Analysing Program Analyses
A Journey in (In)Completeness

Roberto Giacobazzi

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

```plaintext
{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
{n0=n,i=0,n0>=0}
```

© Cousot
Analysis

Bug detection requires precision
A different viewpoint
Obfuscation

n := n0;

i := i - 1
Obfuscation
Obfuscation

```c
int main()
{
    int n, first = 0, second = 1, next, c;
    printf("Enter the number of terms\n");
    scanf("%d", &n);
    printf("First %d terms of Fibonacci series are :-\n", n);
    c = 0

    c < n ?
    {
        if (c <= 1)
        {
            return 0;
        }
        next = first + second;
        first = second;
        second = next;
        printf("%d\n", next);
        c++
    }
    End
}
```
Security
(White-Box Crypto Assumption)

Man At The End
Code Debugging

Safety & Security

Low

High

Analysis Precision

Low

High

Attack

Defence
Code Protection

Safety & Security vs. Analysis Precision

- **Attack**
- **Defence**

High

Low
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
What does it mean being Obscure?

How an analysis performs on code?

What does it mean being Obscure?
Obscurity is Incompleteness

\[ P : x := a \times b \]

\[ \varphi(\mathbb{Z}) \]
\[ \ldots \]
\[ \ldots \{ -5, 2, 4 \} \]
\[ \{-4, -2, 0\} \ldots \]
\[ \{-4, -2\} \ldots \{0\} \ldots \{2, 4\} \]
\[ \emptyset \]
\[ \emptyset \]
\[ \alpha = \text{Sign} \]

\[ \text{Sign}([P]) = [P]^\text{Sign} \]

Complete!
Obscurity is Incompleteness

Failing precision means failing completeness!

Obfuscating programs is making abstract interpreters incomplete.

\[ x = 0; \]

\[ P: \ x = a \ast b \quad \rightarrow\quad \tau(P): \ \text{if } b \leq 0 \text{ then } \{ a = -a; \ b = -b \}; \]
\[ \text{while } b \neq 0 \{ x = a + x; \ b = b - 1 \} \]

\textit{Sign} is complete for \( P \):
\[ \checkmark \quad [P]_{\text{Sign}} = \lambda a, b. \ \text{Sign}(a \ast b) \]

\textit{Sign} is incomplete for \( \tau(P) \):
\[ \checkmark \quad [\tau(P)]_{\text{Sign}} = \lambda a, b. \begin{cases} 0 & \text{if } a = 0 \lor b = 0 \\ ? = Z & \text{otherwise} \end{cases} \]

\textbf{Incomplete!}
Data Obfuscation

We consider variable splitting:

\[ v \in \text{Var}(P) \text{ is split into } \langle v_1, v_2 \rangle \text{ such that } \]
\[ v_1 = f_1(v), \ v_2 = f_2(v) \text{ and } v = g(v_1, v_2) \]

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

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

\[ P: \begin{cases} \quad v = 0; \\ \quad \text{while } v < N \{ v ++ \} \end{cases} \quad [P]^t = \lambda v. [0, N] \]
Data Obfuscation

We consider variable splitting:

\[ v \in \text{Var}(P) \text{ is split into } (v_1, v_2) \text{ such that} \]
\[ v_1 = f_1(v), \quad v_2 = f_2(v) \text{ and } v = g(v_1, v_2) \]
\[ f_1(v) = v \div 10 \]
\[ f_2(v) = v \mod 10 \]
\[ g(v_1, v_2) = 10 \cdot v_1 + v_2 \]

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

\[
\tau(P) : \begin{cases} 
  v_1 = 0; \\
  v_2 = 0; \\
  \text{while } 10 \cdot v_1 + v_2 < N \{ \\
  v_1 = v_1 + (v_2 + 1) \div 10 \\
  v_2 = (v_2 + 1) \mod 10 \\
  \} \\
  c : v = 10 \cdot v_1 + v_2
\end{cases}
\]

Obfuscation induces errors
Dynamic Obfuscation

```php
<?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";
?>
```

\[(N^2 + N)/2\]

---

Our Solution.

We introduce the notion of sound and complete transduction of languages we are not interested in generating signatures but rather for

```
<?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";
?>
```
Obscurity is Incompleteness!

The attacker is an abstract interpreter
Failing precision means failing completeness

Obfuscating is making abstract interpreters incomplete!!

\[ \rho \left( \llbracket P \rrbracket \right) = \llbracket P \rrbracket^\rho \]

\[ \tau \text{ obfuscates } P \text{ if } \llbracket P \rrbracket^\rho \sqsubseteq \llbracket \tau(P) \rrbracket^\rho \]

\[ \llbracket P \rrbracket^\rho = \rho(\llbracket P \rrbracket) = \rho(\llbracket \tau(P) \rrbracket) \sqsubseteq \llbracket \tau(P) \rrbracket^\rho \]
How to make code obscure

The abstraction is the specification of the attacker

- **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])
- **Decomposition**: Abstracts syntactic structures (e.g., reducible loops)
- **Disassembly**: Abstracts binary structures (e.g., recursive traversal)

\[
\alpha \]

**Programming style**

\[
\begin{align*}
\llbracket P \rrbracket &= \llbracket \text{interp}(P, d) \rrbracket \\
&= \llbracket \llbracket \llbracket \text{spec}(\text{interp}, P) \rrbracket \rrbracket \rrbracket (d)
\end{align*}
\]

**Algorithm**
How to make code obscure

The abstraction is the specification of the attacker

- **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)

Maximize incompleteness by code transformation:
- **Obfuscation**

Exploit incompleteness for hiding information:
- **Steganography**

Programming style

\[
[P] = [\text{interp}] (P, d) = [[\text{spec}] (\text{interp}, P)] (d)
\]

Algorithm

\[P \xrightarrow[\alpha]{} \text{Obf}_\alpha (P)\]

Giacobazzi et al PEPM 2012
Main Question

Can we prove that:
a static analysis $\alpha$ of a program $P$
does not raise false alarms?

or equivalently

$P$ is not obscure for $\alpha$?
The Model

Concrete Semantics
Too complex and/or undecidable
The Model

Concrete Semantics
Too complex and/or undecidable

Bad State
No bug!
False Alarms

False alarms may happen with soundness
Soundness

- Analyses are designed to be sound
  \[ \alpha(f(x)) \leq f^\#(\alpha(x)) \]
- False alarms are due to imprecision
Completeness

- Some analyses may be complete

\[ \alpha(f(x)) = f^\#(\alpha(x)) \]

- Completeness may happen!

No Imprecision
Which analysis?

\[ \alpha(f(x)) = f^\#(\alpha(x)) \]

\[ f^\# \text{ best correct approximation} \]

\[ f^\# = \alpha \circ f \circ \gamma \overset{\text{def}}{=} f^\alpha \]

is complete

\[ \alpha(f(x)) = \alpha(f(\gamma(\alpha(x)))) \]

A property of domains \( \langle \alpha, \gamma \rangle \)
Standard ways to achieve completeness in “Small Code”
Completeness

\[ \eta \circ f \circ \rho \geq \eta \circ f \]

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

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

\[ \text{sq}(X) = \left\{ x^2 \mid x \in X \right\} \]

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

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

\[
\mathcal{R}_f(\alpha) = \text{gfp}(\lambda X. \alpha \sqcap R_f(X))
\]

\[
R_f \overset{\text{def}}{=} \lambda X. \mathcal{M}(\bigcup_{y \in X} \max(f^{-1}(\downarrow y)))
\]
Completeness in Abstract Domains

Giacobazzi et al, JACM2000
We want to prove completeness

From Analysis of Programs to Analysis of Analyses
Completeness of...

\[ \alpha([a] S') = [a]^\alpha \alpha(S') \]

\[ \alpha([b] S') = [b]^\alpha \alpha(S') \]

\[ \alpha([P] S') = [P]^\alpha \alpha(S') \]

Arithmetic expressions

Boolean expressions

Programs

Best correct approximations for any set of stores \( S \)
Completeness Classes for $\alpha$

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

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

$C(\alpha) \overset{\text{def}}{=} \{ P \text{ program} \mid \alpha([P]) = [P]^\alpha \}$
Completeness Class

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

Infinite

\[
\begin{array}{c}
skip; \\
skip; skip; \\
skip; skip; skip; \\
skip; skip; skip; skip; \\
skip; skip; skip; skip; skip; \\
\ldots \ldots \\
\end{array}
\]

Obvious!
Completeness Class

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

Non Extensional

\( P \) complete, \( [P] = [Q] \not\Rightarrow Q \) complete

\[ \begin{align*}
P &: \ x := y \\
Q &: \ x := y + 1; \ x := x - 1
\end{align*} \]

\[ \begin{align*}
[P]^{\text{Sign}} \{ y/+ \} &= \{ x/+ , \ y/+ \} \\
[Q]^{\text{Sign}} \{ y/+ \} &= \{ x/\mathbb{Z} , \ y/+ \}
\end{align*} \]
Completeness Class

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

P complete, \([P] = [Q] \not\Rightarrow Q\) complete

\[ P : x := y ; x := x + 1 \]
\[ Q : x := y + 1 ; x := x - 1 \]

\[ \mathbb{C}(\alpha) \text{ and } \overline{\mathbb{C}(\alpha)} \text{ cannot be an index set for partial recursive functions} \]

\[ [P]^{\text{Sign}} \{ y/\text{+} \} = \{ x/\text{+}, y/\text{+} \} \]
\[ [Q]^{\text{Sign}} \{ y/\text{+} \} = \{ x/\mathbb{Z}, y/\text{+} \} \]
Completeness Class

\[ \mathcal{C}(\alpha) \overset{\text{def}}{=} \{ P \text{ program} \mid \alpha([P]) = [P]^\alpha \} \]

Non Trivial  \[ \mathcal{C}(\alpha) = \text{All Programs} \iff \alpha \in \{ \lambda x.x, \lambda x. \top \} \]

For any nontrivial abstraction \( \alpha \) there exists an **incomplete** program

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

\[ C(\alpha) \overset{\text{def}}{=} \{ P \text{ program} \mid \alpha([P]) = [P]^\alpha \} \]

\( C(\alpha) = \text{All Programs} \iff \alpha \in \{ \lambda x.x, \lambda x. \top \} \)

\[ \alpha(\text{non trivial}) \]

\[ \alpha(x) = \begin{cases} 
  a & \text{if } x = a \\
  b & \text{if } x = c \\
  \bot & \text{otherwise}
\end{cases} \]

\[ Q_{abc} \in \text{Imp} \setminus C_\alpha \]

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

\[ \{ a \} \neq \{ a, b \} \]
Completeness Class

\[ \mathcal{C}(\alpha) \overset{\text{def}}{=} \{ P \text{ program} \mid \alpha([P]) = [P]^{\alpha} \} \]

Rice Theorem cannot be used for proving that \( \mathcal{C}(\alpha) \) is undecidable
Completeness Class

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

If \( \alpha \) non trivial (\( \alpha \neq \text{id} \& \alpha \neq \top \))

\( \mathbb{C}_\alpha \) and \( \overline{\mathbb{C}_\alpha} \) are productive sets
Completeness Class

\[ C(\alpha) \overset{\text{def}}{=} \{ P \text{ program} \mid \alpha([P]) = [P]^{\alpha} \} \]

**Hard**

If \( \alpha \) non trivial (\( \alpha \neq \text{id} \& \alpha \neq \top \))

\( C_{\alpha} \) and \( \overline{C_{\alpha}} \) are productive sets

\[
\begin{align*}
P \in \text{Imp} & \quad [P](P) \uparrow \quad \overline{K} \quad \overline{K} \preceq_m C_{\alpha} \\
\alpha([P]) = [P]^{\alpha} & \quad \alpha([P^\top]) = [P^\top]^{\alpha} \\
Q_{abc} \in \text{Imp} & \quad \alpha([Q_{abc}]) \neq [Q_{abc}]^{\alpha}
\end{align*}
\]
**Completeness Class**

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

- **Hard**
  - If \( \alpha \) non trivial (\( \alpha \neq \text{id} \& \alpha \neq \top \))
  - \( \mathbb{C}_\alpha \) 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 \( \alpha \) is complete for \( P \) is **impossible**

Completeness is harder to prove than termination.
Corollary

Obfuscation/De-obfuscation is compilation between completeness classes

\[ \mathcal{C}_\alpha \overset{\text{def}}{=} \left\{ P \in \text{Imp} \mid \alpha([P]) = [P]^\alpha \right\} \]
Corollary

$$
\mathcal{C}(\alpha) \overset{\text{def}}{=} \{ P \text{ program} \mid \alpha([P]) = [P]^\alpha \}
$$

$\alpha$ non trivial

$$
\mathcal{C}_\alpha \not\leq_m \overline{\mathcal{C}_\alpha} \text{ and } \overline{\mathcal{C}_\alpha} \not\leq_m \mathcal{C}_\alpha
$$

Among equivalent programs $\leq_m$

means deciding termination

Completeness in

$$
\mathcal{C}_\alpha
$$

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 $\alpha$ will be complete?

\begin{align*}
  x &:= 9; & x &\in [0,9] \\
  \text{while} (x > 0) \\
  & x := x - 1; \\
  & \text{// query: } x = 0?
\end{align*}

Intervals
The problem

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

Intervals

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

while ($x > 0$)

$\ x := x - 1 ;$

// query: $x = 0 ?$

$\ x \in [0, 0]$

$\Rightarrow$ Completeness
The problem

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

$$x := 9; \quad x \in [-1, 9]$$

```
x := 9;
while (x > 0)
  x := x - 2;
// query: x = -1?
```
Given a program $P$ can we prove whether an analysis of $P$ with $\alpha$ will be complete?

- **Intervals**

  - $x := 9$.
  - $x \in [-1, 9]$
  - **while** $(x > 0)$
  - $x := x - 2$;
  - // query: $x = -1$?
  - $x \in [-1, 0]$

→ **Incompleteness**
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

\[
x := 9; \\
\text{while}(x > 0) \\
\quad x := x - 1; \\
// \text{ query: } x = 0?
\]

Incomplete

\[
x := 9; \\
\text{while}(x > 0) \\
\quad x := x - 2; \\
// \text{ query: } x = -1?
\]

**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

\[
\begin{align*}
x &:= 9; \\
\textbf{while} (x > 0) & \\
& \quad \ x := x - 2; \\
& \quad // \query: \ x = -1?
\end{align*}
\]

Abstractly

\[
\begin{align*}
x &\in [9,9] \\
x &\in [7,9] \\
x &\in [5,9] \\
x &\in [3,9] \\
x &\in [1,9] \\
x &\in [-1,9]
\end{align*}
\]
The problem

Concretely

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

Abstractly

```
x ∈ [9, 9]
x ∈ [7, 9]
x ∈ [5, 9]
x ∈ [3, 9]
x ∈ [1, 9]
x ∈ [-1, 9]
```

\[ \text{Int}(\lceil x \leq 0 \rceil \{ -1, 1, \ldots, 9 \}) = x \in [-1, -1] \]

incomplete Boolean exit

\[ \text{Int}(\lceil x \leq 0 \rceil \text{Int}\{ -1, 1, \ldots, 9 \}) = x \in [-1, 0] \]
The problem

\[\begin{align*}
  &x := 9; \\
  &\text{while} (x > 0) \\
  &\quad x := x - 2; \\
  &\quad // \text{query: } x = -1?
\end{align*}\]

Both tests \(x > 0\) and \(x \leq 0\) are **incomplete** even if they are exactly represented in Intervals!!
Core Proof System: $\vdash_{\alpha} P$

- **[skip]**
  - $\vdash_{\alpha}$ **skip**

- **[if]**
  - $\vdash_{\alpha} C \quad b \in \mathbb{B}(\alpha) \quad \neg b \in \mathbb{B}(\alpha)$
  - $\vdash_{\alpha}$ **if** $b$ **then** $C'$

- **[seq]**
  - $\vdash_{\alpha} P \quad \vdash_{\alpha} Q$
  - $\vdash_{\alpha}$ **P; Q**

- **[while]**
  - $\vdash_{\alpha} C \quad b \in \mathbb{B}(\alpha) \quad \neg b \in \mathbb{B}(\alpha)$
  - $\vdash_{\alpha}$ **while** $b$ **do** $C$

**Soundness**

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

**Completeness**

- No, …of course
Assignments in Non-relational Domains

\[ a \in \mathbb{A}(\alpha) \]

\[ \vdash_{\alpha} x := a \]

is sound!!!
Example in **Nullness**

**Simple nullness analysis**

\[ \text{Null} \equiv \{ \bot, N, NN, \top \} \]

\[ (x = \text{null}) \in \mathbb{B}(\text{Null}) \quad \neg(x = \text{null}) \in \mathbb{B}(\text{Null}) \]

\[ \text{null} \in A(\text{Null}) \]

\[ \vdash_{\text{Null}} x := \text{null} \]

\[ \vdash_{\text{Null}} \text{if } (x = \text{null}) \text{ then } x := \text{new Int} \]

\[ \vdash_{\text{Null}} x := \text{null}; \text{ if}(x = \text{null}) \text{ then } x := \text{new Int} \]

**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.

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

$$\text{Oct}(\lceil z := x + y \rceil S) \subsetneq \text{Oct}(\lceil z := x + y \rceil \text{Oct}(S)).$$

$\cup$

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

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

\[
S = \{(2,1,0), (1,4,2)\}
\]

\[
\text{Oct}(S) = \{1 \leq x \leq 2, 1 \leq y \leq 4, 0 \leq z \leq 2, \\
3 \leq x+y \leq 5, -3 \leq x-y \leq 1, 2 \leq x+z \leq 3, \\
-1 \leq x-z \leq 2, 1 \leq y+z \leq 6, 1 \leq y-z \leq 2\}.
\]

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

\[
\text{Oct}([z := x + y]_S) = \{1 \leq x \leq 2, 1 \leq y \leq 4, 3 \leq z \leq 5, \\
3 \leq x+y \leq 5, -3 \leq x-y \leq 1, 5 \leq x+z \leq 6, \\
-4 \leq x-z \leq -1, 4 \leq y+z \leq 9, -2 \leq y-z \leq -1\}.
\]

\[
(2,3,1) \in \text{Oct}(S)
\]

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

\[
(2,3,5) \notin \text{Oct}([z := x + y]_S)
\]
Assignments for Octagons

**Theorem:** The only complete assignments for Oct are:

\[
\begin{align*}
x & := \pm y + k \\
x & := \pm x + k \\
x & := k
\end{align*}
\]

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

Example in Intervals

\[
\begin{align*}
9 \in A(\text{Int}) \\
\vdash_{\text{Int}} x := 9
\end{align*}
\]

\[
\begin{align*}
(x > 0) \in B(\text{Int}) \\
\neg(x > 0) \in B(\text{Int})
\end{align*}
\]

\[
\begin{align*}
x - 1 \in A(\text{Int}) \\
\vdash_{\text{Int}} x := x - 1
\end{align*}
\]

\[
\begin{align*}
\vdash_{\text{Int}} x := 9; \ while(x > 0) \ do \ x := x - 1
\end{align*}
\]
Boolean guards?

Example in Intervals

Incompleteness

\[
\begin{align*}
9 \in \mathbb{A}(\text{Int}) & \quad \text{?} \quad \text{?} \quad x - 1 \in \mathbb{A}(\text{Int}) \\
\vdash_{\text{Int}} x := 9 & \quad \vdash_{\text{Int}} (x > 0) \in \mathbb{B}(\text{Int}) \quad \vdash_{\text{Int}} \neg (x > 0) \in \mathbb{B}(\text{Int}) \quad \vdash_{\text{Int}} x := x - 1 \\
& \quad \vdash_{\text{Int}} \text{while } (x > 0) \text{ do } x := x - 1 \\
& \quad \vdash_{\text{Int}} x := 9; \text{ while}(x > 0) \text{ do } x := x - 1
\end{align*}
\]

…but the analysis is complete!
Conditional rules for Boolean Guards

\[ [b]^t \overset{\text{def}}{=} \{ \rho \in \text{Store} \mid [b]_\rho = \text{true} \} \]

**Conditional rule**

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

\[
\text{assume}[S : \alpha([b]^t \cap S) = \alpha([b]^t) \land_A \alpha(S)]
\]

\( b \in \mathbb{B}(A) \)
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 \( P \)

\[
\begin{align*}
9 & \in A(\text{Int}) \\
\vdash \text{Int} \ x := 9
\end{align*}
\]

\[
\begin{align*}
\text{assume}[S : \text{Int}\{x \in S \mid x > 0\} = [1, +\infty] \cap \text{Int}(S)] & \quad (x > 0) \in \mathbb{B}(\text{Int}) \\
\text{assume}[S : \text{Int}\{x \in S \mid x \leq 0\} = [-\infty, 0] \cap \text{Int}(S)] & \quad \neg(x > 0) \in \mathbb{B}(\text{Int}) \\
x - 1 & \in A(\text{Int}) \\
\vdash \text{Int} \ x := x - 1
\end{align*}
\]

\[
\begin{align*}
\vdash \text{Int} \ while \ (x > 0) \ do \ x := x - 1 \\
\vdash \text{Int} \ x := 9; \ while(x > 0) \ do \ x := x - 1
\end{align*}
\]

How to verify these assumptions?
Completeness of guards on $\text{Int}$

Two variables $x,y$ and a Boolean guard $R$ representable (by a rectangle) in $\text{Int}$, e.g. $k_1 \leq x \leq k_2 \land y > k_3$
Completeness of guards on \( \text{Int} \)

Two variables \( x, y \) and a Boolean guard \( R \) representable in \( \text{Int} \), a rectangle, e.g. \( k_1 \leq x \leq k_2 \land y > k_3 \)

**Theorem**

\( R \) is complete for \( \text{Int} \) in a set of stores \( S \)

\[
\forall E \in \text{edges}(R). \pi^E (\text{Int}(S) \cap R) \subseteq \text{Int}(\pi^E (S \cap R))
\]
Completeness of guards on \( \text{Oct} \)

Two variables \( x, y \) and a Boolean guard \( O \) representable in \( \text{Oct} \), an octagon, e.g. \( k_1 \leq x \leq y \land y > k_3 \)

\[ \forall E \in \text{edges}(O). \pi^E (\text{Oct}(S) \cap O) \subseteq \text{Int}(\pi^E (S \cap O)) \]
Conditional Proofs of Completeness in \texttt{Int}

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

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$9 \in \mathcal{A}(\texttt{Int})$</td>
<td>$\vdash_{\texttt{Int}} x := 9$</td>
<td>$x - 1 \in \mathcal{A}(\texttt{Int})$</td>
</tr>
<tr>
<td>$\vdash_{\texttt{Int}} (x &gt; 0) \in \mathcal{B}(\texttt{Int})$</td>
<td>$\vdash_{\texttt{Int}} \textbf{while} (x &gt; 0) \textbf{ do } x := x - 1$</td>
<td>$\vdash_{\texttt{Int}} x := 9; \textbf{ while } (x &gt; 0) \textbf{ do } x := x - 1$</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\text{assume}[S: \texttt{Int}\{x \in S \mid x > 0\} = [1, +\infty] \cap \texttt{Int}(S)] & \quad \text{assume}[S: \texttt{Int}\{x \in S \mid x \leq 0\} = [-\infty, 0] \cap \texttt{Int}(S)] \\
(x > 0) \in \mathcal{B}(\texttt{Int}) & \quad \neg(x > 0) \in \mathcal{B}(\texttt{Int})
\end{align*} \]
A proof of completeness for $P$ in $\vdash \alpha$ which depends on all the assumptions made for the Boolean guards of $P$

\[
\begin{align*}
\text{assume}[S : \text{Int}\{x \in S \mid x > 0\} = [1, +\infty] \cap \text{Int}(S)] & \quad (x > 0) \in \mathbb{B}(\text{Int}) & \quad \text{assume}[S : \text{Int}\{x \in S \mid x \leq 0\} = [-\infty, 0] \cap \text{Int}(S)] & \quad \neg(x > 0) \in \mathbb{B}(\text{Int}) & \quad x - 1 \in \mathbb{A}(\text{Int}) \\
\vdash_{\text{Int}} x := 9 & \quad \vdash_{\text{Int}} \text{while } (x > 0) \text{ do } x := x - 1 & \quad \vdash_{\text{Int}} x := 9; \text{ while}(x > 0) \text{ do } x := x - 1
\end{align*}
\]

\[
S' = \{9, 8, 7, \ldots, 0\}
\]
Conditional Proofs of Completeness in $\text{Int}$

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

$\begin{array}{c}
9 \in A(\text{Int}) \\
\vdash_{\text{Int}} x := 9
\end{array}$

$\begin{array}{c}
\text{assume}[S : \text{Int}\{x \in S \mid x > 0\} = [1, +\infty] \cap \text{Int}(S)] \\
(x > 0) \in B(\text{Int})
\end{array}$

$\begin{array}{c}
\vdash_{\text{Int}} \text{while}(x > 0) \text{ do } x := x - 1
\end{array}$

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

$\begin{array}{c}
\text{Int}((x > 0) \cap S) = (x > 0) \cap \text{Int}(S)
\end{array}$

$\begin{array}{c}
\text{Int}((x \leq 0) \cap S) = (x \leq 0) \cap \text{Int}(S)
\end{array}$

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

\[
\text{assume}[S : \text{Oct}\{x \in S \mid x > 0\} = [1, +\infty] \cap_{\text{Oct}} \text{Oct}(S)]
\]

\( (x > 0) \in \mathbb{B}(\text{Oct}) \)

complete

\[
\text{assume}[S : \text{Oct}\{x \in S \mid x \leq 0\} = [-\infty, 0] \cap_{\text{Oct}} \text{Oct}(S)]
\]

\( \neg(x > 0) \in \mathbb{B}(\text{Oct}) \)
Conditional Proofs of Completeness in $\text{Oct}$

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

\begin{align*}
\text{assume}[S : \text{Oct}\{x \in S \mid x > 0\} &= [1, +\infty] \cap \text{Oct}(S)] \\
(x > 0) &\in \mathbb{B}(\text{Oct})
\end{align*}

\begin{align*}
\text{assume}[S : \text{Oct}\{x \in S \mid x \leq 0\} &= [-\infty, 0] \cap \text{Oct}(S)] \\
\neg(x > 0) &\in \mathbb{B}(\text{Oct})
\end{align*}

\text{complete}

\begin{align*}
S &= \{(3, 0), (2, 1), (1, 2), (0, 3)\}
\end{align*}

$\text{Oct}(\{x > 0\} \cap S) = \{x > 0\} \cap \text{Oct}(S)$
Conditional Proofs of Completeness in Oct

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

complete

\[ \text{assume}[S : \text{Oct}\{x \in S \mid x > 0\} = [1, \infty] \cap_{\text{Oct}} \text{Oct}(S)] \]

\((x > 0) \in \mathbb{B}(\text{Oct})\)

complete

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

complete

\[ \text{Oct}((x > 0) \cap S) = (x > 0) \cap \text{Oct}(S) \]

No holes in 0

\[ \text{Oct}((x \leq 0) \cap S) = (x \leq 0) \cap \text{Oct}(S) \]
Conditional Proofs of Completeness in Oct

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

- complete
- \( \text{assume}[S : \text{Oct}\{x \in S \mid x > 0\} = [1, +\infty) \cap_{\text{Oct}} \text{Oct}(S)] \)
- \((x > 0) \in \mathbb{B}(\text{Oct})\)

- complete
- \( \text{assume}[S : \text{Oct}\{x \in S \mid x \leq 0\} = [-\infty, 0) \cap_{\text{Oct}} \text{Oct}(S)] \)
- \(\neg(x > 0) \in \mathbb{B}(\text{Oct})\)

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

- incomplete
- \( \text{Oct}((x > 0) \cap S) = (x > 0) \cap \text{Oct}(S) \)
- \( \text{Oct}((x \leq 0) \cap S) \not\equiv (x \leq 0) \cap \text{Oct}(S) \)

Hole in 0
Completeness in Abstract Interpretation

? Analysis of Analyses

Giacobazzi et al POPL'15

Giacobazzi Ranzato Scozzari '00

Mycroft '92

Steffen '89

Cousot & Cousot '79
Challenges

- Proving $P \in ?C_\alpha$ 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

Obfuscation & Security

Francesco

Completeness