Analysing (Completeness in) Program Analyses

Roberto Giacobazzi
U.Verona

Francesco Logozzo
Facebook Seattle

Francesco Ranzato
U.Padova
Main Question

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

\[ x(t) \]

\[ t \]
No Bugs

$\mathbf{x}(t)$

Bad State
Sound Analysis

Bad State

$x(t)$
Analyses raise False Alarms

False alarms due to sound approximations
• Analyses are designed to be **sound**

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

• False alarms due to **imprecision**
Zero False Alarms

$\x(t)$

Bad State

Completeness $\Rightarrow$ NO false alarm
• Analyses may be **complete**

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

• Completeness **may happen**

**Completeness**

No Imprecision
Which analysis?

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

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

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

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

Property of Domains
To get completeness (or improve precision)...

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

...one can refine domains

Instead we want to prove completeness
Completeness of...

$$\alpha(\llbracket a \rrbracket S') = \llbracket a \rrbracket^\alpha \alpha(S')$$

$$\text{arithmetic expressions}$$

$$\alpha(\llbracket b \rrbracket S') = \llbracket b \rrbracket^\alpha \alpha(S')$$

$$\text{Boolean expressions}$$

$$\alpha(\llbracket P \rrbracket S') = \llbracket P \rrbracket^\alpha \alpha(S')$$

$$\text{programs}$$

Best correct approximations

For any set of stores $S$
The Abstraction $\alpha$ is Fixed
Completeness Class for $\alpha$

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

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

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

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

We study this object

\( C_\alpha \)

Programs

Incomplete programs

\( \overline{C_\alpha} \)
Completeness Class

\[ \mathcal{C}(\alpha) \overset{\text{def}}{=} \{ P \text{ program} \mid \alpha([P]) = [P]^\alpha \} \]
Corresponding classes of completeness are not related. In particular, the following two programs:

The lifting of is known in static analysis where semantics-preserving program complete commands is still complete. It is also non-extensional, and generic expressions are:

Moreover, by a straightforward padding argument, any of these programs (e.g.,) can be extended with the following partial recursive function that coarse abstractions may induce a complete static analysis for some program where more precise ones instead fail for that same given question on their behavior. We therefore introduce the notion of all programs whose static analysis on a given abstraction is complete for the program properties represented by the set of programs that are complete for the program properties reciprocal to reason on the precision of an abstraction.

This incomplete program is defined similarly to Rice Theorem's proof.

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

This incomplete program is defined similarly to Rice Theorem's proof.

**Non Trivial**

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

\[ \psi_{b,c} : \text{Store} \rightarrow \text{Store} \]

\[ \psi_{bc}(x) \triangleq \begin{cases} 
  x & \text{if } x \in S \\
  b & \text{if } x = c \\
  \text{undefined} & \text{otherwise}
\end{cases} \]
Completeness Class

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

Non Extensional

\( P \) complete, \( [P] = [Q] \nRightarrow \) \( Q \) complete

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

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

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

Non Trivial + Non Extensional

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

Undecidable? Semi-decidable (r.e.)? Not r.e.?
Completeness Class

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

If \( \alpha \) is nontrivial then \( \mathbb{C}(\alpha) \) and \( \overline{\mathbb{C}(\alpha)} \) are productive sets

\( \mathbb{C}(\alpha) \) and \( \overline{\mathbb{C}(\alpha)} \) are not r.e., i.e. are encodings of first-order arithmetics

automating the proof that \( \alpha \) is complete for \( P \) is impossible

Completeness is harder to prove than termination
Technical Question

Can we prove that

\[ P \in \mathbb{C}(\alpha) \]?

We provide 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?
Numerical Abstractions

Intervals

Octagons
The problem

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

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

\[
\text{while}(x > 0)
\]

\[
x := x - 1;
\]

// query: \( x = 0? \)

\[
x \in [0, 0]
\]

Intervals

\[\implies\] Completeness
The problem

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

---

$x := 9$; \hspace{1cm} $x \in [-1, 9]$

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

$x \in [-1, 0]$

---

Intervals $\implies$ Incompleteness
Recall: analyses $\lbrack P \rbrack^\alpha$ are best correct approximations

Analyses $\lbrack P \rbrack^\alpha$ use abstract joins not widenings

Abstract joins are always complete in Galois connection based analyses

Incompleteness is never due to abstract joins
**Complete**

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

**Incomplete**

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

---

**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**

\[
\begin{align*}
  x &= 9 \\
  x &= 7 \\
  x &= 5 \\
  x &= 3 \\
  x &= 1 \\
  x &= -1
\end{align*}
\]

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

**Abstractly**

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

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

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

```
x := 9;
while (x > 0)
  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 \( \vdash \alpha P \)

\[
\begin{array}{c}
\vdash_{\alpha} \mathsf{skip} \quad [\mathsf{skip}] \\
\vdash_{\alpha} \mathsf{if} \mathsf{b} \mathsf{then} C \quad [\mathsf{if}] \\
\vdash_{\alpha} C \quad \mathsf{b} \in \mathbb{B}(\alpha) \quad \mathsf{\neg} b \in \mathbb{B}(\alpha) \\
\vdash_{\alpha} \mathsf{while} \mathsf{b} \mathsf{do} C \quad [\mathsf{while}] \\
\end{array}
\]

\[
\begin{array}{c}
\vdash_{\alpha} P \quad \vdash_{\alpha} Q \quad [\mathsf{seq}] \\
\vdash_{\alpha} P ; Q \\
\end{array}
\]
Core proof system $\vdash^\alpha P$

Soundness

\[
\vdash^\alpha P \Rightarrow \text{seq } P \in \mathbb{C}(\alpha)
\]

Completeness

No, of course
What about assignments?

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

is not sound
Example in Octagons

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

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

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

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

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

The problem is that Oct is **relational**
Assignments in Nonrelational Domains

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

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

is sound
We argued that the main problem in proving completeness of guards is characterized by Lemma 5.7. This may suggest an interesting analysis focused on Boolean guards. Let $P$ be a Boolean predicate in a further distinct step.

Hence, a conditional completeness proof of guards of Boolean guards of $P$ is complete for any set of possible input states for a Boolean guard $G$, whose assumptions on the guard $G$ are made in a conditional proof of $P$. We can safely conclude that the remaining assignments are the only which guarantee that the sets of possible input states for a Boolean guard of $P$ are as follows:

$$\text{Example in Nullness}$$

**Simple nullness analysis**

$$\text{Null} \overset{\text{def}}{=} \{ \bot, N, NN, \top \}$$

$$\begin{align*}
null &\in A(\text{Null}) \\
(x = \text{null}) &\in B(\text{Null}) \\
\neg(x = \text{null}) &\in B(\text{Null}) \\
\text{new Int} &\in A(\text{Null}) \\
\vdash_{\text{Null}} x := \text{null} \\
\vdash_{\text{Null}} \text{if}(x = \text{null}) \text{ then } x := \text{new Int} \\
\vdash_{\text{Null}} x := \text{null}; \text{if}(x = \text{null}) \text{ then } x := \text{new Int}
\end{align*}$$

Proved!
Assignments in Relational Domains

Nothing can be done in general

Need for domain-specific analyses of completeness of assignments
Assignments for Octagons

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

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

Example in Intervals

\[
\begin{align*}
9 \in A(\text{Int}) & \quad \Rightarrow \quad \Box_{\text{Int}} x := 9 \\
(x > 0) \in B(\text{Int}) & \quad \Rightarrow \quad \Box_{\text{Int}} \text{ while } (x > 0) \text{ do } x := x - 1 \\
\neg(x > 0) \in B(\text{Int}) & \quad \Rightarrow \quad \Box_{\text{Int}} x := x - 1 \\
x - 1 \in A(\text{Int}) & \quad \Rightarrow \quad \Box_{\text{Int}} x := x - 1 \\
\end{align*}
\]
Back to Boolean guards

Example in *Intervals*

**Incompleteness**

\[
\begin{align*}
9 & \in \mathbb{A}(\text{Int}) \\
\vdash_{\text{Int}} & x := 9 \\
\frac{\neg(x > 0) \in \mathbb{B}(\text{Int})}{\neg \Gamma} & \quad \text{(no guarantee)}
\end{align*}
\]

\[
\begin{align*}
\frac{\neg(x > 0) \in \mathbb{B}(\text{Int})}{\neg \Gamma} & \quad \text{(no guarantee)}
\end{align*}
\]

\[
\begin{align*}
x - 1 & \in \mathbb{A}(\text{Int}) \\
\vdash_{\text{Int}} & x := x - 1 \\
\frac{x - 1 > 0 \in \mathbb{B}(\text{Int})}{x - 1 \in \mathbb{A}(\text{Int})} & \quad \text{(no guarantee)}
\end{align*}
\]

\[
\begin{align*}
x - 1 & \in \mathbb{A}(\text{Int}) \\
\vdash_{\text{Int}} & x := x - 1 \\
\frac{x - 1 > 0 \in \mathbb{B}(\text{Int})}{x - 1 \in \mathbb{A}(\text{Int})} & \quad \text{(no guarantee)}
\end{align*}
\]

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

...but the program is complete!
Conditional rule 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 \( b \):

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

\[
b \in \mathcal{B}(A)
\]
Conditional Proofs of Completeness

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

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

\[ \text{Completeness} \quad \text{Incompleteness} \]

**Theorem**

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

\[ \forall E \in \text{edges}(R). \pi^E (\text{Int}(S) \cap R) \subseteq \text{Int}(\pi^E (S \cap R)) \]
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 $P$

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

\[
S = \{9, 8, 7, \ldots, 0\}
\]

\[
\text{Int}((x > 0) \cap S) = (x > 0) \cap \text{Int}(S) \quad \text{Int}((x \leq 0) \cap S) = (x \leq 0) \cap \text{Int}(S)
\]

With $x := x - 1$; we don't have "holes" in 0 for $S$
Completeness of guards on \textit{Oct}

\textbf{Completeness}

\textbf{Incompleteness}

\textit{Oct}

\textit{Int}

Similar to \textit{Int}
Conclusions

- Completeness class of a given abstraction
- A proof system for proving completeness
- Need of specific proofs for assignments on numerical abstractions
- Proofs typically fail for Boolean guards, seldom complete
- Conditional proofs for Boolean guards

Future Work

Even analyses with widening may be complete

Proving completeness of widenings wrt abstract (not concrete) joins