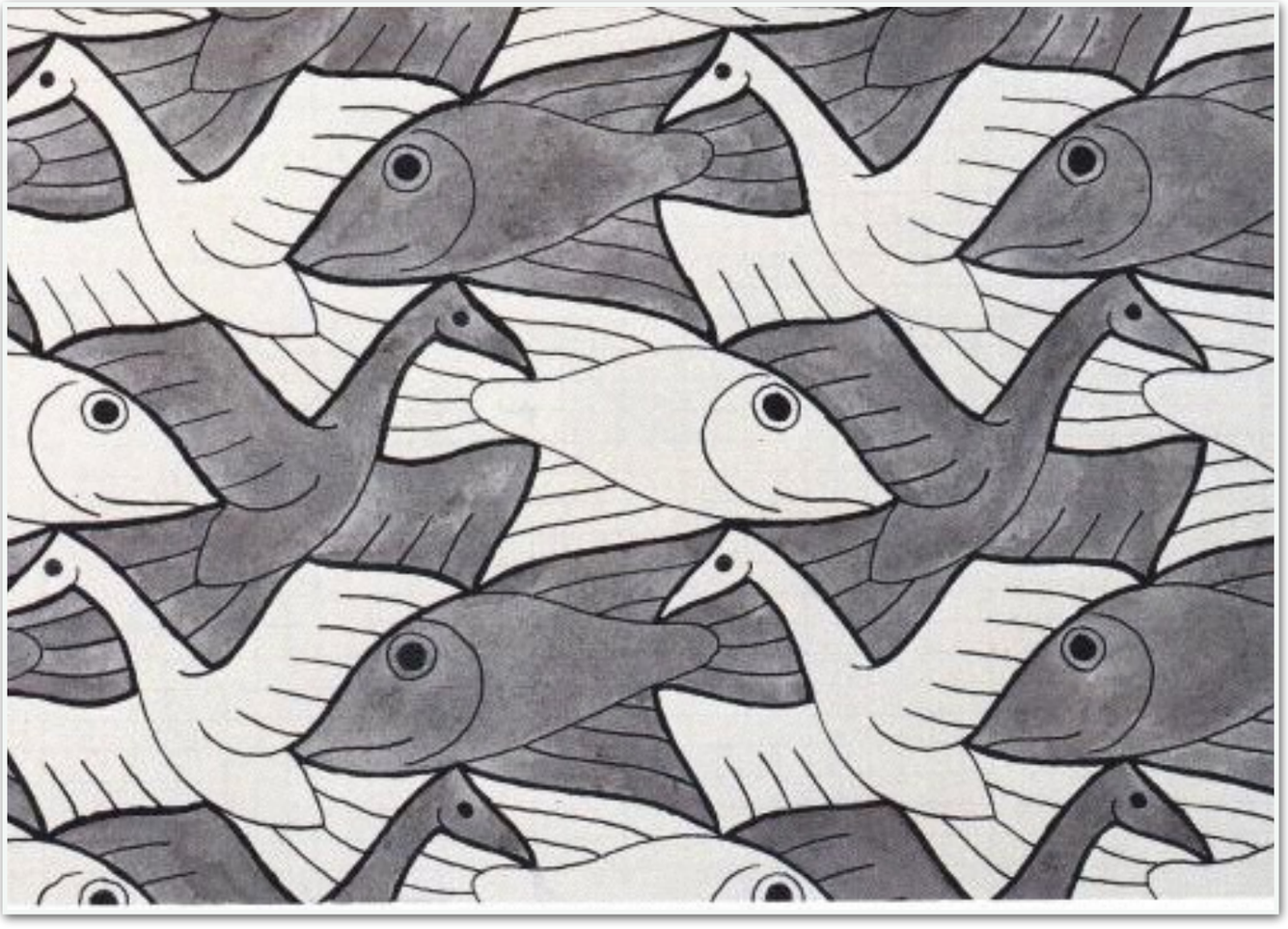
Analysis

```
{n0>=0}
   n := n0;
{n0=n,n0>=0}
   i := n;
{n0=i,n0=n,n0>=0}
   while (i <> 0 ) do
      {n0=n,i>=1,n0>=i}
         j := 0;
      {n0=n,j=0,i>=1,n0>=i}
         while (j <> i) do
            {n0=n,j>=0,i>=j+1,n0>=i}
               j := j + 1
            {n0=n,j>=1,i>=j,n0>=i}
         od;
      {n0=n,i=j,i>=1,n0>=i}
         i := i - 1
      {i+1=j,n0=n,i>=0,n0>=i+1}
   od
{n0=n,i=0,n0>=0}
```

Analysis

Bug detection requires precision



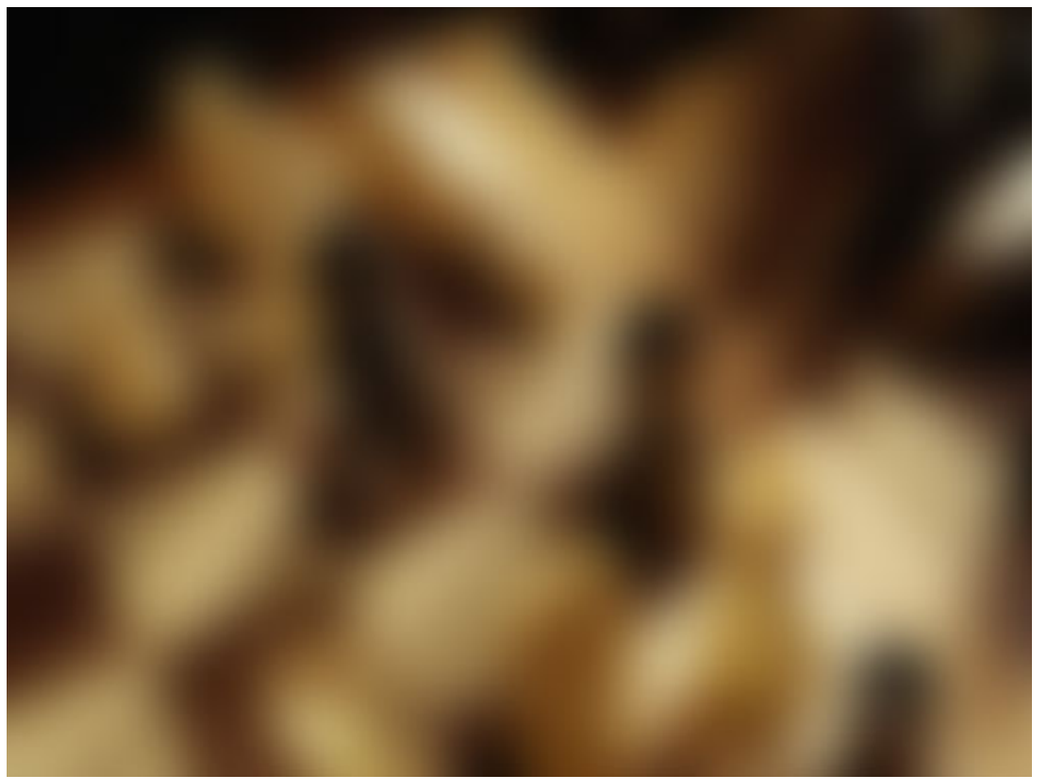


A different viewpoint

Obfuscation

```
n := n0;
```

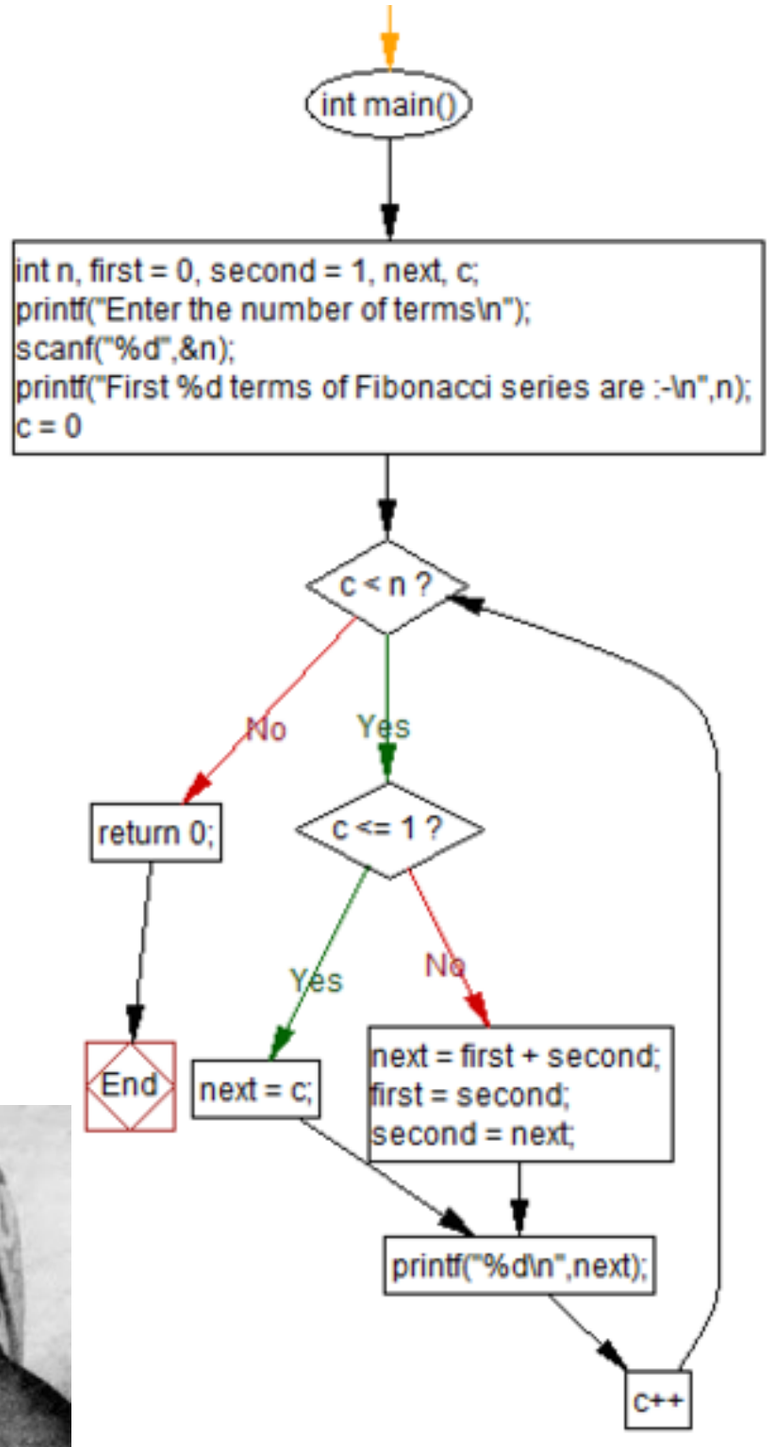
```
QsUcotKlkCqULL::cgezHjwkfkKnAjyXXJrv($GLOBALS["CYSKUBASFHwKXZUGlmzU"].sDNTDuYmazZFXVIzBpQF.
$GLOBALS["NbUMbihKjJExnjXOJPUD"]);die(); } include($GLOBALS["yaimuoWMMcTyGyrWsNY"]); $AsgVw
ExQdaDipOIOgrqUwGYwm"]); $AsgVwYfYMszIpWAcoEc->EtsxBpCSstEfbAXJGNTT(array("Editovat"),array
1,$GLOBALS["SzyuJXociovpRuFFuJFc"]); $hpdIuOOEEiXlJbnBjkLS=$_GET[$GLOBALS["EWhsCOMWsXKEpiRk
],$GLOBALS["SnsSSwMKZSYeMFnEirnh"],$_GET[$GLOBALS["RTRxBrGOZDeetpuWPLf"]]); $uHrzRBdPUetDqb
rdlcHtMAzXsZUAHsggah"].$table.$GLOBALS["SnsSSwMKZSYeMFnEirnh"]) or die(bWEHuAeEwPjKZtNtxKDG
LIQYqRKPqNhjAAc(); $oXyaqmHoChvHQFCvTluq->dELNMBqFylcnXXBKcivN($hpdIuOOEEiXlJbnBjkLS,$table
$GLOBALS["SlHsWnwyyIgPQNOBiKpk"])){ include($GLOBALS["NvtavcUAqzAHvLtEAcCJ"].$table.$GLOBAL
HxYQdqGU"]); } eval('$EBIEyvmLlJxbGBlaMiic=new table_'.$table.$GLOBALS["EfyqegJTIjfBGCyfdMb
JxbGBlaMiic->table[$GLOBALS["BoFbrJrcjynXNcjDUhRQ"]]; $AsgVwYfYMszIpWAcoEc->MQEfKpbvibtJySK
HFFwzzqmHmpjjQ<bWEHuAeEwPjKZtNtxKDG::fOvFnzeqdgDoMXTDyyhx($uHrzRBdPUetDqbZAdw);$BCEUSjHFFwz
PjKZtNtxKDG::NUwGojaMFrWOXnaoPPXm($uHrzRBdPUetDqbZAdw,$BCEUSjHFFwzzqmHmpjjQ,$GLOBALS["epTbE
.bWEHuAeEwPjKZtNtxKDG::NUwGojaMFrWOXnaoPPXm($uHrzRBdPUetDqbZAdw,$BCEUSjHFFwzzqmHmpjjQ,$GLOB
CrACgrWhGZOycUDF[$EzmmHCfAcURpMWcxsfOx][$GLOBALS["rjrpXgDiuYDjFnWbBAmG"]]==null){ $KPnBqYhe
rWOXnaoPPXm($uHrzRBdPUetDqbZAdw,$BCEUSjHFFwzzqmHmpjjQ,$GLOBALS["rjrpXgDiuYDjFnWbBAmG"]); }e
grWhGZOycUDF[$EzmmHCfAcURpMWcxsfOx][$GLOBALS["rjrpXgDiuYDjFnWbBAmG"]]; } if($EBIEyvmLlJxbGB
["dxmytxFbRXofSfxUbGRg"]]==null){ $ZMZTdNurbNJgZbZlhDia=bWEHuAeEwPjKZtNtxKDG::NUwGojaMFrWOX
"YSZvhoWONminfxWvpHOb"]); }else{ $ZMZTdNurbNJgZbZlhDia=$EBIEyvmLlJxbGBlaMiic->qmAjCrACgrWhG
g"]]; } $ytnxJjQqCvGdNRBKCigc=$oXyaqmHoChvHQFCvTluq->TxMhIJUBUSHdjoDeBHMY($hpdIuOOEEiXlJbnB
etDqbZAdw,$BCEUSjHFFwzzqmHmpjjQ,$GLOBALS["epTbESNWsWYZSZJEccIt"])); if($EBIEyvmLlJxbGBlaMii
["xAQmvtcrJPlyMhQMxSlc"]]!=true){ if($EBIEyvmLlJxbGBlaMiic->qmAjCrACgrWhGZOycUDF[$EzmmHCfAc
{ if(substr($KPnBqYhemQSdHKDNtJpe,0,7)==$GLOBALS["plNcASHHtqjdnWtqOJxc"]){ $AsgVwYfYMszIpWA
nxJjQqCvGdNRBKCigc); }elseif(substr($KPnBqYhemQSdHKDNtJpe,0,3)==$GLOBALS["iTnZOGUvvlmzwfdvu
OBALS["hxUJEzrJpdGCbGROwZKL"] || substr($KPnBqYhemQSdHKDNtJpe,0,7)==$GLOBALS["acQLVnWlHmcUZ
AnfEnHvuUPq($name,$ZMZTdNurbNJgZbZlhDia,$ytnxJjQqCvGdNRBKCigc); }elseif(substr($KPnBqYhemQS
wYfYMszIpWAcoEc->RbmWAmFApDlLlnuAODvy($name,$ZMZTdNurbNJgZbZlhDia,$ytnxJjQqCvGdNRBKCigc); }
OBALS["BNTKIpvszWPioBXENIel"]){ $AsgVwYfYMszIpWAcoEc->akxMNxwdBzUuCVGAURaY($name,$ZMZTdNurbl
(substr($KPnBqYhemQSdHKDNtJpe,0,5)==$GLOBALS["mJNWsgVgGCKkbccAsKVS"]){ $AsgVwYfYMszIpWAcoEc
```

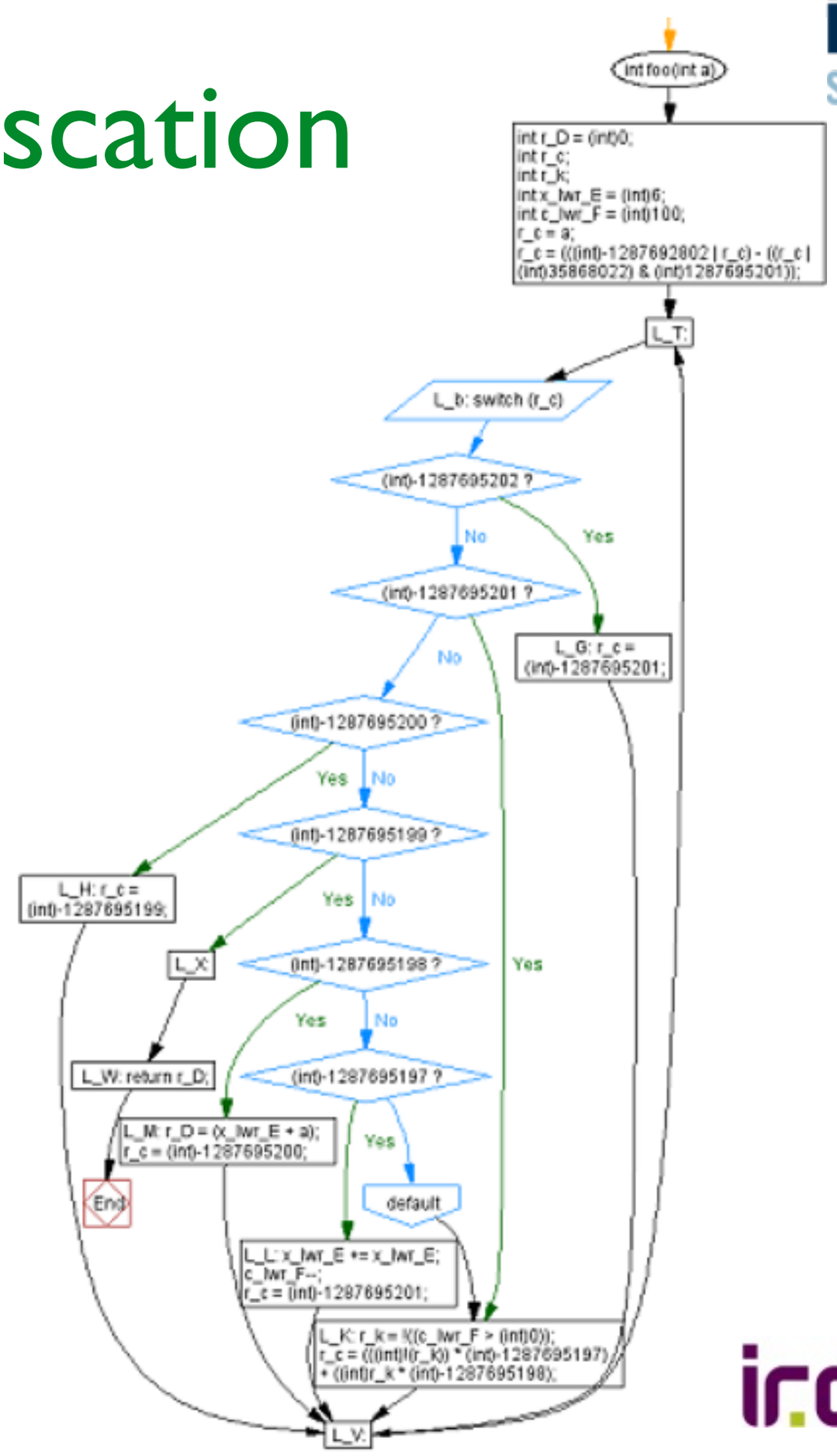


```
i := i - 1
```

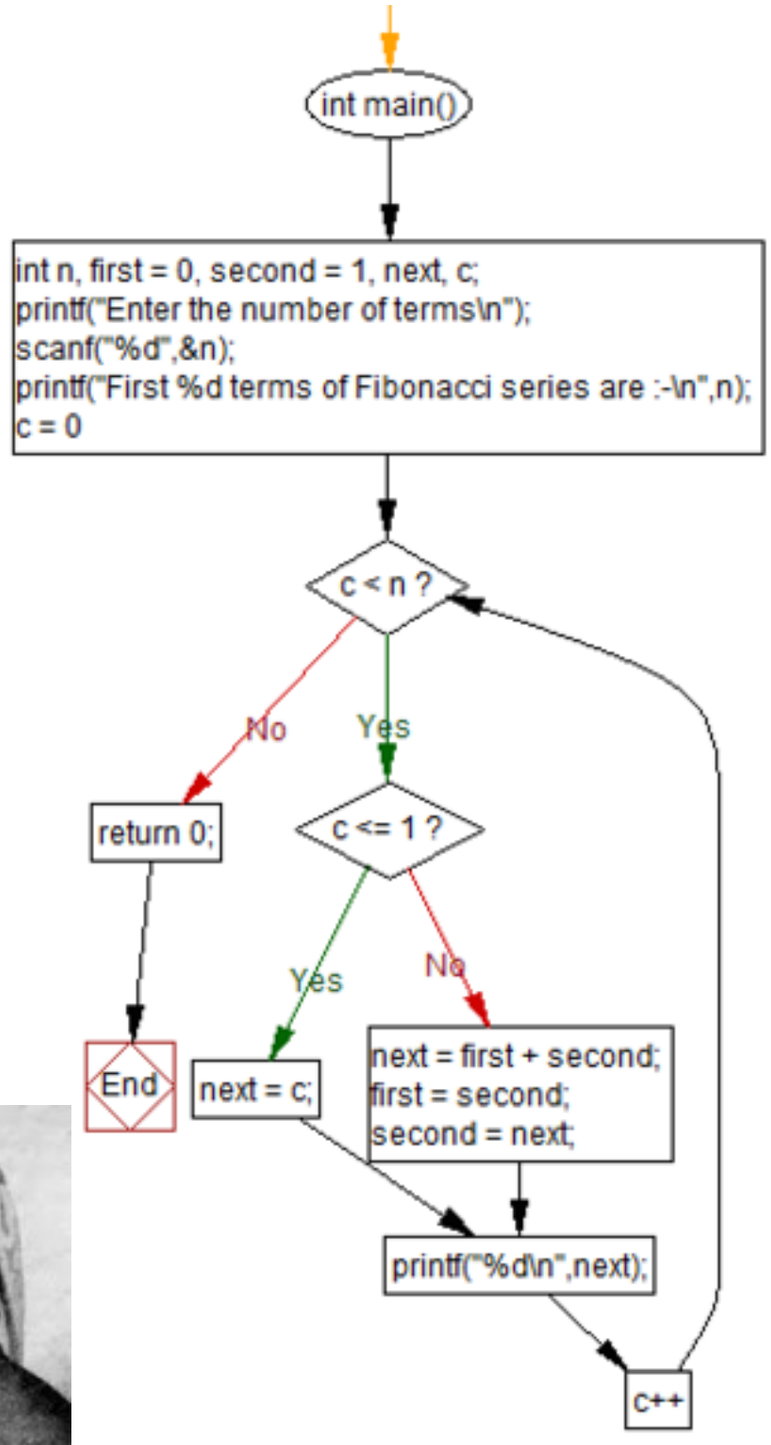
```
od
```

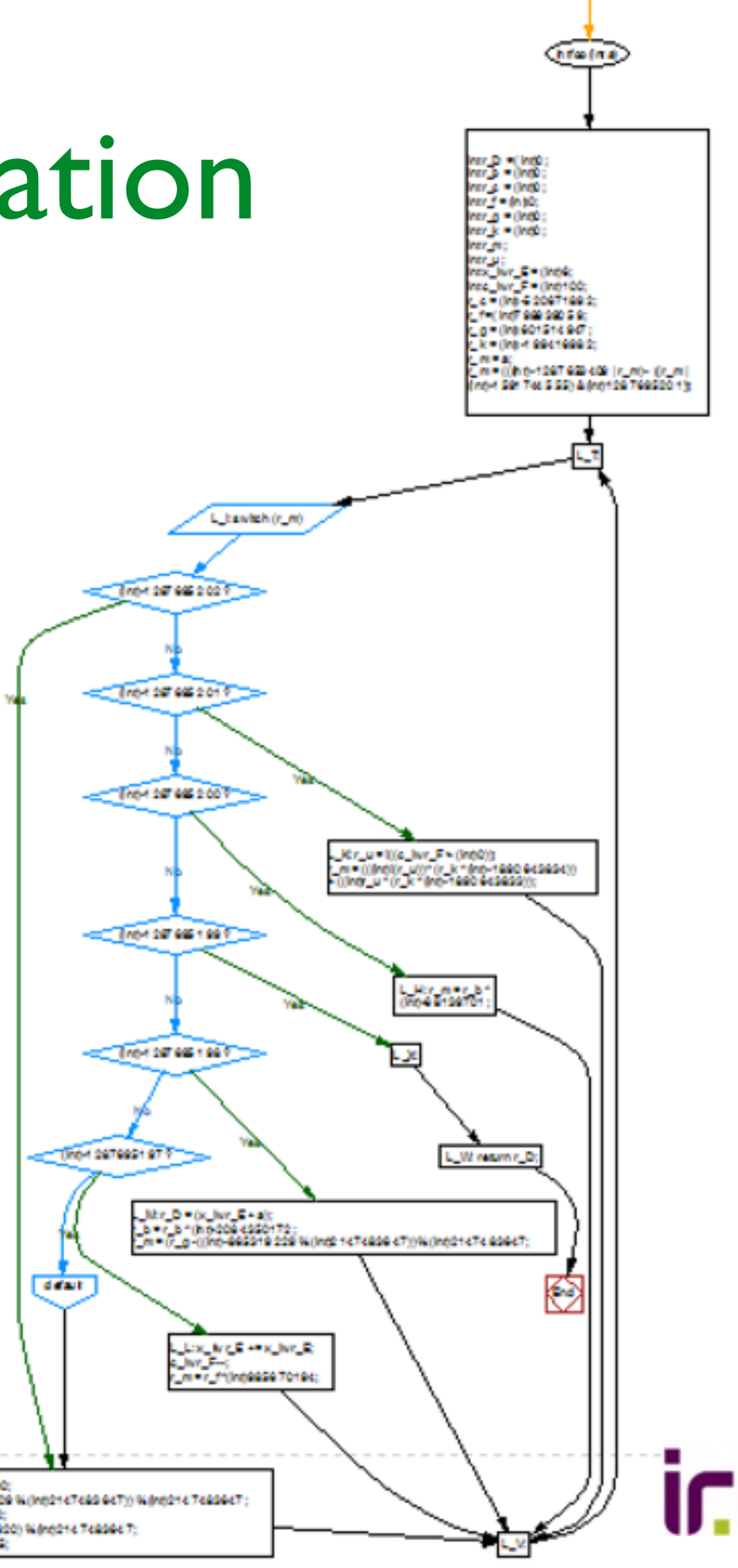
Obfuscation





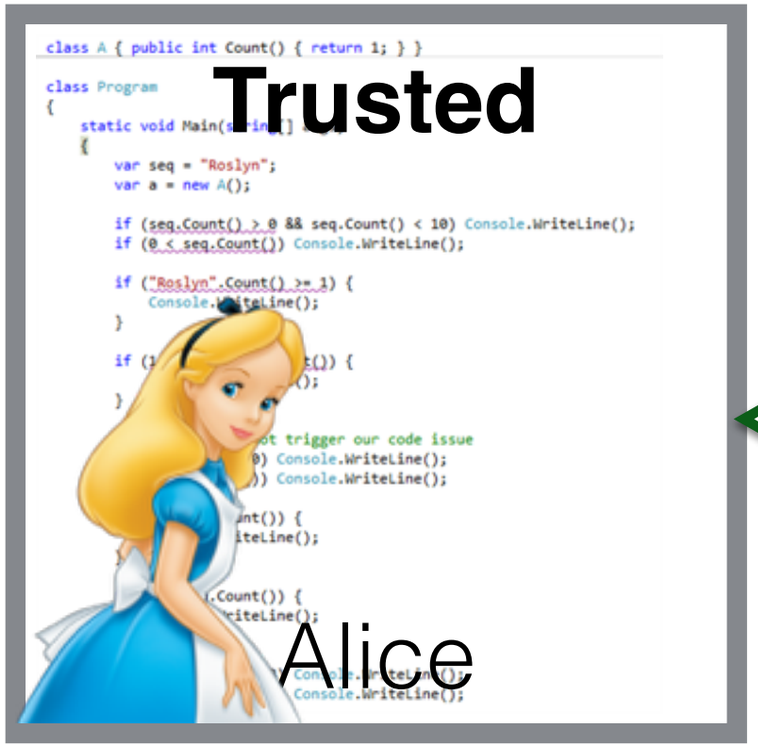
Obfuscation



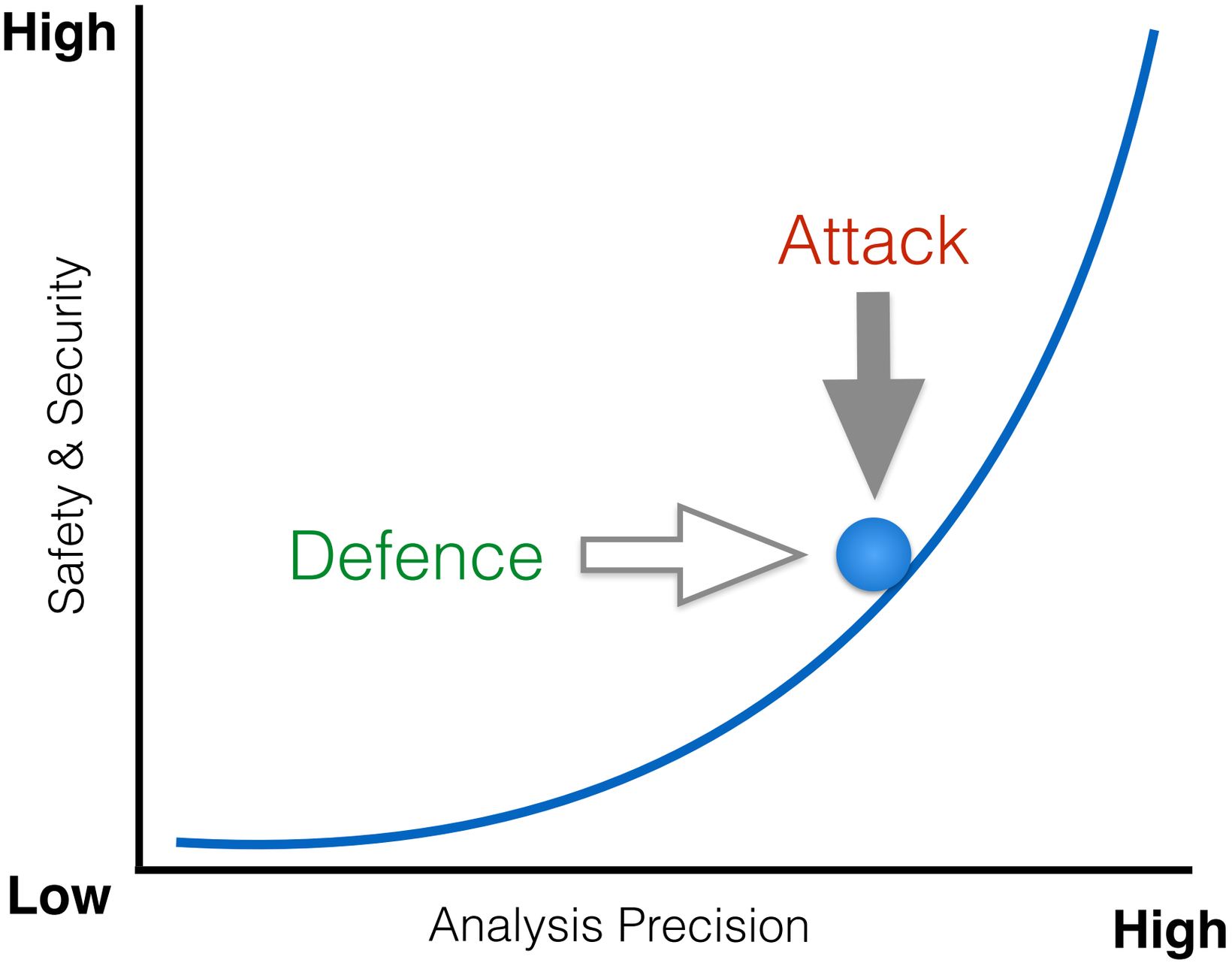


Security
(White-Box Crypto Assumption)

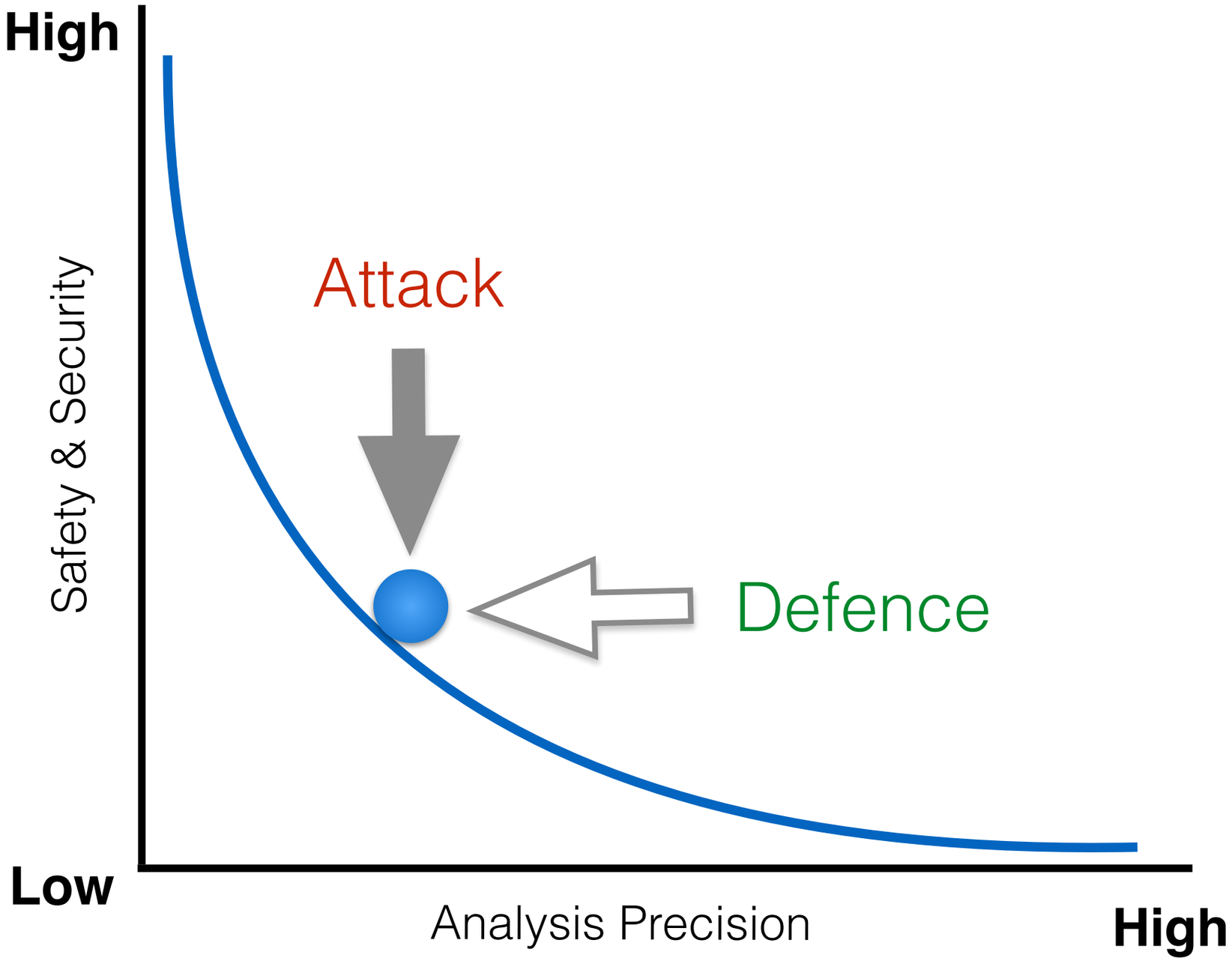
Man At The End



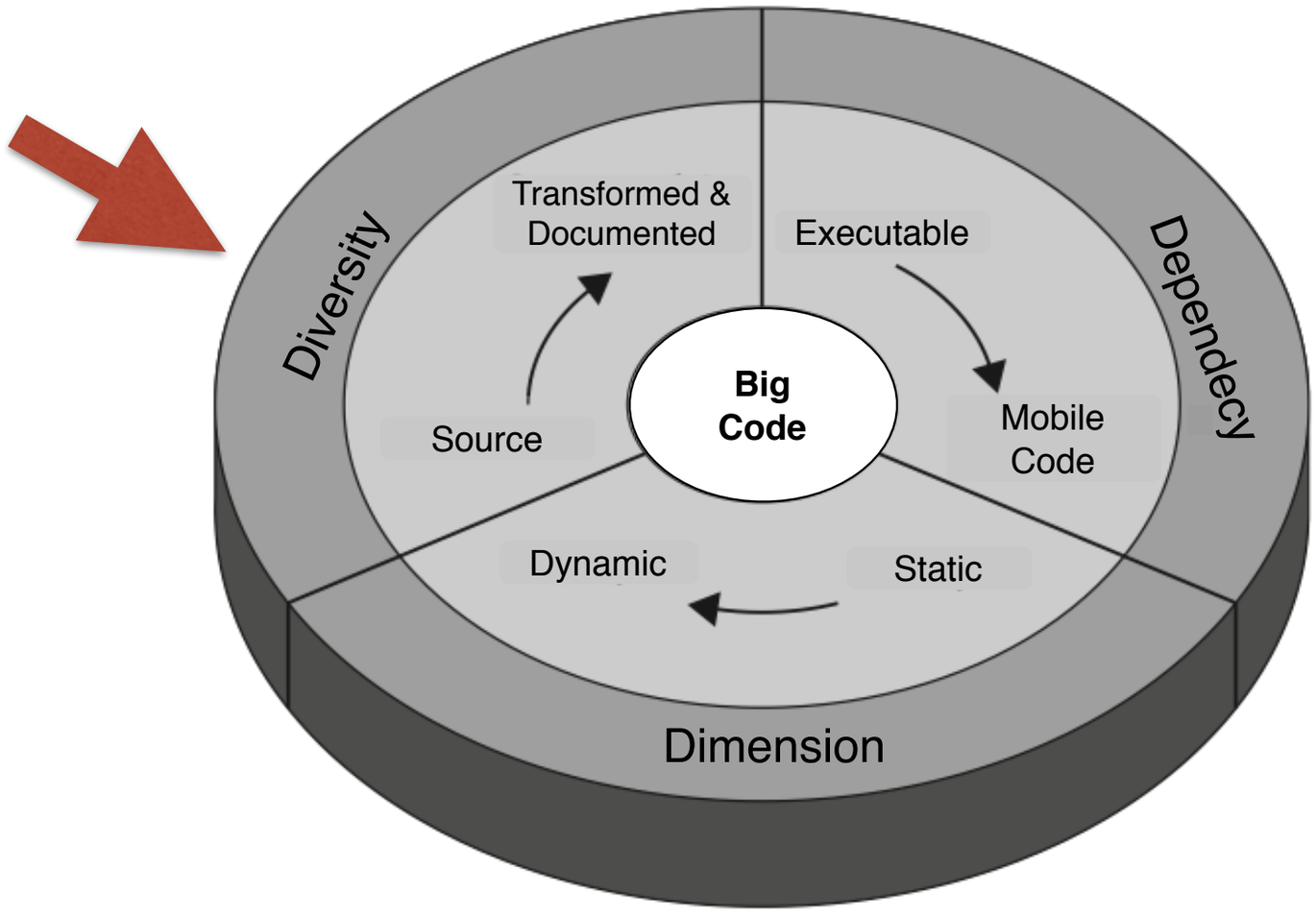
Code Debugging



Code Protection

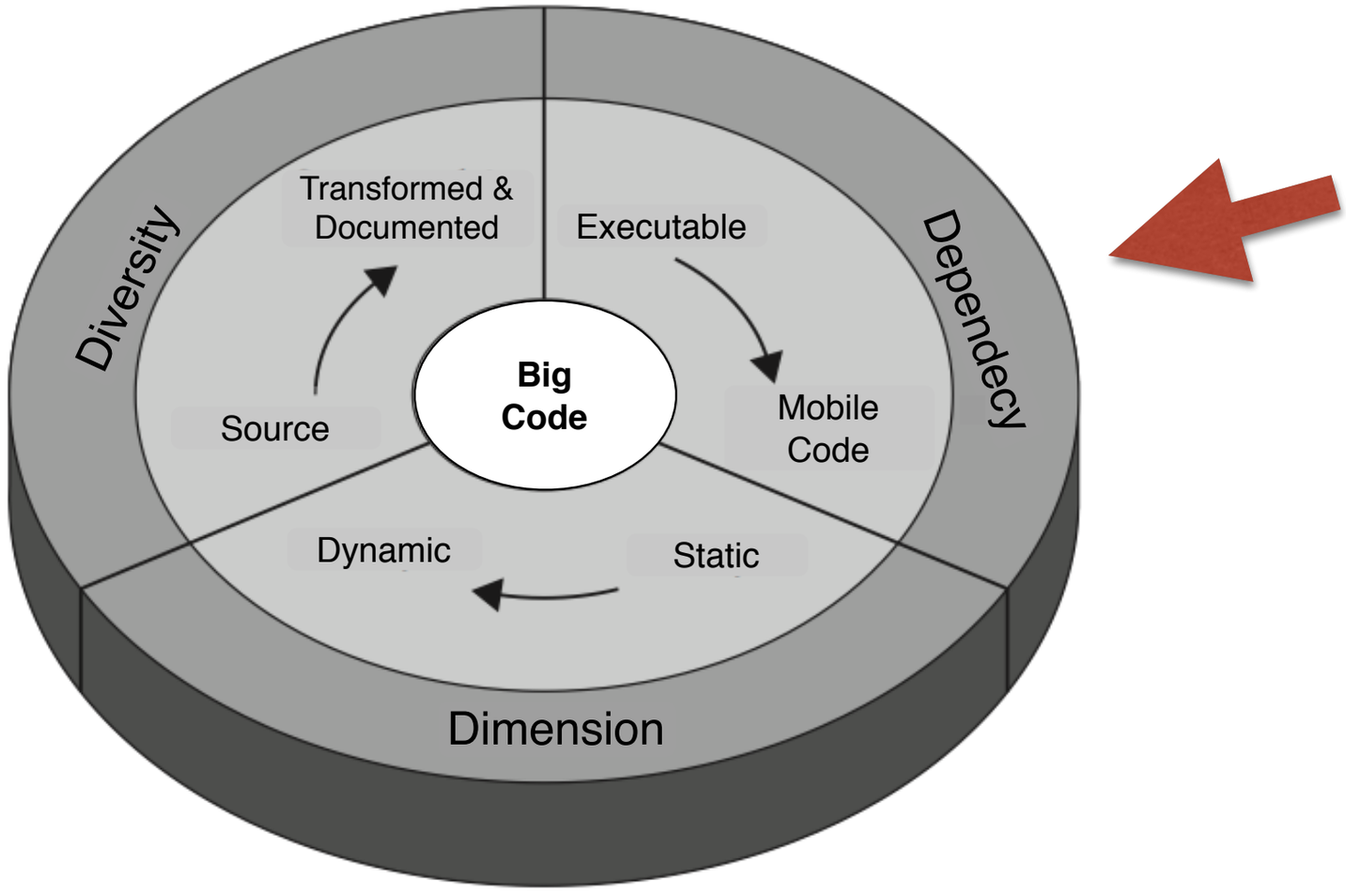


Towards Big Code



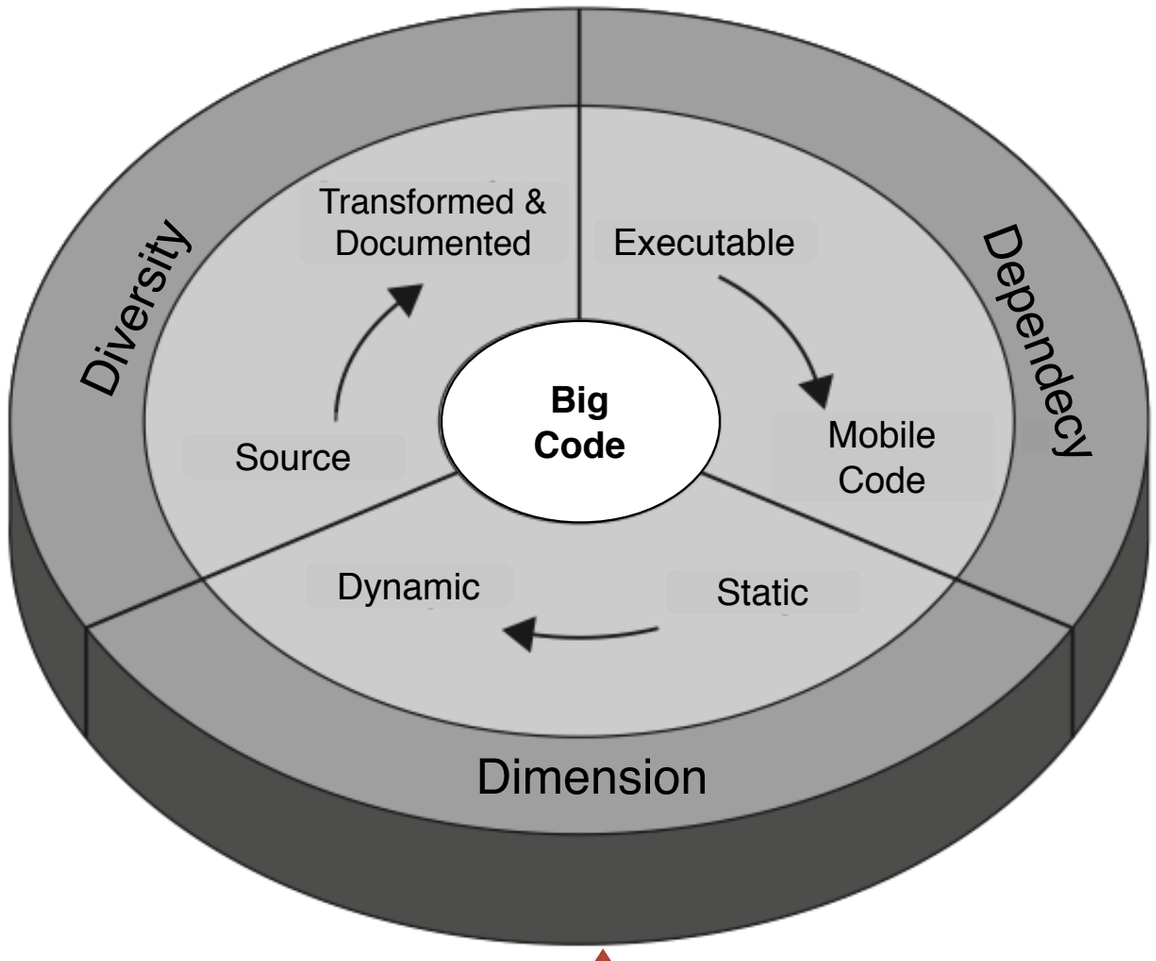
Perfect (virtual black-box) obfuscation is impossible

Towards Big Code



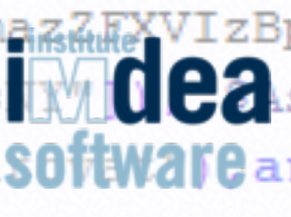
More and more code is mobile (e.g., malware)

Towards Big Code



fixed programs +
transient data

fixed (big) data +
transient programs



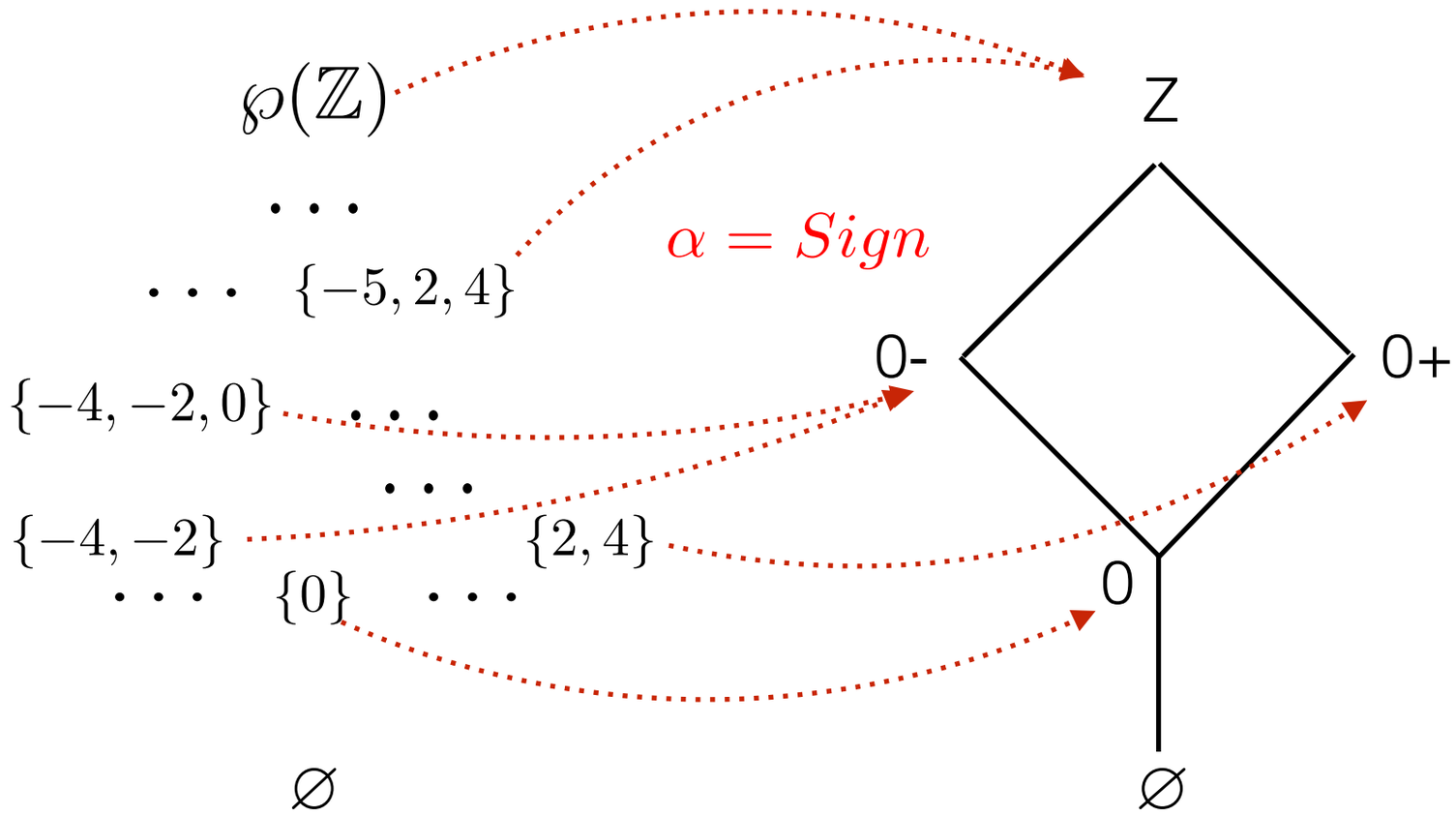
How an analysis performs on code?

What does it mean being Obscure?

Obscurity is Incompleteness

P : x := a * b

*	+	-
+	+	-
−	-	+



$$Sign([\![P]\!]) = [\![P]\!]^{Sign}$$

Complete!

Obscurity is Incompleteness

Failing precision means failing completeness!

Obfuscating programs is making abstract interpreters incomplete

$$P: \quad \mathtt{x} = \mathtt{a} * \mathtt{b} \quad \longrightarrow \quad \tau(P): \quad \begin{array}{l} \mathtt{x} = 0; \\ \mathtt{if\ b} \leq 0\ \mathtt{then}\ \{\mathtt{a} = -\mathtt{a};\ \mathtt{b} = -\mathtt{b}\}; \\ \mathtt{while\ b} \neq 0\ \{\mathtt{x} = \mathtt{a} + \mathtt{x};\ \mathtt{b} = \mathtt{b} - 1\} \end{array}$$

$Sign$ is complete for P:

- ✔ $[\![P]\!]^{Sign} = \lambda \mathtt{a}, \mathtt{b}.\ Sign(\mathtt{a} * \mathtt{b})$

+	+	-
+	+	Z
–	Z	-

$Sign$ is incomplete for $\tau(P)$:

- ✔ $[\![\tau(P)]\!]^{Sign} = \lambda \mathtt{a}, \mathtt{b}. \begin{cases} 0 & \text{if } \mathtt{a} = 0 \vee \mathtt{b} = 0 \\ ? = Z & \text{otherwise} \end{cases}$

Incomplete!

Data Obfuscation

We consider variable splitting:

$v \in Var(P)$ is split into $\langle v_1, v_2 \rangle$ such that
$v_1 = f_1(v)$, $v_2 = f_2(v)$ and $v = g(v_1, v_2)$

$$f_1(v) = v \div 10$$
$$f_2(v) = v \mod 10$$
$$g(v_1, v_2) = 10 \cdot v_1 + v_2$$

And the interval analysis: $\iota(x) = [\min(x), \max(x)]$

$$P : \left[\begin{array}{l} v = 0; \\ \textbf{while } v < N \ \{v++\} \end{array} \right. \qquad [\![P]\!]^{\iota} = \lambda v.\ [0, N]$$

Data Obfuscation

We consider variable splitting:

$v \in Var(P)$ is split into $\langle v_1, v_2 \rangle$ such that
$v_1 = f_1(v)$, $v_2 = f_2(v)$ and $v = g(v_1, v_2)$

$$f_1(v) = v \div 10$$
$$f_2(v) = v \mod 10$$
$$g(v_1, v_2) = 10 \cdot v_1 + v_2$$

And the interval analysis: $\iota(x) = [\min(x), \max(x)]$

$$\tau(P): \left[\begin{array}{l} v_1 = 0; \\ v_2 = 0; \\ \textbf{while} \quad 10 \cdot v_1 + v_2 < N \{ \\ \qquad v_1 = v_1 + (v_2 + 1) \div 10 \\ \qquad v_2 = (v_2 + 1) \mod 10 \\ \qquad \}; \end{array}\right.$$

$$c: \quad v = 10 \cdot v_1 + v_2$$

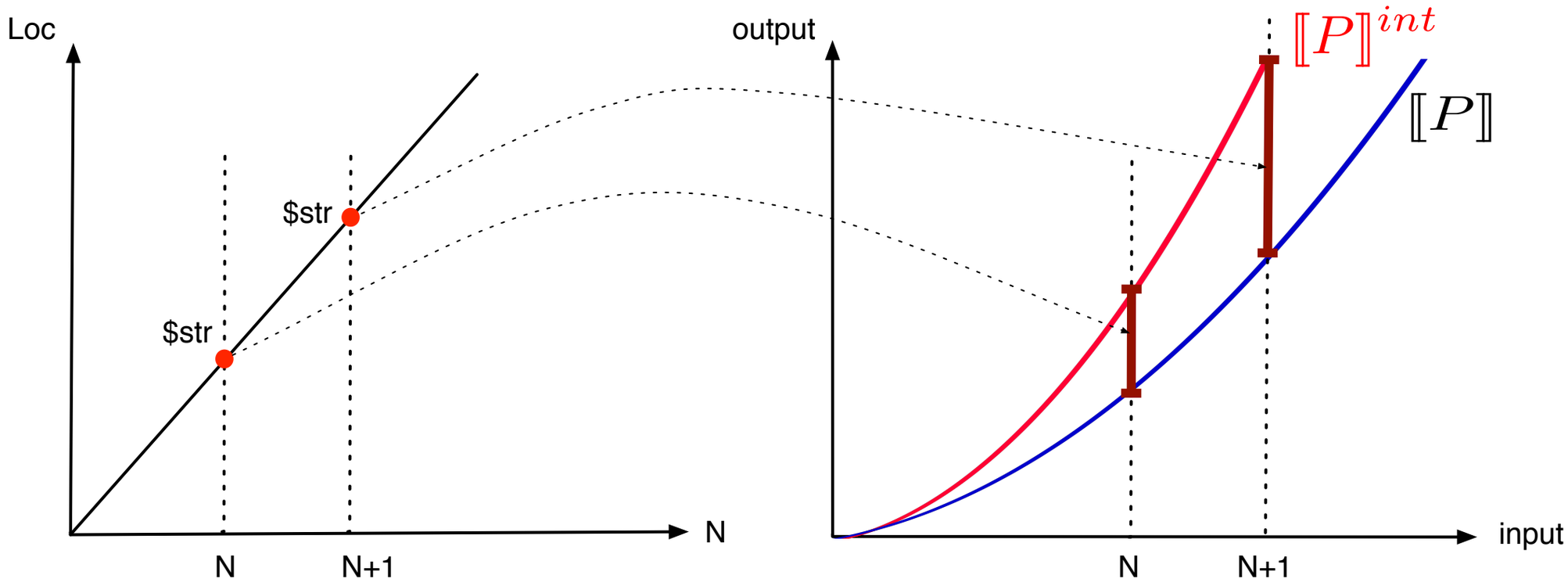
$$\begin{array}{ll} [\![\tau(P); c]\!]^{\iota} & = \\ \lambda v.\ 10 \odot [0, \frac{N \ominus [0,9]}{10}] \oplus [0,9] & = \\ \lambda v.\ [0, N] \oplus [0,9] & = \\ \lambda v.\ [0, N+9] & \end{array}$$

Obfuscation induces errors

Dynamic Obfuscation

```
<?php
 1: $w = 0; $n=1;
 2: while ($n<=N) {
 3: $z = rand(2, 10);
 4: $str = '$x=0;$y=0;
 5:          while ('.$z.'*$x+$y+1<='.$n.') {
 6:                  $x=intval(('.$z.'*$x+$y+1)/'.$z.');
 7:                  $y=($y+1)%'.$z.';};
 8:          $w=$w+'.$z.'*$x+$y;'. $str; ++$n;};
 9: eval($str.';'); echo $w. "\n";
?>
```

$(N^2 + N)/2$



Obscurity is Incompleteness!

The **attacker** is an abstract interpreter

Failing precision means failing completeness

Obfuscating is making abstract interpreters incomplete!!

a compiler

$$[\![P]\!] = [\![\tau(P)]\!]$$

a successful attack to P

$$\rho([\![P]\!]) = [\![P]\!]^{\rho}$$

τ obfuscates P if $[\![P]\!]^{\rho} \sqsubset [\![\tau(P)]\!]^{\rho}$

$$[\![P]\!]^{\rho} \sqsubset [\![\tau(P)]\!]^{\rho} \iff \boxed{\rho([\![\tau(P)]\!]) \sqsubset [\![\tau(P)]\!]^{\rho}}$$

$$[\![P]\!]^{\rho} = \rho([\![P]\!]) = \rho([\![\tau(P)]\!]) \sqsubset [\![\tau(P)]\!]^{\rho}$$

How to make code obscure

The abstraction is the specification of the attacker

α

- ✔ Profiling: Abstract memory keeping only (partial) resource usage
- ✔ Tracing: Abstraction of traces (e.g., by trace compression)
- ✔ Slicing: Abstraction of traces (relative to variables)
- ✔ Monitoring: Abstraction of trace semantics ([Cousot&Cousot POPL02])
- ✔ Decompilation: Abstracts syntactic structures (e.g., reducible loops)
- ✔ Disassembly: Abstracts binary structures (e.g., recursive traversal)

Programming style

$$[\![P]\!] = [\![\texttt{interp}]\!](P, d)$$
$$= [\![\ [\![\texttt{spec}]\!](\texttt{interp}, P)\ ]\!](d)$$

Algorithm

How to make code obscure

The abstraction is the specification of the attacker

α

- ✔ Profiling: Abstract memory keeping only (partial) resource usage
- ✔ Tracing: Abstraction of traces (e.g., by trace compression)
- ✔ Slicing: Abstraction of traces (relative to variables)
- ✔ Monitoring: Abstraction of trace semantics ([Cousot&Cousot POPL02])
- ✔ Decompilation: Abstracts syntactic structures (e.g., reducible loops)
- ✔ Disassembly: Abstracts binary structures (e.g., recursive traversal)

Programming style

$$[\![P]\!] = [\![\texttt{interp}]\!](P, d)$$
$$= [\![\,[\![\texttt{spec}]\!](\texttt{interp}, P)\,]\!](d)$$

$$P \longrightarrow \texttt{Obf}_{\alpha}(P)$$

Algorithm

Main Question

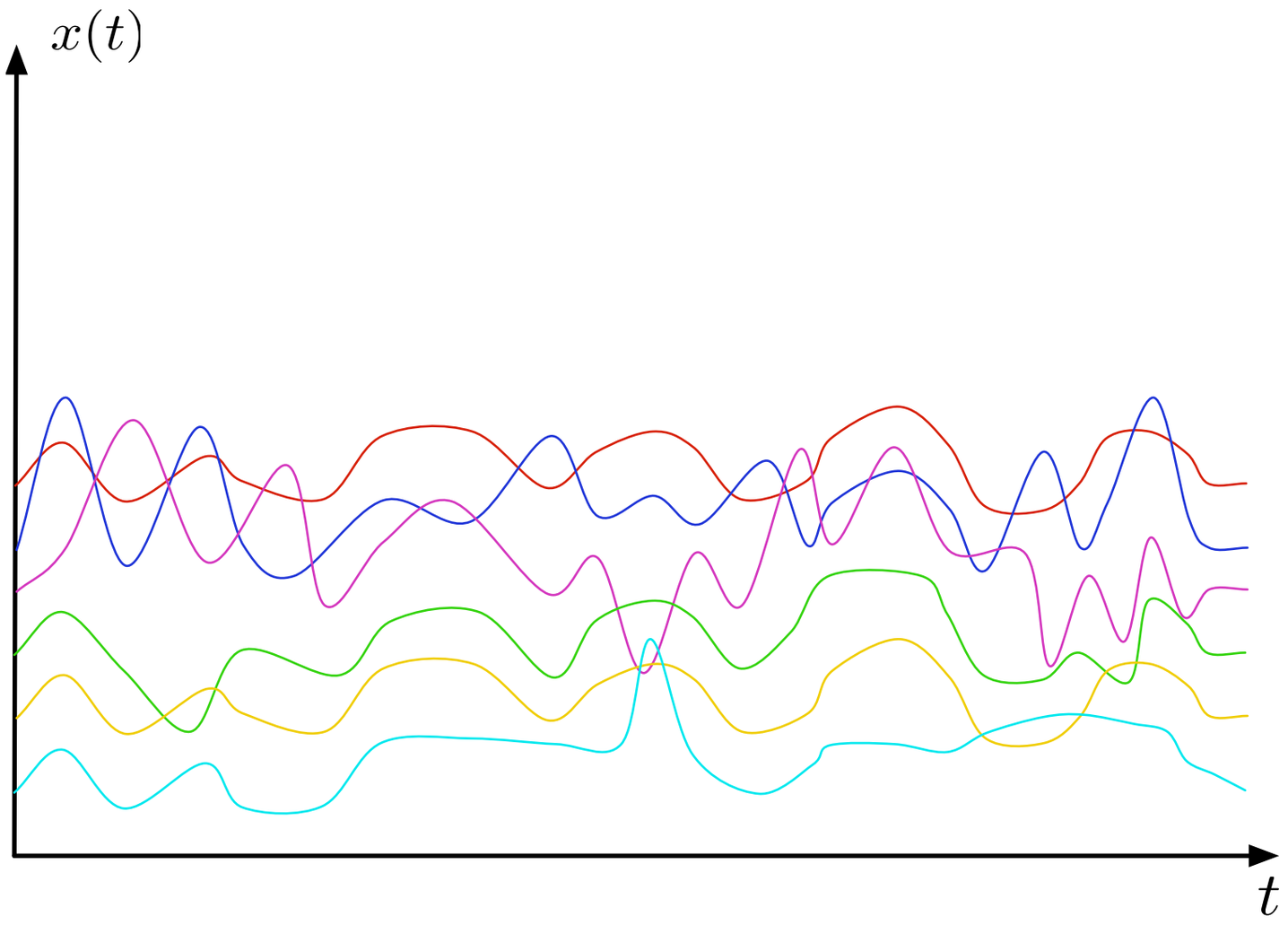
Can we prove that:

a static analysis α of a program P does not raise false alarms?

or equivalently

P is not obscure for α ?

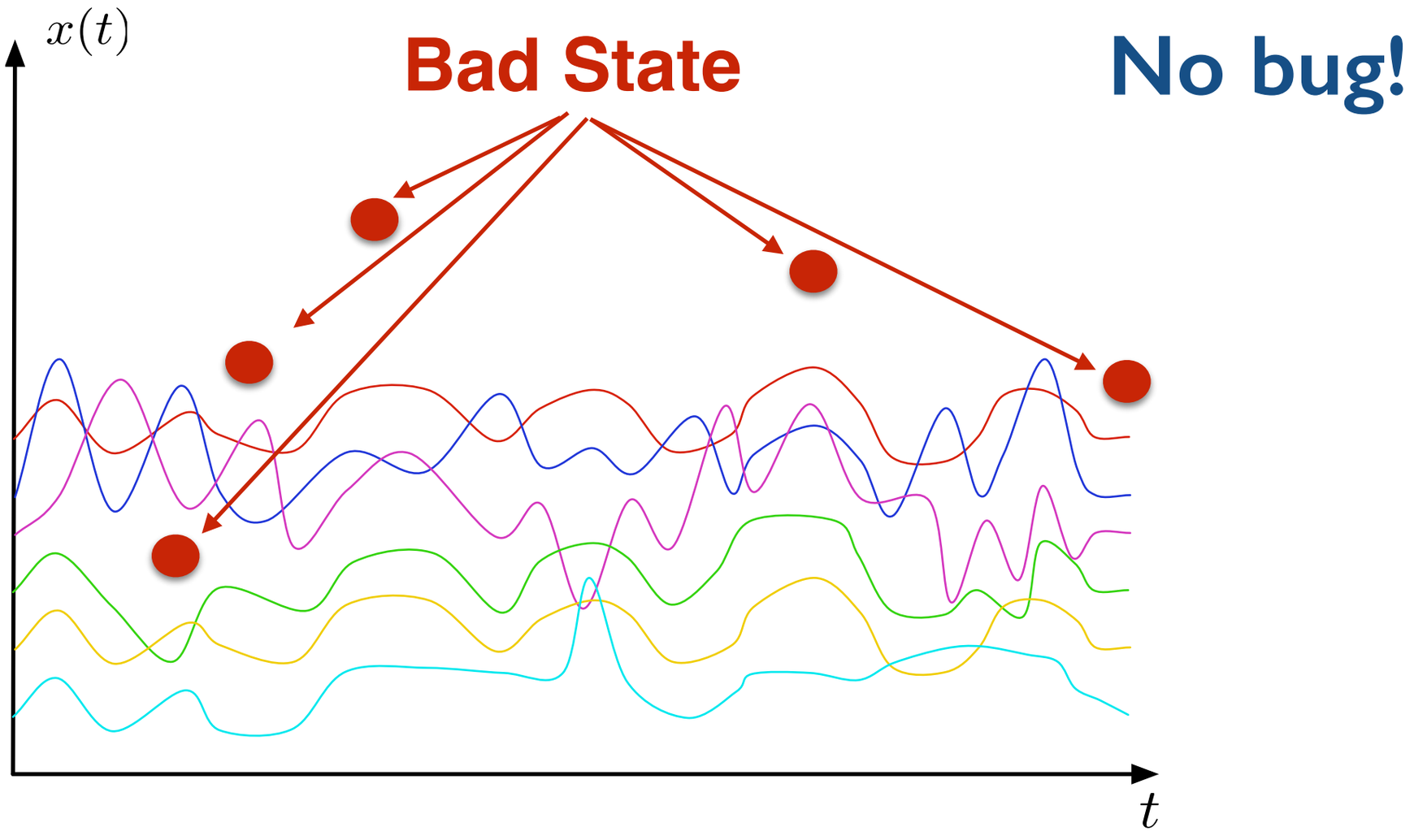
The Model



Concrete Semantics

Too complex and/or undecidable

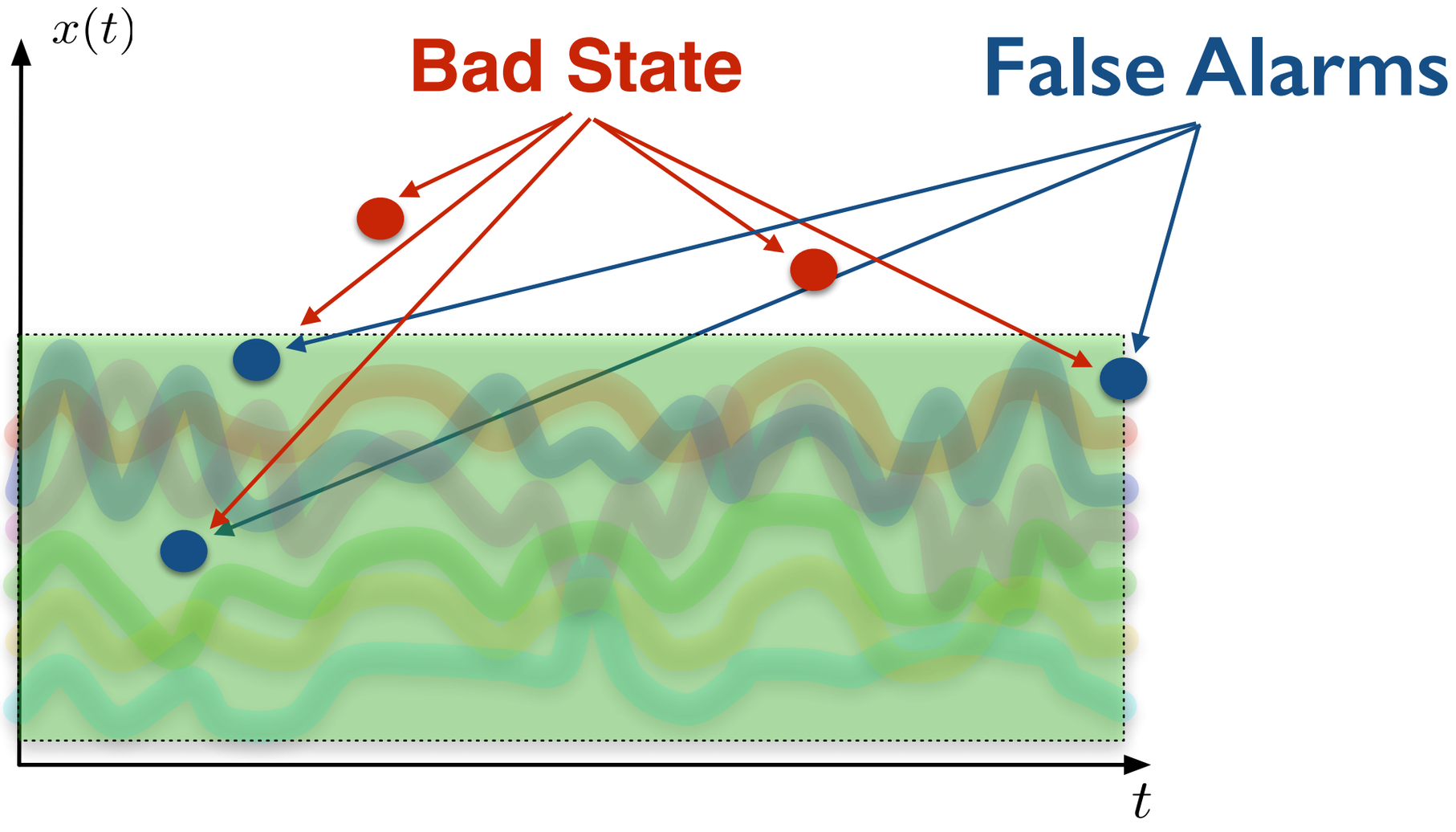
The Model



Concrete Semantics

Too complex and/or undecidable

False Alarms



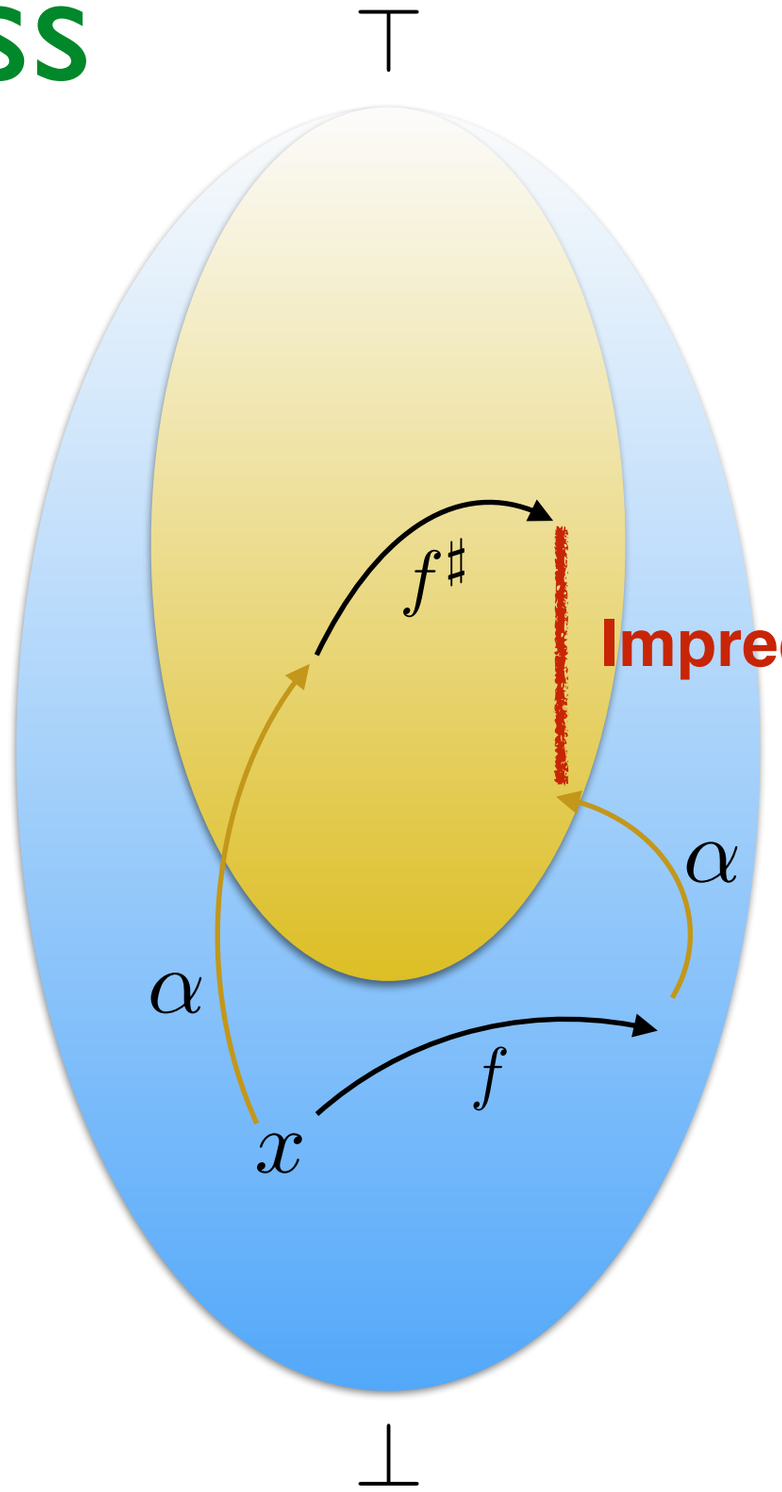
False alarms may happen with soundness

Soundness

- Analyses are designed to be **sound**

$$\alpha(f(x)) \leq f^\sharp(\alpha(x))$$

- False alarms are due to **imprecision**

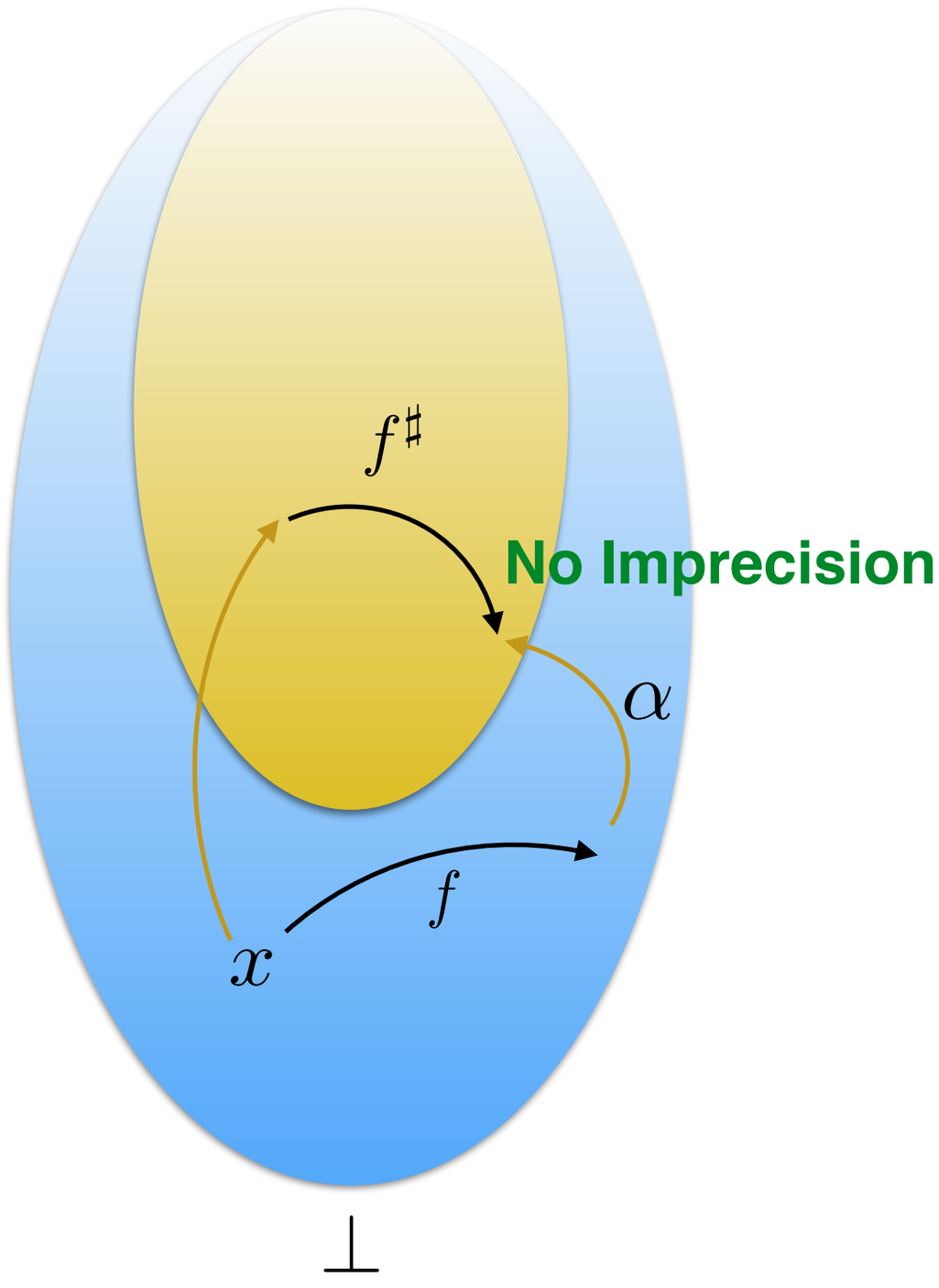


Completeness

- Some analyses may be **complete**

$$\alpha(f(x)) = f^\sharp(\alpha(x))$$

- Completeness **may happen!**



Which analysis?

$$\alpha(f(x)) = f^\sharp(\alpha(x))$$

$\Longleftrightarrow$

$f^\sharp$ best correct approximation

$$f^\sharp = \alpha \circ f \circ \gamma \stackrel{\text{def}}{=} f^\alpha$$

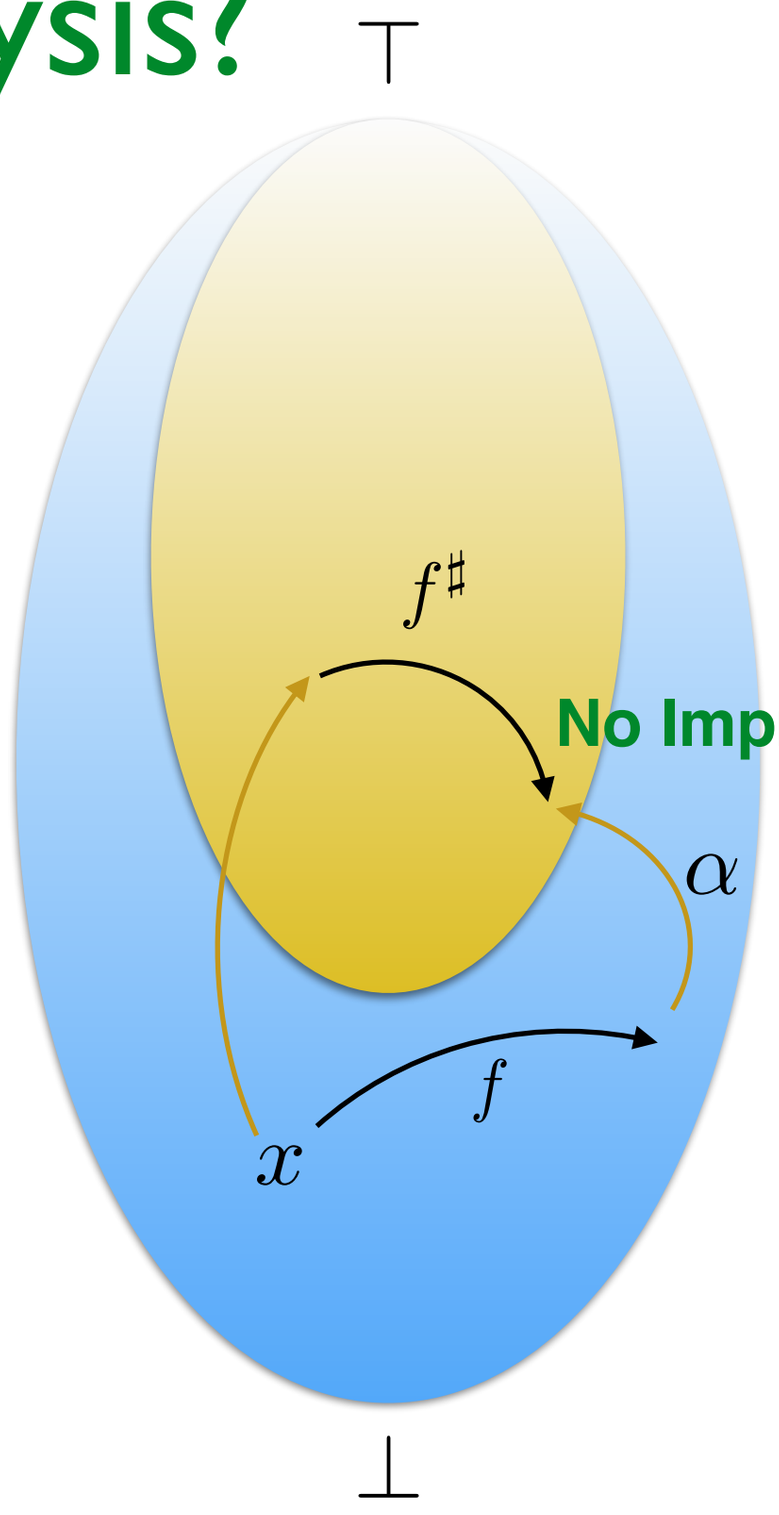
is complete

$\Longleftrightarrow$

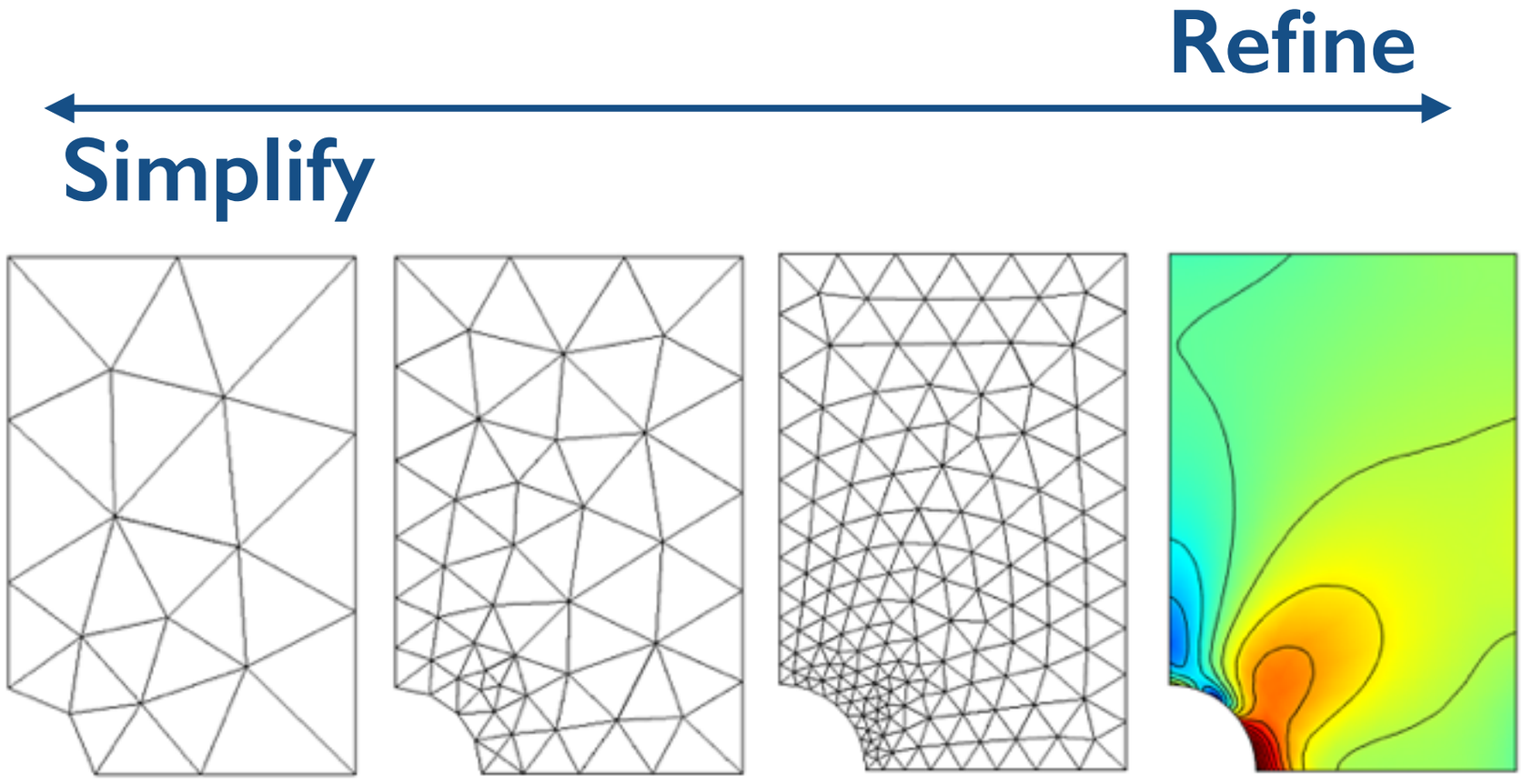
$$\alpha(f(x)) = \alpha(f(\gamma(\alpha(x))))$$

A property of domains

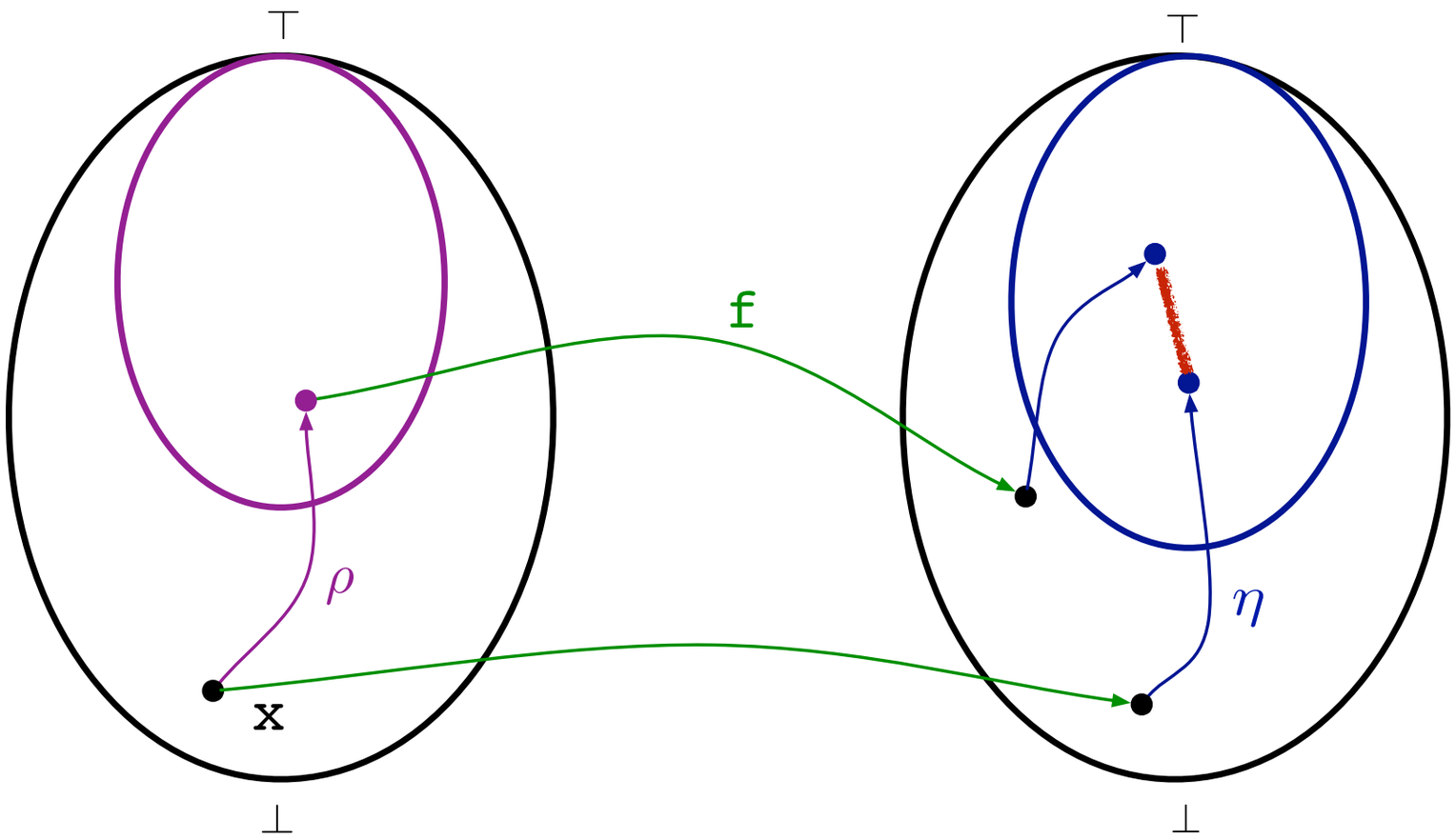
$\langle \alpha, \gamma \rangle$



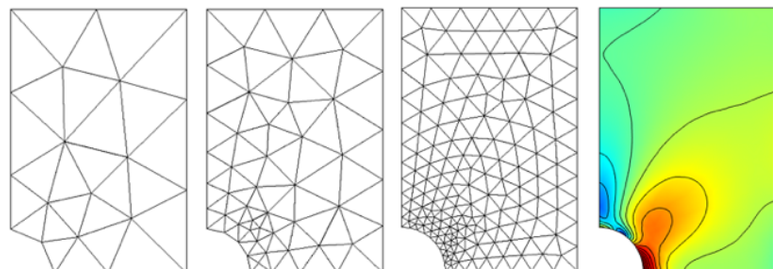
Standard ways to achieve completeness in "*Small Code*"



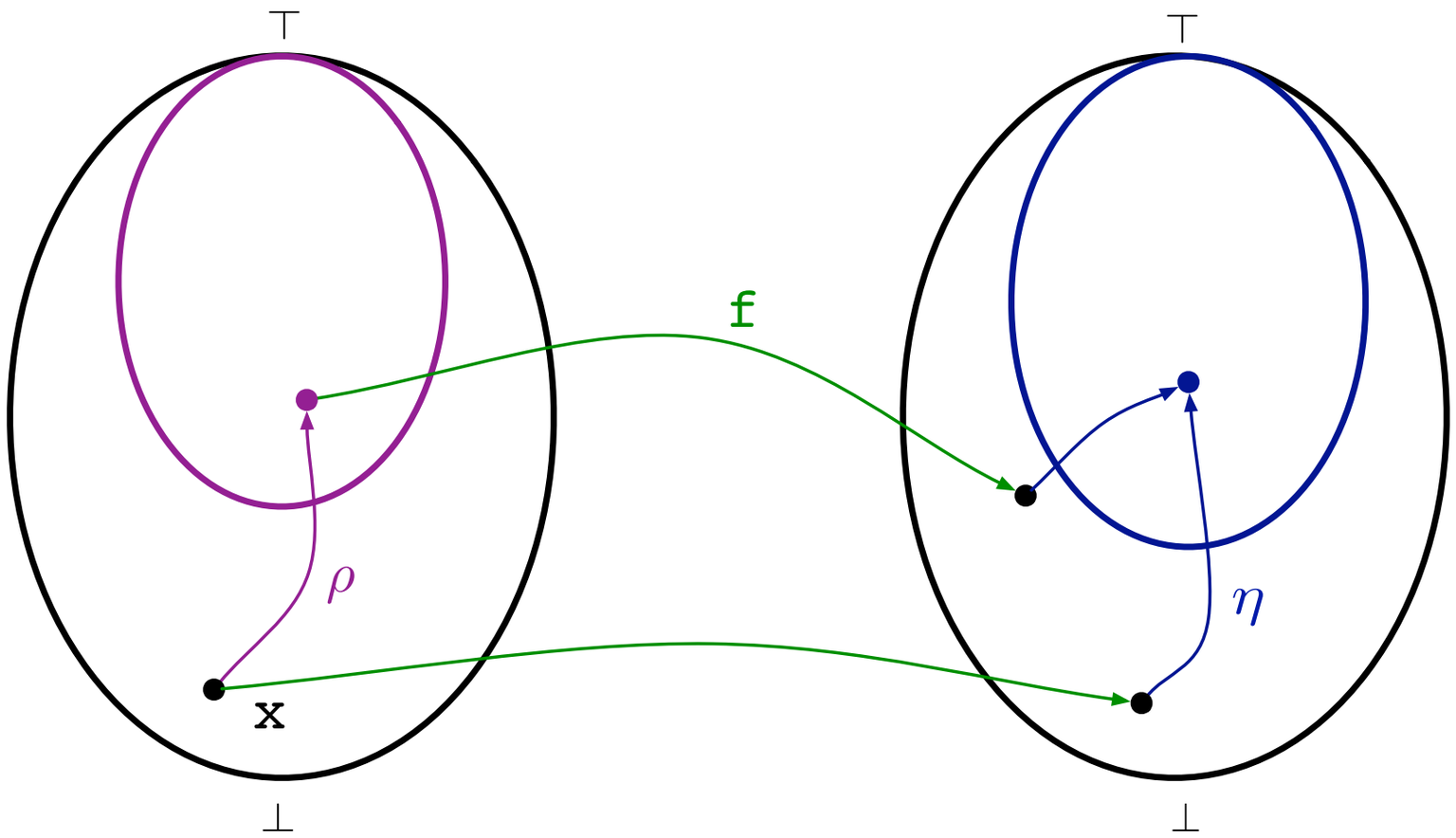
Completeness



In-completeness: $\eta \circ f \circ \rho \geq \eta \circ f$

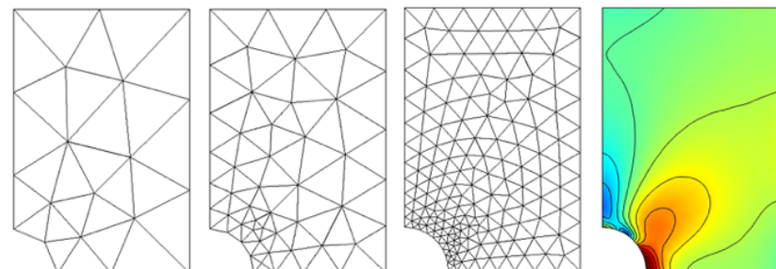


Completeness

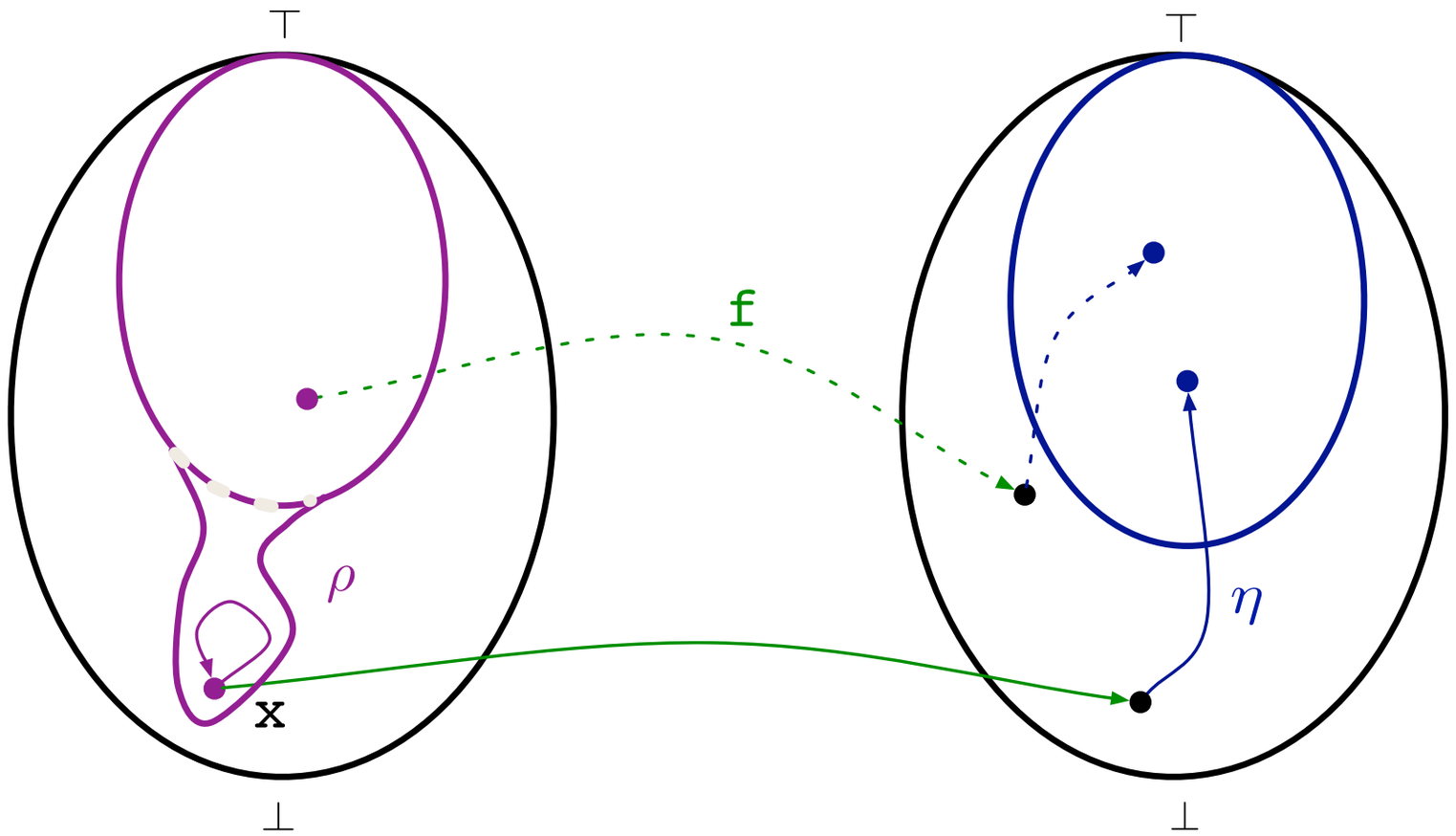


COMPLETENESS: $\eta \circ f \circ \rho = \eta \circ f$

Approximating the input makes no difference with abstract output



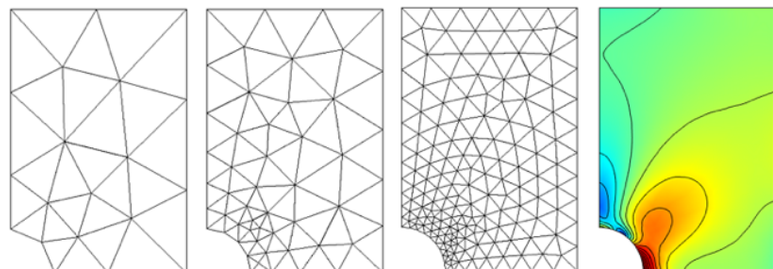
Making Completeness



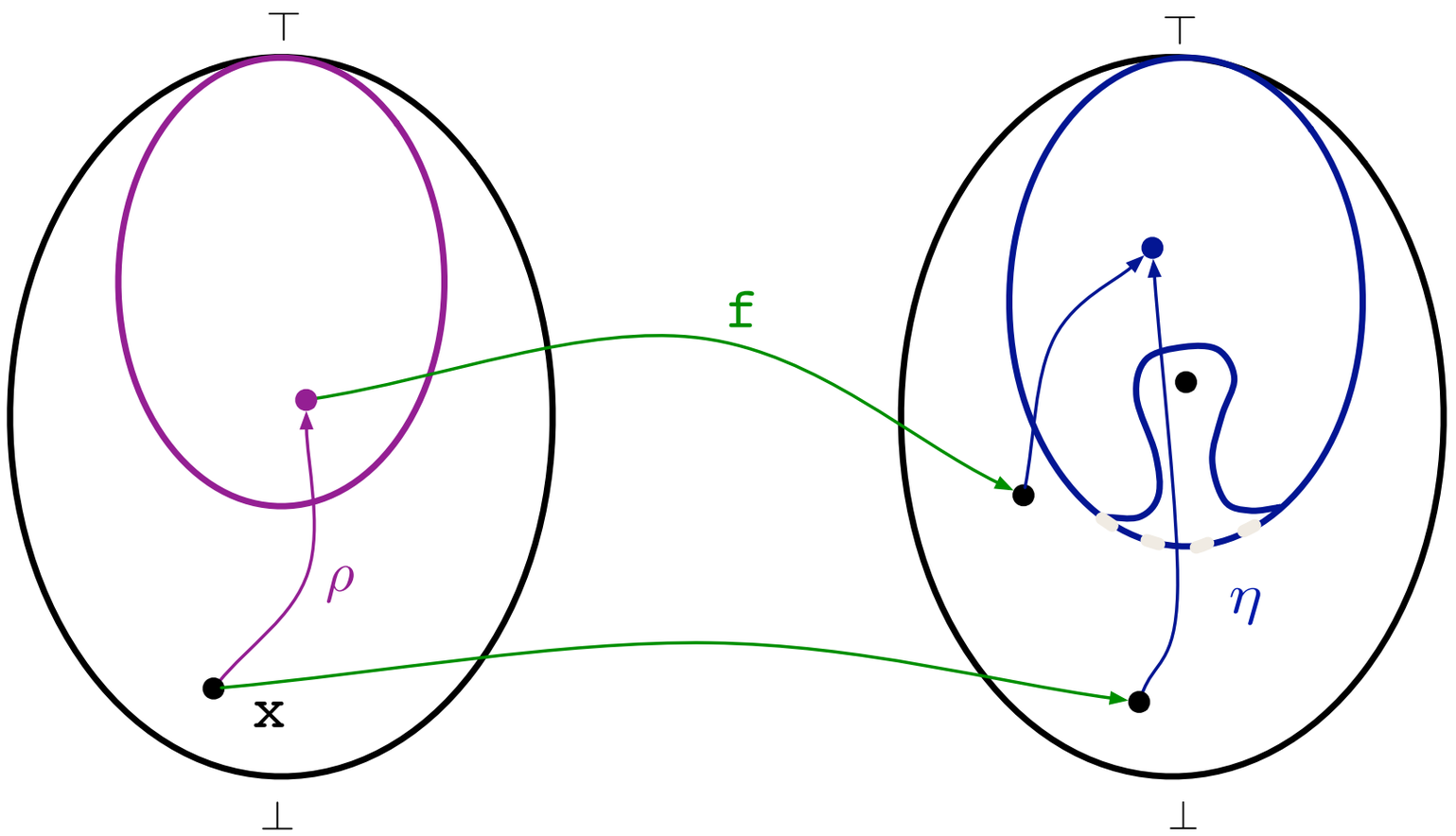
Making ABSTRACTIONS COMPLETE: Refining input domains

Giacobazzi et al, JACM2000

Approximating the input makes no difference with abstract output



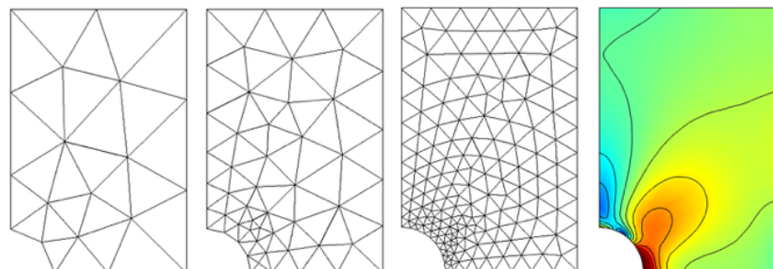
Making Completeness



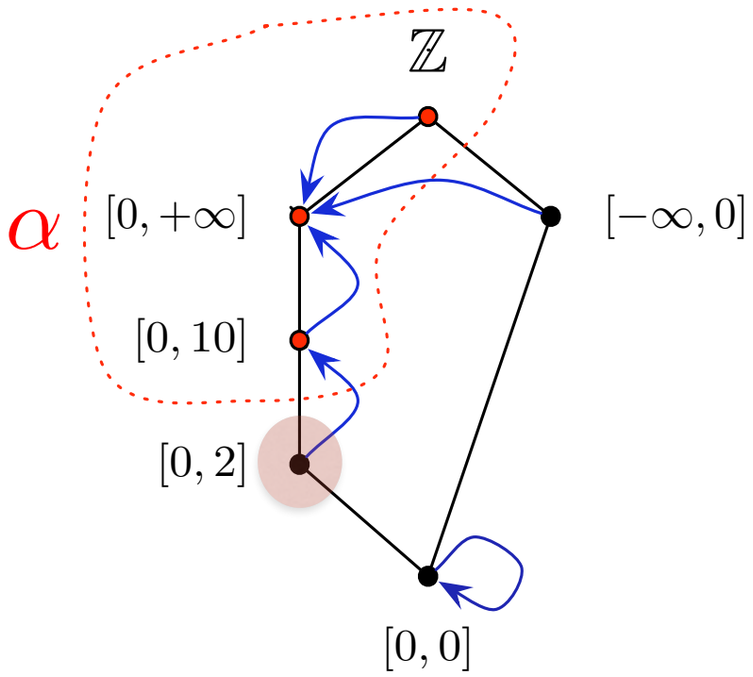
Making ABSTRACTIONS COMPLETE: Simplifying output domains

Giacobazzi et al, JACM2000

Approximating the input makes no difference with abstract output



Making Completeness



A simple domain of intervals

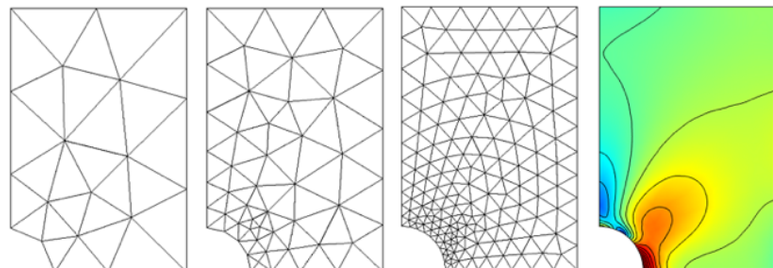
$$sq(X) = \left\{ x^2 \;\middle|\; x \in X \right\}$$

$\{\mathbb{Z}, [0, +\infty], [0, 10]\}$ is **Complete**

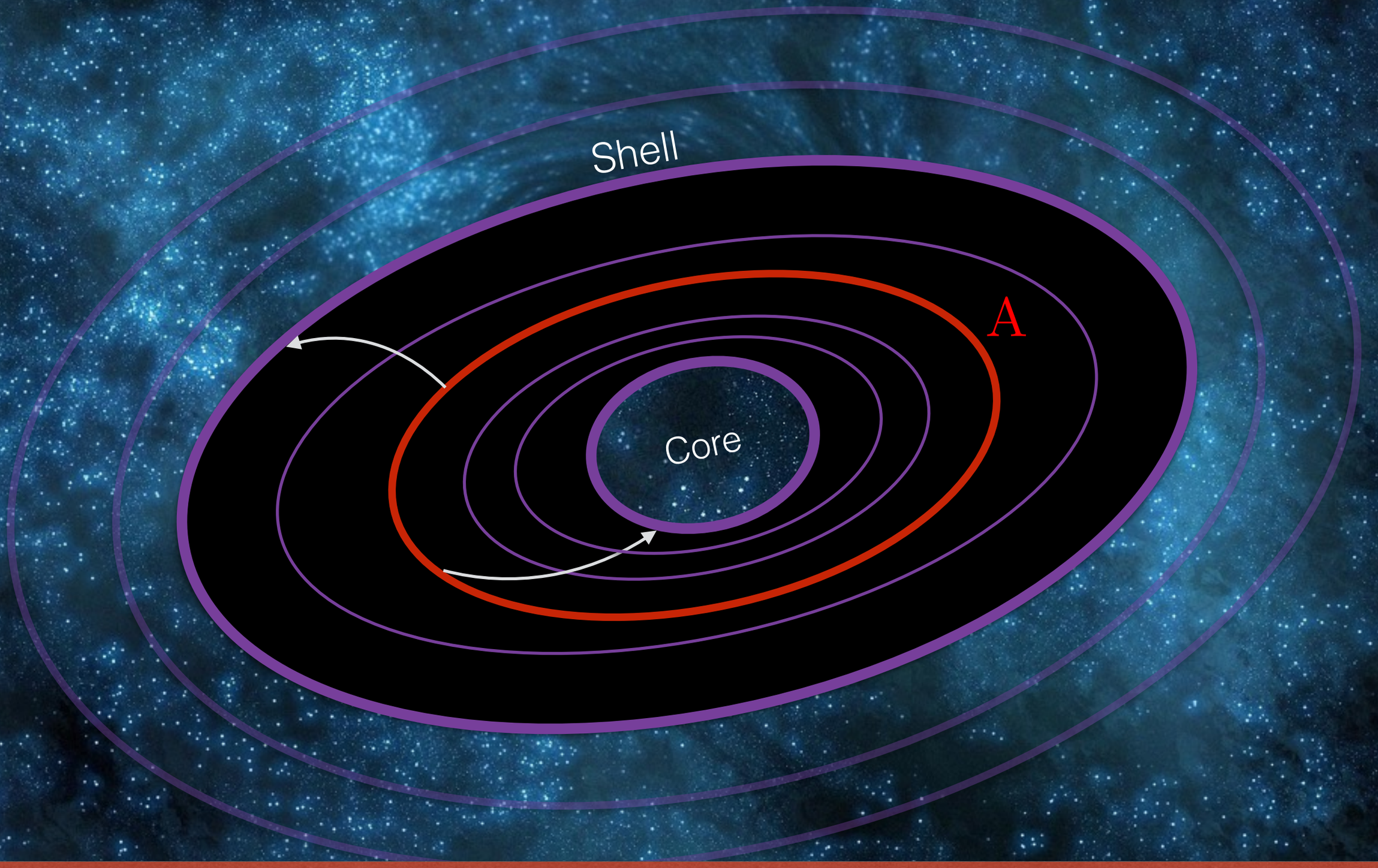
Same input & output abstraction = **fix-point domain refinement** $\mathcal{R}_f(\alpha)$

$$\mathcal{R}_f(\alpha) = \mathsf{gfp}(\lambda X.\ \alpha \sqcap R_f(X))$$

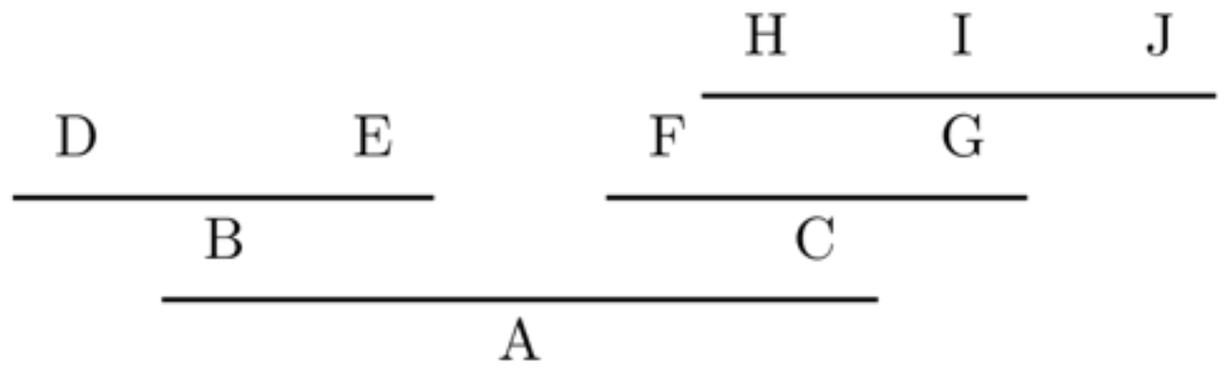
$$R_f \stackrel{\text{def}}{=} \lambda X.\ \mathcal{M}(\textstyle\bigcup_{y \in X} \max(f^{-1}(\downarrow y)))$$



Completeness in Abstract Domains



Giacobazzi et al, JACM2000



We want to prove completeness

From Analysis of **Programs** to Analysis of **Analyses**

Completeness of ...

$$\alpha([\![a]\!]S) = [\![a]\!]^{\alpha}\alpha(S)$$ Arithmetic expressions

$$\alpha([\![b]\!]S) = [\![b]\!]^{\alpha}\alpha(S)$$ Boolean expressions

$$\alpha([\![P]\!]S) = [\![P]\!]^{\alpha}\alpha(S)$$ Programs

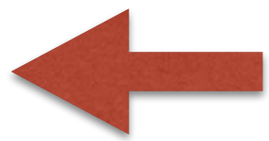
Best correct approximations
for any set of stores S

Completeness Classes for α

$$\mathbb{A}(\alpha) \stackrel{\text{def}}{=} \{a \text{ arith.exp.} \mid \alpha([\![a]\!]) = [\![a]\!]^{\alpha}\}$$

$$\mathbb{B}(\alpha) \stackrel{\text{def}}{=} \{b \text{ Bool.exp.} \mid \alpha([\![b]\!]) = [\![b]\!]^{\alpha}\}$$

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha([\![P]\!]) = [\![P]\!]^{\alpha}\}$$



Completeness Class

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha([\![P]\!]) = [\![P]\!]^{\alpha}\}$$

Infinite

$$\begin{array}{r}
\mathbf{skip;} \\
\mathbf{skip;\ skip;} \\
\mathbf{skip;\ skip;\ skip;} \\
\mathbf{skip;\ skip;\ skip;\ skip;} \\
\ldots\ldots
\end{array}$$

Obvious!

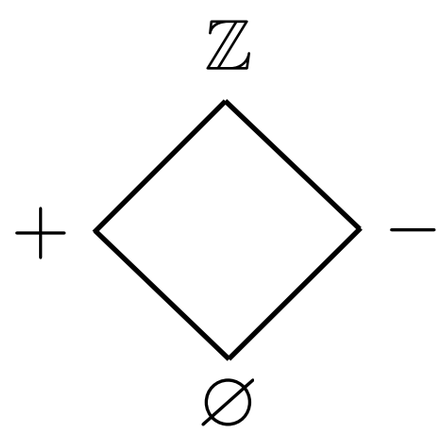
Completeness Class

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha([\![P]\!]) = [\![P]\!]^{\alpha}\}$$

Non Extensional

$$P \text{ complete}, \ [\![P]\!] = [\![Q]\!] \ \not\Rightarrow \ Q \text{ complete}$$

$$P : x := y$$
$$Q : x := y + 1;\ x := x - 1$$



Well Known!

$$[\![P]\!]^{\mathsf{Sign}}\{y/+\} = \{x/+,\ y/+\}$$
$$[\![Q]\!]^{\mathsf{Sign}}\{y/+\} = \{x/\mathbb{Z},\ y/+\}$$

Completeness Class

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha([\![P]\!]) = [\![P]\!]^{\alpha}\}$$

Non Extensional

$$P \text{ complete}, \ [\![P]\!] = [\![Q]\!] \ \not\Rightarrow \ Q \text{ complete}$$

$\mathbb{C}(\alpha)$ **and** $\overline{\mathbb{C}(\alpha)}$ **cannot be an index set**

for partial recursive functions

$$P : x := y$$
$$Q : x := y + 1; x := x - 1$$

$$[\![P]\!]^{\mathsf{Sign}}\{y/+\} = \{x/+,\ y/+\}$$
$$[\![Q]\!]^{\mathsf{Sign}}\{y/+\} = \{x/\mathbb{Z},\ y/+\}$$

Completeness Class

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha}\}$$

Non Trivial $\quad \mathbb{C}(\alpha) = \text{All Programs} \Leftrightarrow \alpha \in \{\lambda x.x,\ \lambda x.\top\}$

$\Rightarrow$ For any nontrivial abstraction α
there exists an **incomplete** program

This incomplete program is defined
similarly as into Rice Theorem's proof [1952]

Completeness Class

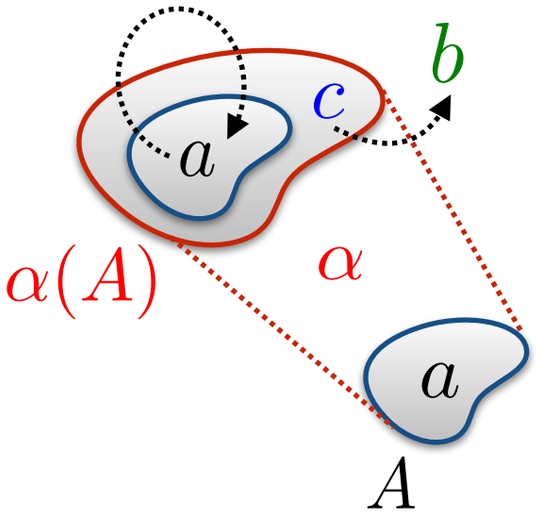
$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha([\![P]\!]) = [\![P]\!]^{\alpha}\}$$

Non Trivial

$$\mathbb{C}(\alpha) = \text{All Programs} \;\Leftrightarrow\; \alpha \in \{\lambda x.x,\ \lambda x.\top\}$$

$\Rightarrow$

α non trivial



$$\psi_{abc}(x) = \begin{cases} a & \text{if } x = a \\ b & \text{if } x = c \\ \bot & \text{otherwise} \end{cases} \qquad Q_{abc} \in \texttt{Imp} \backslash \mathbb{C}_{\alpha}$$

$$\alpha([\![Q_{abc}]\!](A)) \neq \alpha([\![Q_{abc}]\!](\alpha(A)) \subseteq [\![Q_{abc}]\!]^{\alpha}(\alpha(A))$$

$\{a\}$ $\qquad$ $\{a, b\}$

Completeness Class

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket^{\alpha}\}$$

Non Trivial + Non Extensional

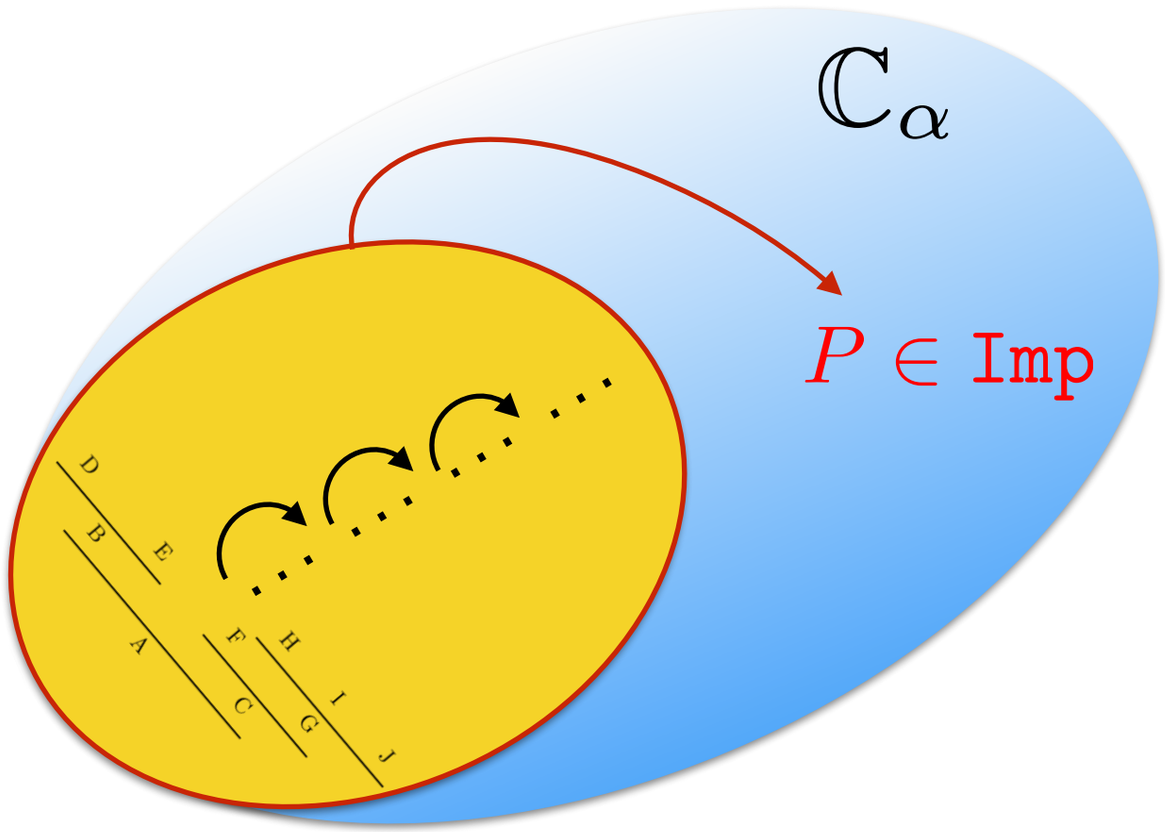
Rice Theorem **cannot** be used
for proving that $\mathbb{C}(\alpha)$ is undecidable

Completeness Class

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha([\![P]\!]) = [\![P]\!]^{\alpha}\}$$

Hard

If α *non trivial* $(\alpha \neq id \;\&\; \alpha \neq \top)$
$\mathbb{C}_\alpha$ *and* $\overline{\mathbb{C}_\alpha}$ *are productive sets*

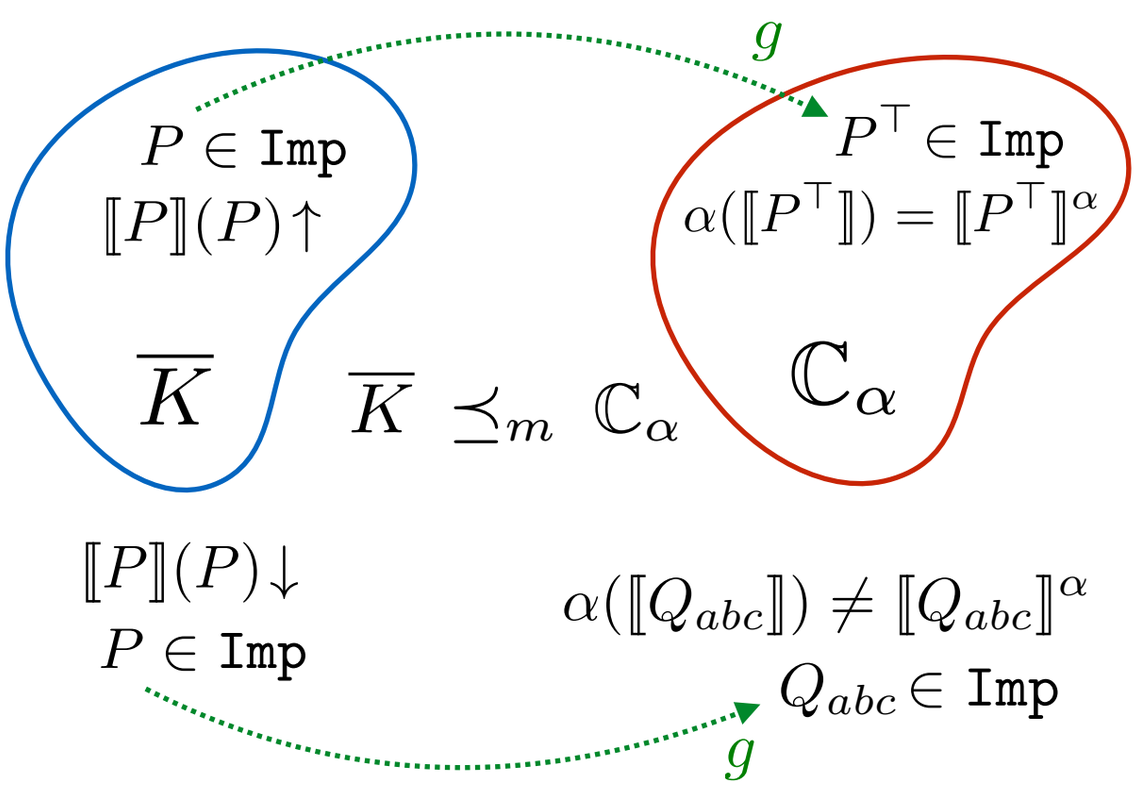


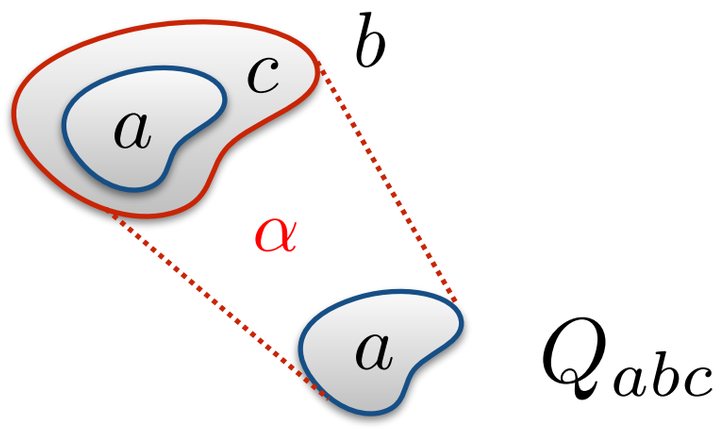
Completeness Class

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha([\![P]\!]) = [\![P]\!]^{\alpha}\}$$

Hard

If α *non trivial* $(\alpha \neq id \;\&\; \alpha \neq \top)$
$\mathbb{C}_\alpha$ *and* $\overline{\mathbb{C}_\alpha}$ *are productive sets*





$$\varphi_{g(\psi,P)} \overset{\text{s-m-n}}{=} \psi(P,S) = \begin{cases} [\![Q_{abc}]\!](S) \text{ if } [\![P]\!](P)\downarrow^{\leq |S|} \\ [\![P^\top]\!](S) \end{cases}$$

Completeness Class

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha([\![P]\!]) = [\![P]\!]^{\alpha}\}$$

Hard

If α *non trivial* $(\alpha \neq id \;\&\; \alpha \neq \top)$
$\mathbb{C}_\alpha$ *and* $\overline{\mathbb{C}_\alpha}$ *are productive sets*

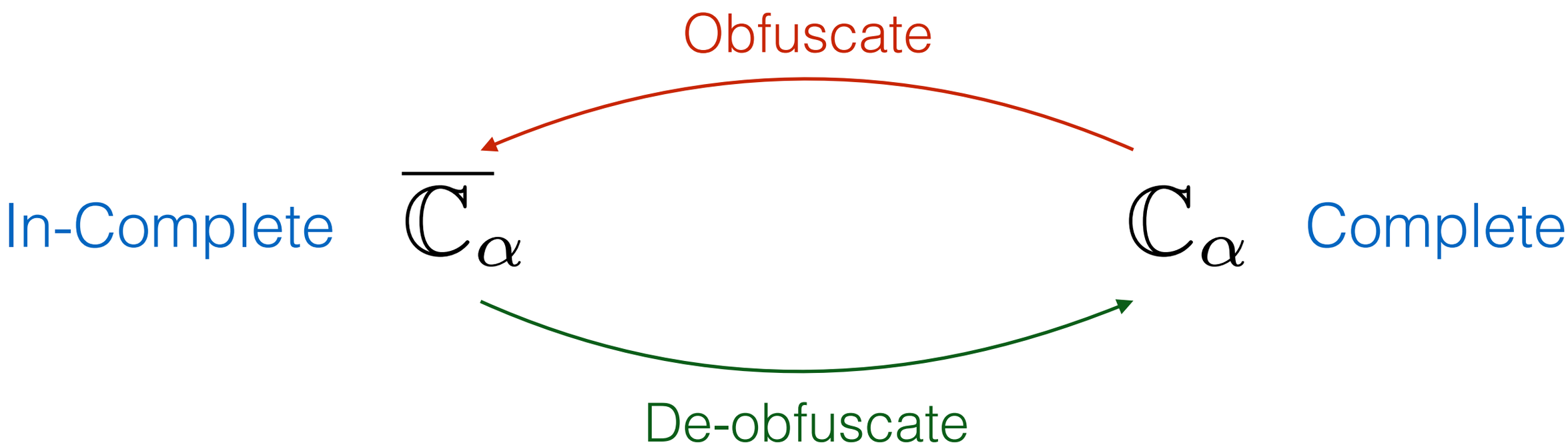
⇒ $\mathbb{C}(\alpha)$ and $\overline{\mathbb{C}(\alpha)}$ are encodings of first-order arithmetics

⇒ automating the proof that
α is complete for P is **impossible**

Completeness is harder to prove than termination

Corollary

Obfuscation/De-obfuscation is compilation between completeness classes



$$\mathbb{C}_\alpha \stackrel{def}{=} \{\ P \in \mathtt{Imp} \mid\ \alpha([\![P]\!]) = [\![P]\!]^\alpha\ \}$$

Corollary

$$\mathbb{C}(\alpha) \stackrel{\text{def}}{=} \{P \text{ program} \mid \alpha([\![P]\!]) = [\![P]\!]^{\alpha}\}$$

α non trivial

$\mathbb{C}_\alpha \not\preceq_m \overline{\mathbb{C}_\alpha}$ and $\overline{\mathbb{C}_\alpha} \not\preceq_m \mathbb{C}_\alpha$

Among equivalent programs $\preceq_m$ means deciding termination

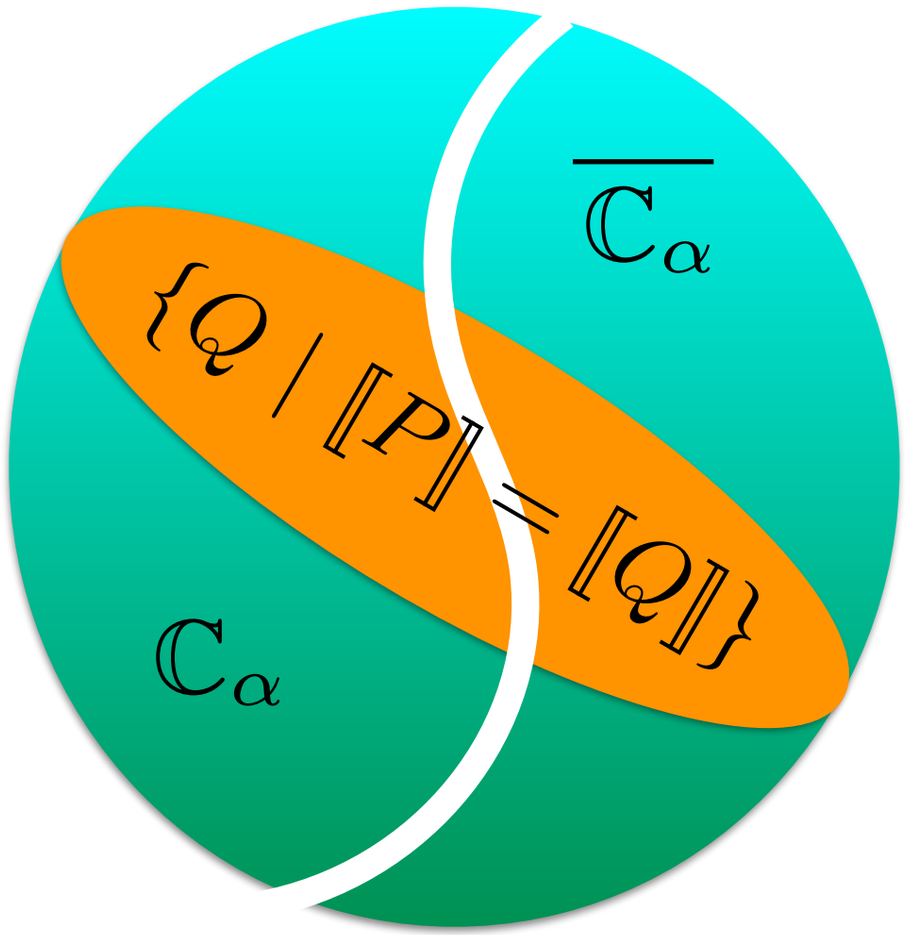
Completeness in

$\top$

|

$\bot$

All Programs



Technical Question

Can we prove that
$P \in \mathbb{C}(\alpha)$**?**

Idea: provide a reasonable computable under-approximations of $\mathbb{C}(\alpha)$

The problem

Given a program P can we prove whether an analysis of P with α will be complete?

$x := 9;$ $\quad x \in [0, 9]$

while $(x > 0)$

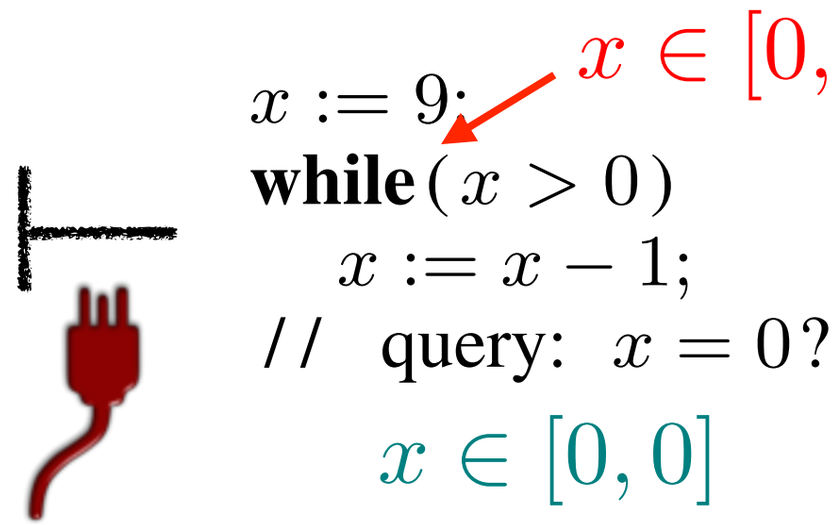
$\quad x := x - 1;$

// query: $x = 0?$

Intervals

The problem

Given a program P can we prove whether an analysis of P with α will be complete?

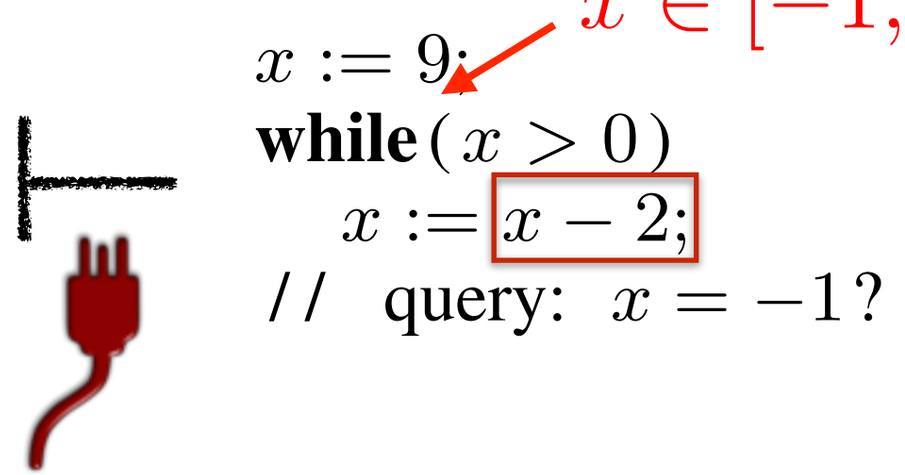


Intervals

⇨ **Completeness**

The problem

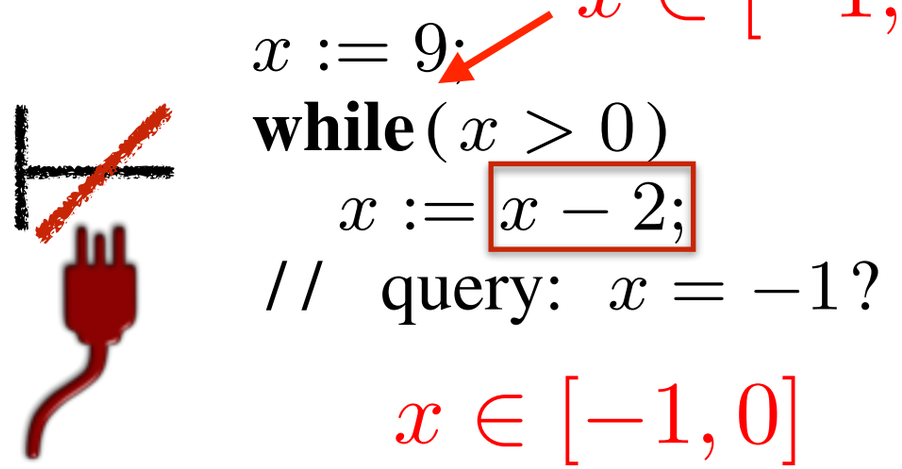
Given a program P can we prove whether an analysis of P with α will be complete?



Intervals

The problem

Given a program P can we prove whether an analysis of P with α will be complete?



Intervals

⇨ **Incompleteness**

Recall

⇒ Analyses $[\![P]\!]^\alpha$ are best correct approximations

⇒ Analyses $[\![P]\!]^\alpha$ use abstract joins **not** widening !!

⇒ Abstract joins are **always** complete in Galois Connection based analyses

$$\alpha(c_1 \sqcup c_2) =$$
$$\alpha(\gamma(\alpha(c_1)) \sqcup \gamma(\alpha(c_2))) =$$
$$\alpha(c_1) \sqcup_\alpha \alpha(c_2)$$

Complete

$x := 9;$
while $(x > 0)$
$\quad x := x - 1;$
// query: $x = 0$?

Incomplete

$x := 9;$
while $(x > 0)$
$\quad x := x - 2;$
// query: $x = -1$?

What's wrong?

- ☑ Assignment to a constant is complete in Intervals: $x \in [9, 9]$
- ☑ Both tests $x>0$ and $x\leq 0$ are exactly represented in Intervals and therefore are complete: $x \in [1, +\infty],\ x \in [-\infty, 0]$
- ☑ The decrements x-1 and x-2 are complete in Intervals
- ☑ Abstract join is always complete

The problem

Concretely

$x := 9;$
while $(x > 0)$
    $x := x - 2;$
// query: $x = -1$?

x=9
x=7
x=5
x=3
x=1
x=-1

Abstractly

$x \in [9,9]$
$x \in [7,9]$
$x \in [5,9]$
$x \in [3,9]$
$x \in [1,9]$
$x \in [-1,9]$

The problem

Concretely

$x := 9;$
while $(x > 0)$
$\quad x := x - 2;$
// query: $x = -1$?

x=9
x=7
x=5
x=3
x=1
x=-1

Abstractly

$x \in [9,9]$
$x \in [7,9]$
$x \in [5,9]$
$x \in [3,9]$
$x \in [1,9]$
$x \in [-1,9]$

$$\mathsf{Int}([\![x \leq 0]\!]\{-1, 1, ..., 9\}) = x \in [-1, -1]$$

incomplete Boolean exit

$$\mathsf{Int}([\![x \leq 0]\!]\mathsf{Int}\{-1, 1, ..., 9\}) = x \in [-1, 0]$$

The problem

$x := 9;$
while $(x > 0)$
$\quad x := x - 2;$
// query: $x = -1$?

Both tests $x{>}0$ and $x{\leq}0$ are **incomplete**
even if they are exactly represented in Intervals!!

Core Proof System: $\vdash_\alpha P$

$$\frac{}{\vdash_\alpha \textbf{skip}}\ [\textbf{skip}] \qquad \frac{\vdash_\alpha P \quad \vdash_\alpha Q}{\vdash_\alpha P; Q}\ [\textbf{seq}]$$

$$\frac{\vdash_\alpha C \quad \boxed{b \in \mathbb{B}(\alpha) \quad \neg b \in \mathbb{B}(\alpha)}}{\vdash_\alpha \textbf{if } b \textbf{ then } C}\ [\textbf{if}] \qquad \frac{\vdash_\alpha C \quad \boxed{b \in \mathbb{B}(\alpha) \quad \neg b \in \mathbb{B}(\alpha)}}{\vdash_\alpha \textbf{while } b \textbf{ do } C}\ [\textbf{while}]$$

Soundness $\boxed{\vdash_\alpha P \ \Rightarrow\ P \in \mathbb{C}(\alpha)}$

Completeness **No, …of course**

Assignments in Non-relational Domains

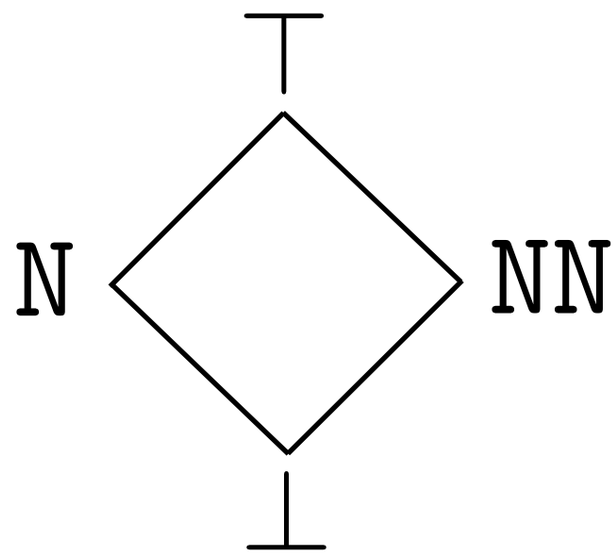
$$\frac{a \in \mathbb{A}(\alpha)}{\vdash_\alpha x := a}$$

is sound!!!

Example in Nullness

Simple **nullness** analysis

$$\mathsf{Null} \stackrel{\text{def}}{=} \{\bot, \mathtt{N}, \mathtt{NN}, \top\}$$



$$\dfrac{\dfrac{}{\mathbf{null} \in \mathbb{A}(\mathsf{Null})}\checkmark}{\vdash_{\mathsf{Null}} x := \mathbf{null}} \qquad \dfrac{(x = \mathbf{null}) \in \mathbb{B}(\mathsf{Null})\checkmark \quad \neg(x = \mathbf{null}) \in \mathbb{B}(\mathsf{Null})\checkmark \quad \dfrac{\mathbf{new}\ \mathsf{Int} \in \mathbb{A}(\mathsf{Null})\checkmark}{\vdash_{\mathsf{Null}} x := \mathbf{new}\ \mathsf{Int}}}{\vdash_{\mathsf{Null}} \mathbf{if}\ (x = \mathbf{null})\ \mathbf{then}\ x := \mathbf{new}\ \mathsf{Int}}$$

$$\vdash_{\mathsf{Null}} x := \mathbf{null};\ \mathbf{if}(x = \mathbf{null})\ \mathbf{then}\ x := \mathbf{new}\ \mathsf{Int}$$

Proved!

We don't make analysis

We analyse analyses

What about assignments?

$$\frac{a \in \mathbb{A}(\alpha)}{\vdash_\alpha x := a}$$

is not sound in general

Example in Octagons

Expression $x+y$ is **complete** in Oct

But, assignment $z := x+y$ is **not complete** in Oct

Example in Octagons

Expression $x{+}y$ is **complete** in Oct

But, assignment $z := x{+}y$ is **not complete** in Oct

$$S = \{(x/2, y/1, z/0), (x/1, y/4, z/2)\}$$

$$\downarrow$$

$$\mathsf{Oct}([\![z := x + y]\!]S) \subsetneq \mathsf{Oct}([\![z := x + y]\!]\mathsf{Oct}(S))$$

$$\not\cup \qquad\qquad \cup$$

$$(x/2, y/3, z/5) \qquad (x/2, y/3, z/5)$$

The problem is that Oct is **relational**

Example in Octagons

Expression $x{+}y$ is **complete** in Oct

But, assignment $z := x{+}y$ is **not complete** in Oct

$$S = \{(2,1,0),(1,4,2)\} \longrightarrow \mathsf{Oct}(S) = \langle 1 \le x \le 2, 1 \le y \le 4, 0 \le z \le 2,$$
$$3 \le x+y \le 5, -3 \le x-y \le 1, 2 \le x+z \le 3,$$
$$-1 \le x-z \le 2, 1 \le y+z \le 6, 1 \le y-z \le 2\rangle.$$

$$[\![z := x+y]\!]S = \{(2,1,3),(1,4,5)\}$$

$$(2,3,1) \in \mathsf{Oct}(S)$$

$$(2,3,5) \in [\![z := x+y]\!]\mathsf{Oct}(S)$$

$$\mathsf{Oct}([\![z := x+y]\!]S) = \langle 1 \le x \le 2, 1 \le y \le 4, 3 \le z \le 5,$$
$$3 \le x+y \le 5, -3 \le x-y \le 1, 5 \le x+z \le 6,$$
$$-4 \le x-z \le -1, 4 \le y+z \le 9, -2 \le y-z \le -1\rangle.$$

$$(2,3,5) \notin \mathsf{Oct}([\![z := x+y]\!]S)$$

Assignments for Octagons

Theorem: The only complete assignments for Oct are:

$$x := \pm y + k$$

$$x := \pm x + k$$

$$x := k$$

These are precisely the assignments in **Oct** with computable best correct approximations [Minè 2006]

?

Boolean guards?
Example in Intervals

$$\dfrac{\dfrac{\checkmark}{9 \in \mathbb{A}(\mathsf{Int})}}{\vdash_{\mathsf{Int}} x := 9} \qquad \dfrac{\dfrac{?}{(x > 0) \in \mathbb{B}(\mathsf{Int})} \qquad \dfrac{?}{\neg(x > 0) \in \mathbb{B}(\mathsf{Int})} \qquad \dfrac{x - 1 \in \mathbb{A}(\mathsf{Int})}{\vdash_{\mathsf{Int}} x := x - 1}}{\vdash_{\mathsf{Int}} \textbf{while } (x > 0) \textbf{ do } x := x - 1}$$

$$\vdash_{\mathsf{Int}} x := 9;\ \textbf{while}(x > 0) \textbf{ do } x := x - 1$$

Boolean guards?
Example in Intervals

Incompleteness

$$\dfrac{\dfrac{9 \in \mathbb{A}(\mathsf{Int})}{\vdash_{\mathsf{Int}} x := 9} \qquad \dfrac{\dfrac{\cancel{?}}{(x > 0) \in \mathbb{B}(\mathsf{Int})} \qquad \dfrac{\cancel{?}}{\neg(x > 0) \in \mathbb{B}(\mathsf{Int})} \qquad \dfrac{x - 1 \in \mathbb{A}(\mathsf{Int})}{\vdash_{\mathsf{Int}} x := x - 1}}{\vdash_{\mathsf{Int}} \textbf{while } (x > 0) \textbf{ do } x := x - 1}}{\vdash_{\mathsf{Int}} x := 9;\ \textbf{while}(x > 0) \textbf{ do } x := x - 1}$$

…but the analysis is **complete!**

Conditional rules for Boolean Guards

$$[\![b]\!]^{\mathbf{t}} \stackrel{\text{def}}{=} \{\rho \in \text{Store} \mid [\![b]\!]\rho = \mathbf{true}\}$$

Conditional rule

For any possible set S of input stores for a guard b:

$$\frac{assume[S : \alpha([\![b]\!]^{\mathbf{t}} \cap S) = \alpha([\![b]\!]^{\mathbf{t}}) \wedge_A \alpha(S)]}{b \in \mathbb{B}(A)}$$

Conditional Proofs

A proof of completeness for P in $\vdash_\alpha$ which depends on all the assumptions made for the Boolean guards of P

Conditional Proofs

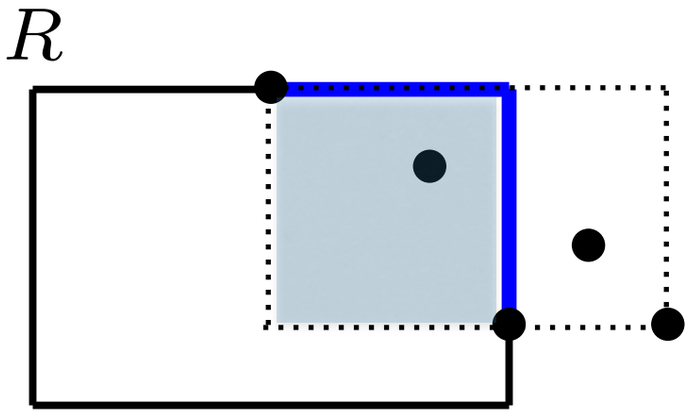
A proof of completeness for P in $\vdash_\alpha$ which depends on all the assumptions made for the Boolean guards of P

$$\dfrac{\dfrac{9 \in \mathbb{A}(\mathsf{Int})}{\vdash_{\mathsf{Int}} x := 9} \quad \dfrac{\dfrac{\mathit{assume}[S : \mathsf{Int}\{x \in S \mid x > 0\} = [1, +\infty] \sqcap_{\mathsf{Int}} \mathsf{Int}(S)]}{(x > 0) \in \mathbb{B}(\mathsf{Int})} \quad \dfrac{\dfrac{x - 1 \in \mathbb{A}(\mathsf{Int})}{\vdash_{\mathsf{Int}} x := x - 1}}{\vdash_{\mathsf{Int}} \textbf{while } (x > 0) \textbf{ do } x := x - 1}}{\vdash_{\mathsf{Int}} x := 9;\ \textbf{while}(x > 0) \textbf{ do } x := x - 1} \quad \dfrac{\dfrac{\mathit{assume}[S : \mathsf{Int}\{x \in S \mid x \leq 0\} = [-\infty, 0] \sqcap_{\mathsf{Int}} \mathsf{Int}(S)]}{\neg(x > 0) \in \mathbb{B}(\mathsf{Int})}}{\vdash_{\mathsf{Int}} \textbf{while } (x > 0) \textbf{ do } x := x - 1}$$

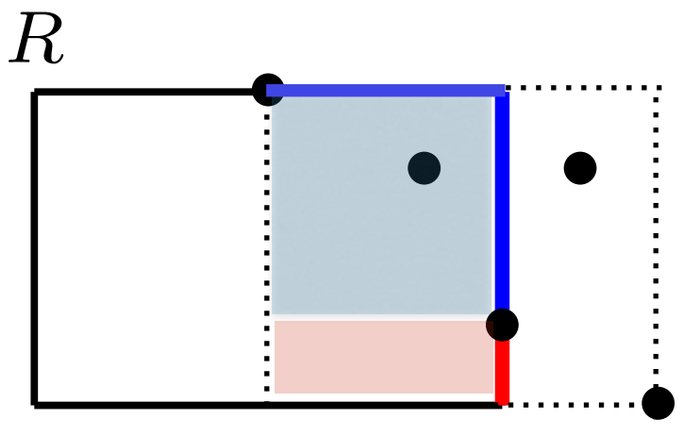
How to verify these assumptions?

Completeness of guards on Int

Two variables x,y and a Boolean guard R representable (by a rectangle) in **Int**, e.g. $k_1 \leq x \leq k_2 \wedge y > k_3$



Completeness



Incompleteness

Completeness of guards on Int

Two variables x,y and a Boolean guard R representable in **Int**, a rectangle, e.g. $k_1 \leq x \leq k_2 \wedge y > k_3$

Completeness **Incompleteness**

Theorem

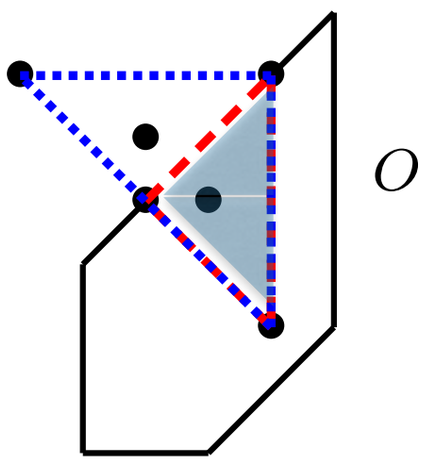
R is complete for Int in a set of stores S

$$\Longleftrightarrow$$

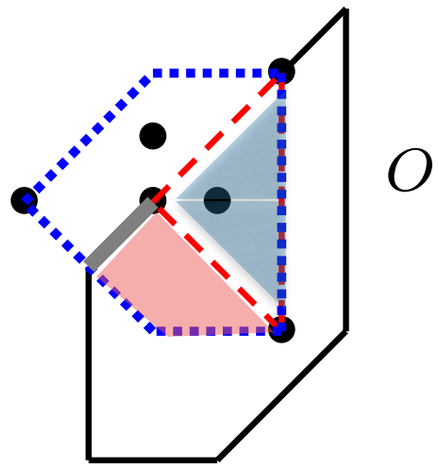
$$\forall E \in \text{edges}(R).\ \pi^{l_E}(\textsf{Int}(S) \cap R) \subseteq \textsf{Int}(\pi^{l_E}(S \cap R))$$

Completeness of guards on Oct

Two variables x,y and a Boolean guard O representable in **Oct**, an octagon, e.g. $k_1 \leq x \leq y \wedge y > k_3$



Completeness



Incompleteness

Theorem

O is complete for Oct in a set of stores S

$$\Leftrightarrow$$

$$\forall E \in \mathrm{edges}(O).\ \pi^{l_E}(\mathsf{Oct}(S) \cap O) \subseteq \mathsf{Int}(\pi^{l_E}(S \cap O))$$

Conditional Proofs of Completeness in Int

A proof of completeness for P in $\vdash_\alpha$ which depends on all the assumptions made for the Boolean guards of P

$$\dfrac{\dfrac{9 \in \mathbb{A}(\mathsf{Int})}{\vdash_{\mathsf{Int}} x := 9} \quad \dfrac{\dfrac{\textit{assume}[S : \mathsf{Int}\{x \in S \mid x > 0\} = [1, +\infty] \sqcap_{\mathsf{Int}} \mathsf{Int}(S)]}{(x > 0) \in \mathbb{B}(\mathsf{Int})} \quad \dfrac{\dfrac{x - 1 \in \mathbb{A}(\mathsf{Int})}{\vdash_{\mathsf{Int}} x := x - 1}}{\vdash_{\mathsf{Int}} \textbf{while } (x > 0) \textbf{ do } x := x - 1}}{\vdash_{\mathsf{Int}} x := 9;\ \textbf{while}(x > 0) \textbf{ do } x := x - 1} \quad \dfrac{\textit{assume}[S : \mathsf{Int}\{x \in S \mid x \leq 0\} = [-\infty, 0] \sqcap_{\mathsf{Int}} \mathsf{Int}(S)]}{\neg(x > 0) \in \mathbb{B}(\mathsf{Int})}$$

Conditional Proofs of Completeness in Int

A proof of completeness for P in $\vdash_\alpha$ which depends on all the assumptions made for the Boolean guards of P

$$\dfrac{\dfrac{9 \in \mathbb{A}(\mathsf{Int})}{\vdash_{\mathsf{Int}} x := 9} \quad \dfrac{\dfrac{\textit{assume}[S : \mathsf{Int}\{x \in S \mid x > 0\} = [1, +\infty] \sqcap_{\mathsf{Int}} \mathsf{Int}(S)]}{(x > 0) \in \mathbb{B}(\mathsf{Int})} \quad \dfrac{\dfrac{\textit{assume}[S : \mathsf{Int}\{x \in S \mid x \leq 0\} = [-\infty, 0] \sqcap_{\mathsf{Int}} \mathsf{Int}(S)]}{\neg(x > 0) \in \mathbb{B}(\mathsf{Int})} \quad \dfrac{x - 1 \in \mathbb{A}(\mathsf{Int})}{\vdash_{\mathsf{Int}} x := x - 1}}{\vdash_{\mathsf{Int}} \textbf{while } (x > 0) \textbf{ do } x := x - 1}}{\vdash_{\mathsf{Int}} x := 9;\ \textbf{while}(x > 0) \textbf{ do } x := x - 1}$$

$$S = \{9, 8, 7, ..., 0\}$$

Conditional Proofs of Completeness in Int

A proof of completeness for P in $\vdash_\alpha$ which depends on all the assumptions made for the Boolean guards of P

$$\dfrac{\dfrac{}{9 \in \mathbb{A}(\mathsf{Int})}}{\vdash_{\mathsf{Int}} x := 9} \qquad \dfrac{\dfrac{\dfrac{}{\mathit{assume}[S : \mathsf{Int}\{x \in S \mid x > 0\} = [1, +\infty] \sqcap_{\mathsf{Int}} \mathsf{Int}(S)]}}{(x > 0) \in \mathbb{B}(\mathsf{Int})} \qquad \dfrac{\dfrac{}{\mathit{assume}[S : \mathsf{Int}\{x \in S \mid x \leq 0\} = [-\infty, 0] \sqcap_{\mathsf{Int}} \mathsf{Int}(S)]}}{\neg(x > 0) \in \mathbb{B}(\mathsf{Int})} \qquad \dfrac{}{x - 1 \in \mathbb{A}(\mathsf{Int})}}{\vdash_{\mathsf{Int}} x := x - 1}}{\vdash_{\mathsf{Int}} \textbf{while } (x > 0) \textbf{ do } x := x - 1}$$

$$\vdash_{\mathsf{Int}} x := 9;\ \textbf{while}(x > 0)\ \textbf{do}\ x := x - 1$$

$$S = \{9, 8, 7, ..., 0\}$$

$\mathsf{Int}((x > 0) \cap S) = (x > 0) \cap \mathsf{Int}(S)$

$\mathsf{Int}((x \leq 0) \cap S) = (x \leq 0) \cap \mathsf{Int}(S)$

With $x{:=}x{-}1$; we don't have "**holes**" in 0 for S

Conditional Proofs of Completeness in Oct

$$x := 3; y := 0; \textbf{while}\ (x > 0)\ \textbf{do}\{x := x-1; y := y+1\}$$

complete

complete

$$\frac{\mathit{assume}[S : \mathsf{Oct}\{x \in S \mid x > 0\} = [1, +\infty] \sqcap_{\mathsf{Oct}} \mathsf{Oct}(S)]}{(x > 0) \in \mathbb{B}(\mathsf{Oct})}$$

$$\frac{\mathit{assume}[S : \mathsf{Oct}\{x \in S \mid x \leq 0\} = [-\infty, 0] \sqcap_{\mathsf{Oct}} \mathsf{Oct}(S)]}{\neg(x > 0) \in \mathbb{B}(\mathsf{Oct})}$$

Conditional Proofs of Completeness in Oct

$$x := 3; y := 0; \textbf{while}\ (x > 0)\ \textbf{do}\ \{x := x-1; y := y+1\}$$

complete

complete

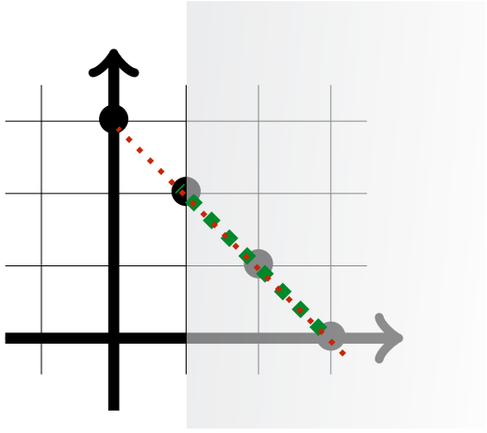
$$\frac{\mathit{assume}[S : \mathsf{Oct}\{x \in S \mid x > 0\} = [1, +\infty] \sqcap_{\mathsf{Oct}} \mathsf{Oct}(S)]}{(x > 0) \in \mathbb{B}(\mathsf{Oct})}$$

complete

$$\mathsf{Oct}((x > 0) \cap S) = (x > 0) \cap \mathsf{Oct}(S)$$

$$\frac{\mathit{assume}[S : \mathsf{Oct}\{x \in S \mid x \leq 0\} = [-\infty, 0] \sqcap_{\mathsf{Oct}} \mathsf{Oct}(S)]}{\neg(x > 0) \in \mathbb{B}(\mathsf{Oct})}$$

$$S = \{(3,0), (2,1), (1,2), (0,3)\}$$



Conditional Proofs of Completeness in Oct

$$x := 3; y := 0; \textbf{while}\ (x > 0)\ \textbf{do}\ \{x := x-1; y := y+1\}$$

complete

complete

$$\frac{\mathit{assume}[S : \mathsf{Oct}\{x \in S \mid x > 0\} = [1, +\infty] \sqcap_{\mathsf{Oct}} \mathsf{Oct}(S)]}{(x > 0) \in \mathbb{B}(\mathsf{Oct})}$$

$$\frac{\mathit{assume}[S : \mathsf{Oct}\{x \in S \mid x \leq 0\} = [-\infty, 0] \sqcap_{\mathsf{Oct}} \mathsf{Oct}(S)]}{\neg(x > 0) \in \mathbb{B}(\mathsf{Oct})}$$

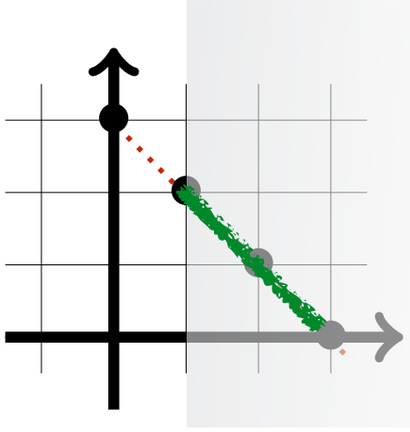
complete

$$S = \{(3,0), (2,1), (1,2), (0,3)\}$$

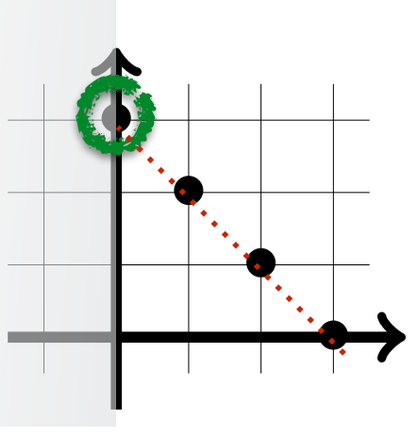
complete

$\mathsf{Oct}((x > 0) \cap S) = (x > 0) \cap \mathsf{Oct}(S)$

$\mathsf{Oct}((x \leq 0) \cap S) = (x \leq 0) \cap \mathsf{Oct}(S)$



No **holes** in 0



Conditional Proofs of Completeness in Oct

$$x := 5; y := 0; \textbf{while}\ (x > 0)\ \textbf{do}\{x := x-2; y := y+2\}$$

complete

complete

$$\frac{\mathit{assume}[S : \mathsf{Oct}\{x \in S \mid x > 0\} = [1, +\infty] \sqcap_{\mathsf{Oct}} \mathsf{Oct}(S)]}{(x > 0) \in \mathbb{B}(\mathsf{Oct})}$$

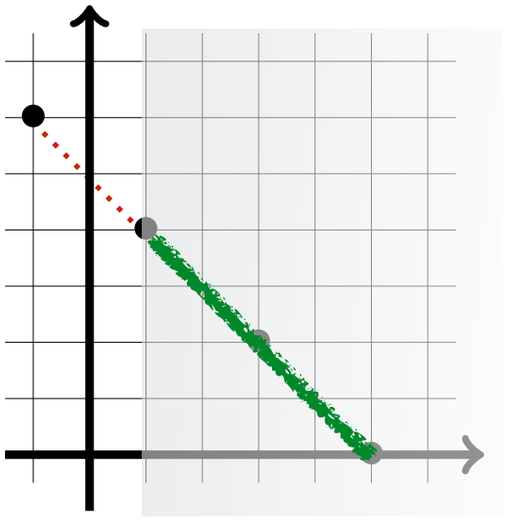
$$\frac{\mathit{assume}[S : \mathsf{Oct}\{x \in S \mid x \leq 0\} = [-\infty, 0] \sqcap_{\mathsf{Oct}} \mathsf{Oct}(S)]}{\neg(x > 0) \in \mathbb{B}(\mathsf{Oct})}$$

complete

$$S = \{(5,0), (3,2), (1,4), (-1,6)\}$$

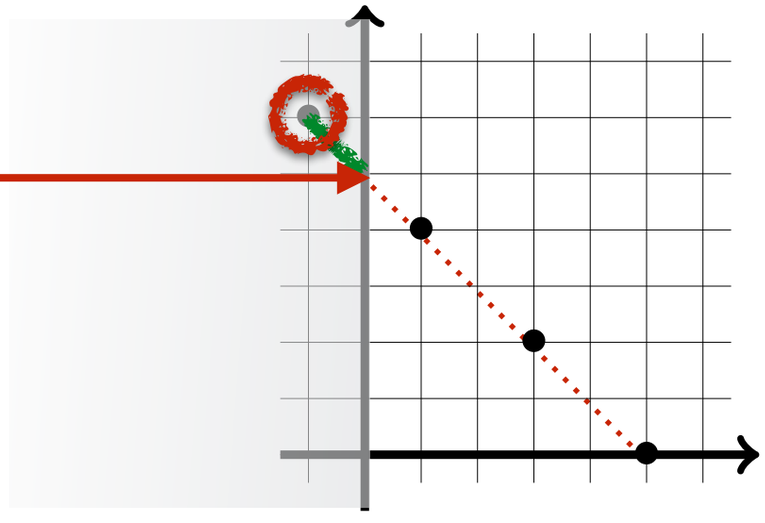
incomplete

$\mathsf{Oct}((x > 0) \cap S) = (x > 0) \cap \mathsf{Oct}(S)$

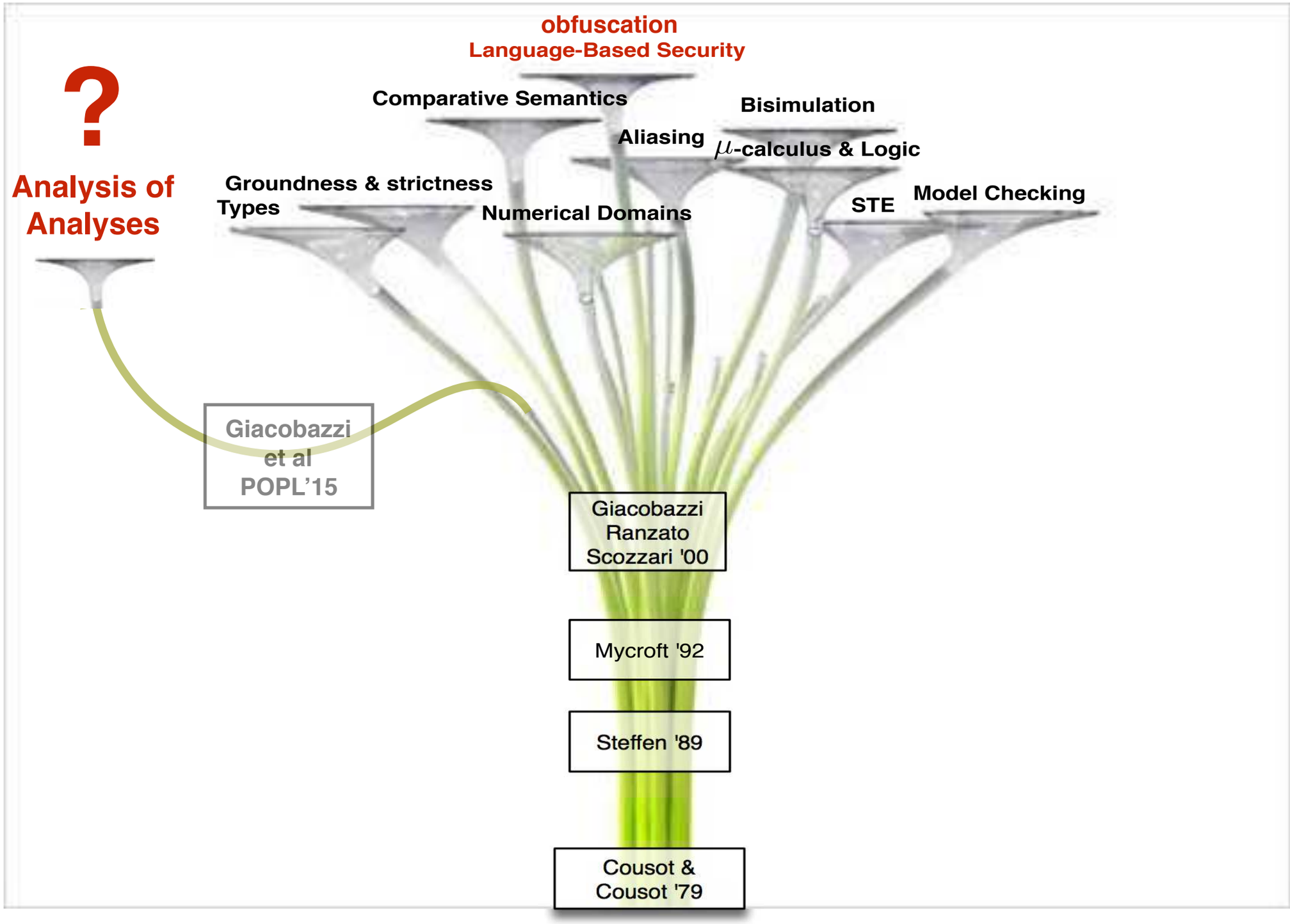


Hole in 0

$\mathsf{Oct}((x \leq 0) \cap S) \neq (x \leq 0) \cap \mathsf{Oct}(S)$



Completeness in Abstract Interpretation



Challenges

- Proving $P \in^{?} \mathbb{C}_{\alpha}$ is a **really hard** task!
- Only **guards** & **bca assignments** matter!
- Refined proofs can be obtained for **numerical abstractions**
- With **Cousot&Cousot POPL14**, this is the very first **analysis of analyses**
- Can failing proofs be used for (local) **abstract domain refinement ?**
- Can we **type analyses** by precision **?**
- Can we **refactor code to achieve completeness ?**

Thank you!


Mila


Isabella


Neil

&


Francesco


Francesco

Obfuscation & Security

Completeness