Local completeness for program correctness and incorrectness

Roberto (Verona)

Roberta (Pisa)

Francesco (Padova)

Roberto (Pisa)

CALCO 2023

Motivation and Main Result







Over- and Under-approximations in program analysis

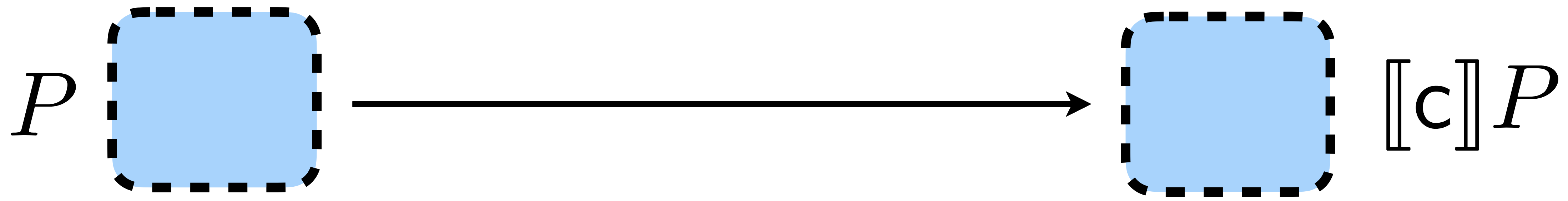
CALCO 2023
MFPS XXXIX

What is the contribution about?

A logic to prove the absence
as well as the presence of bugs

how seemingly opposite concepts may actually be complementary, interconnected, and interdependent

Over- vs Under-approximations

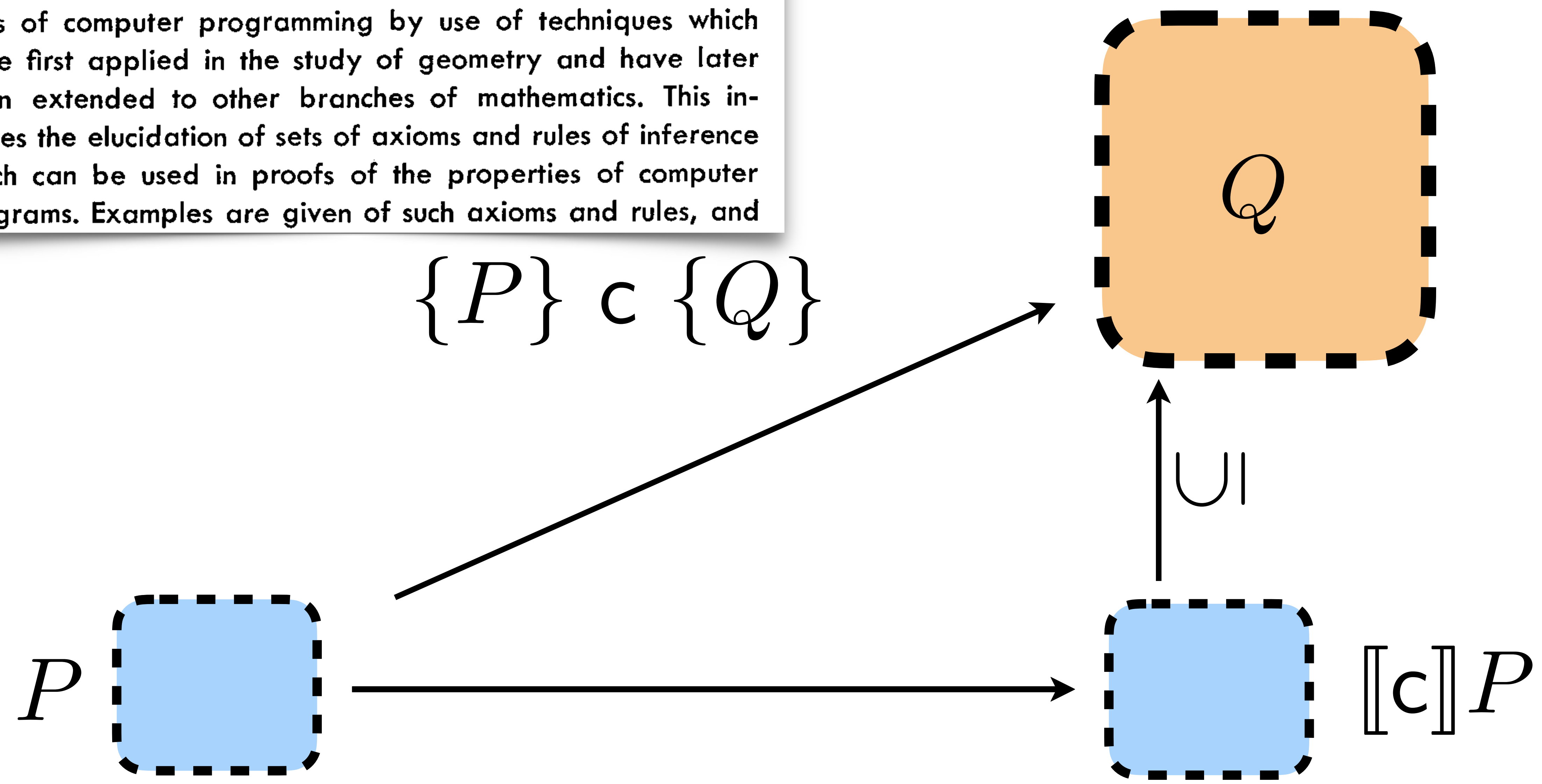


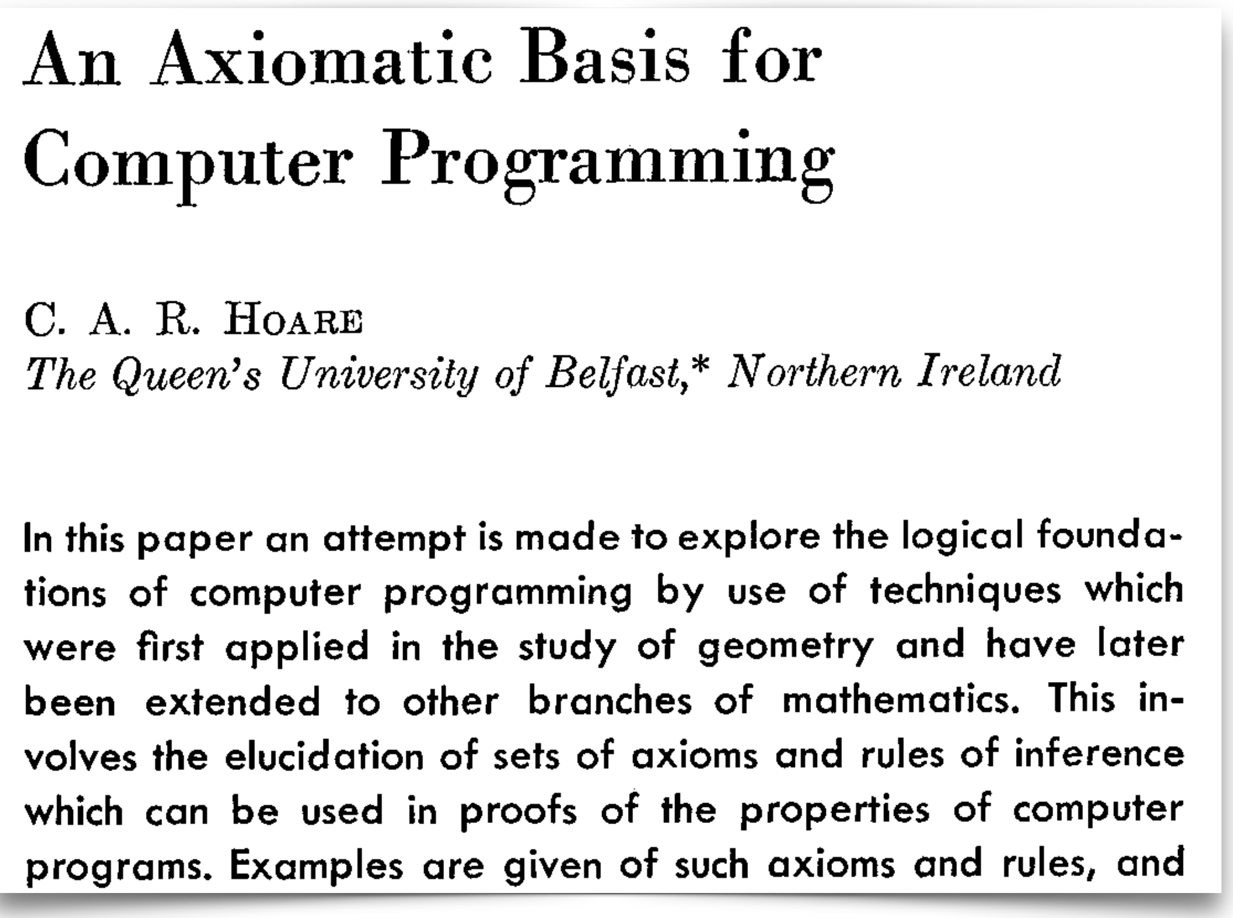
Over- vs Under-approximations

An Axiomatic Basis for Computer Programming

C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and

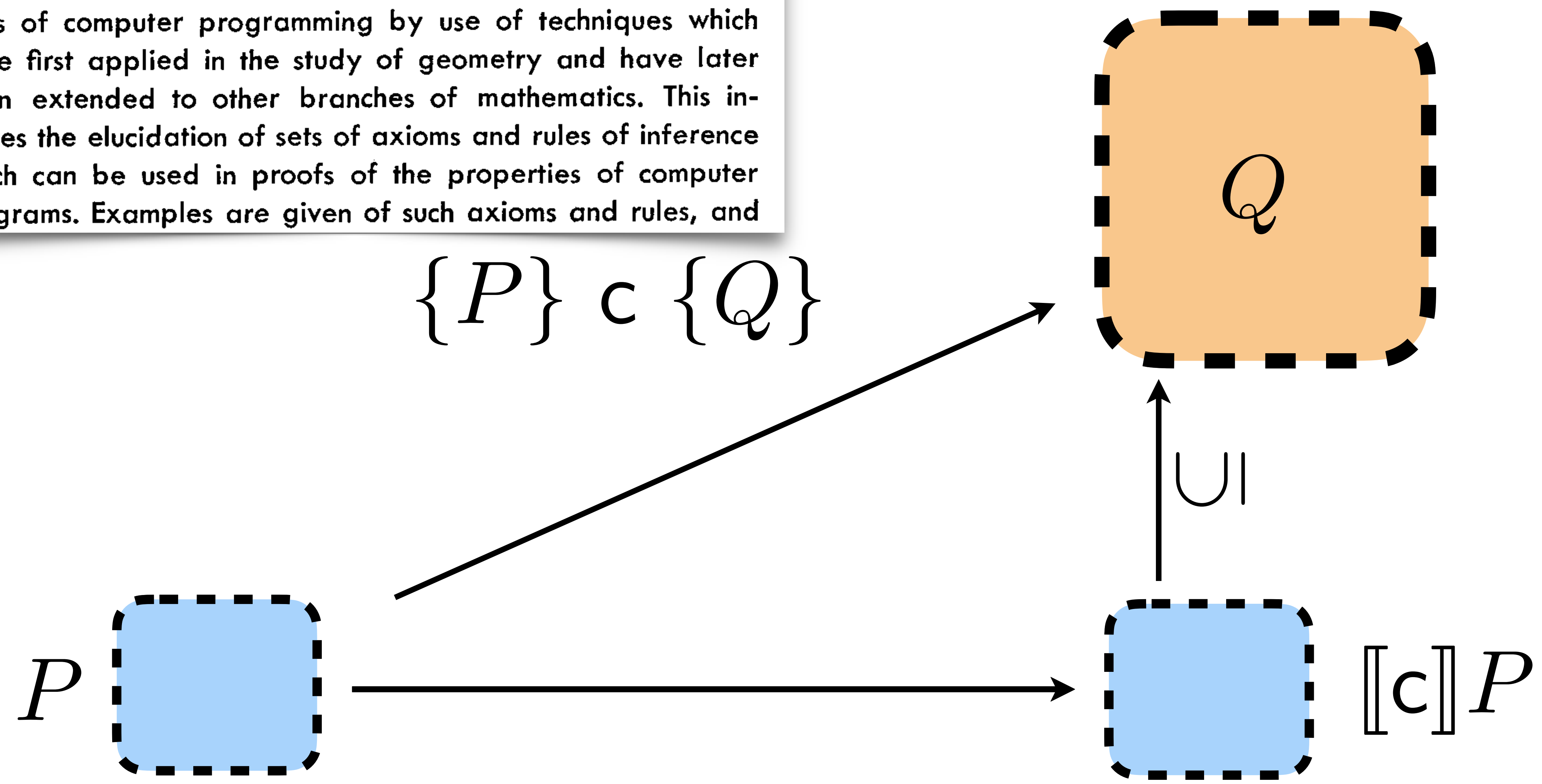


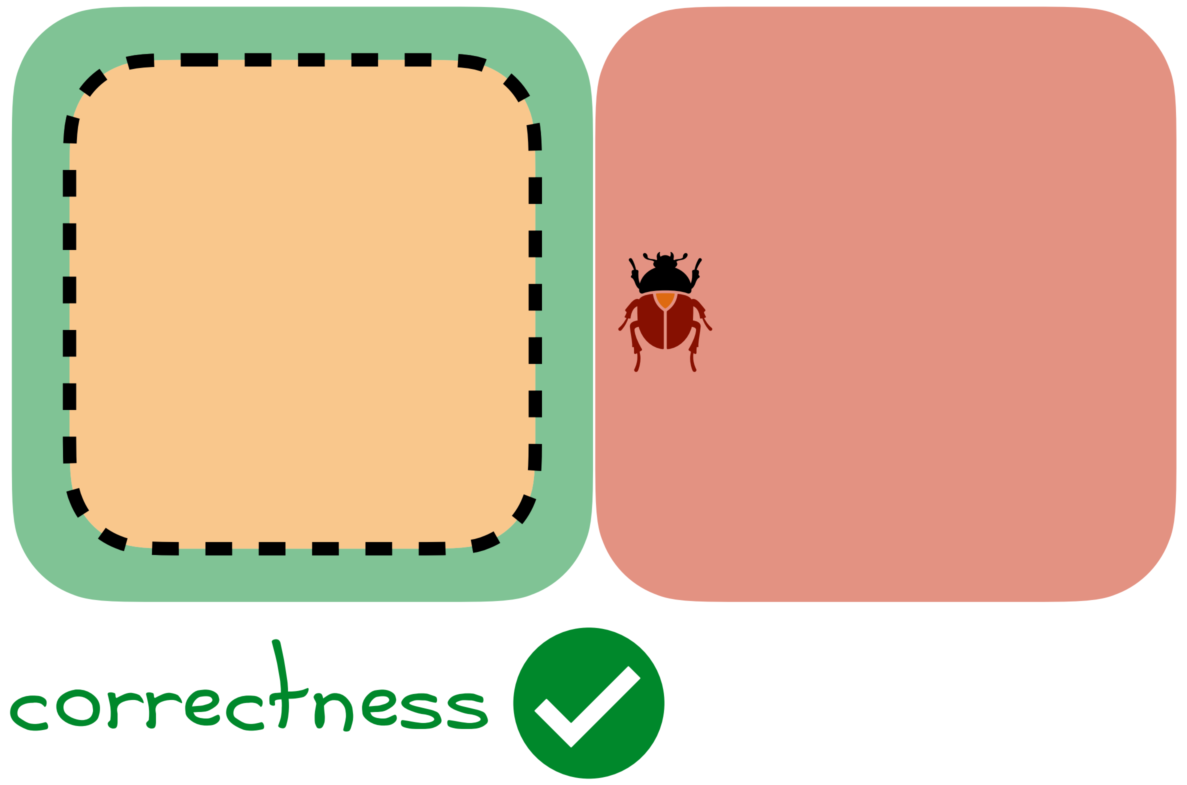

An Axiomatic Basis for Computer Programming

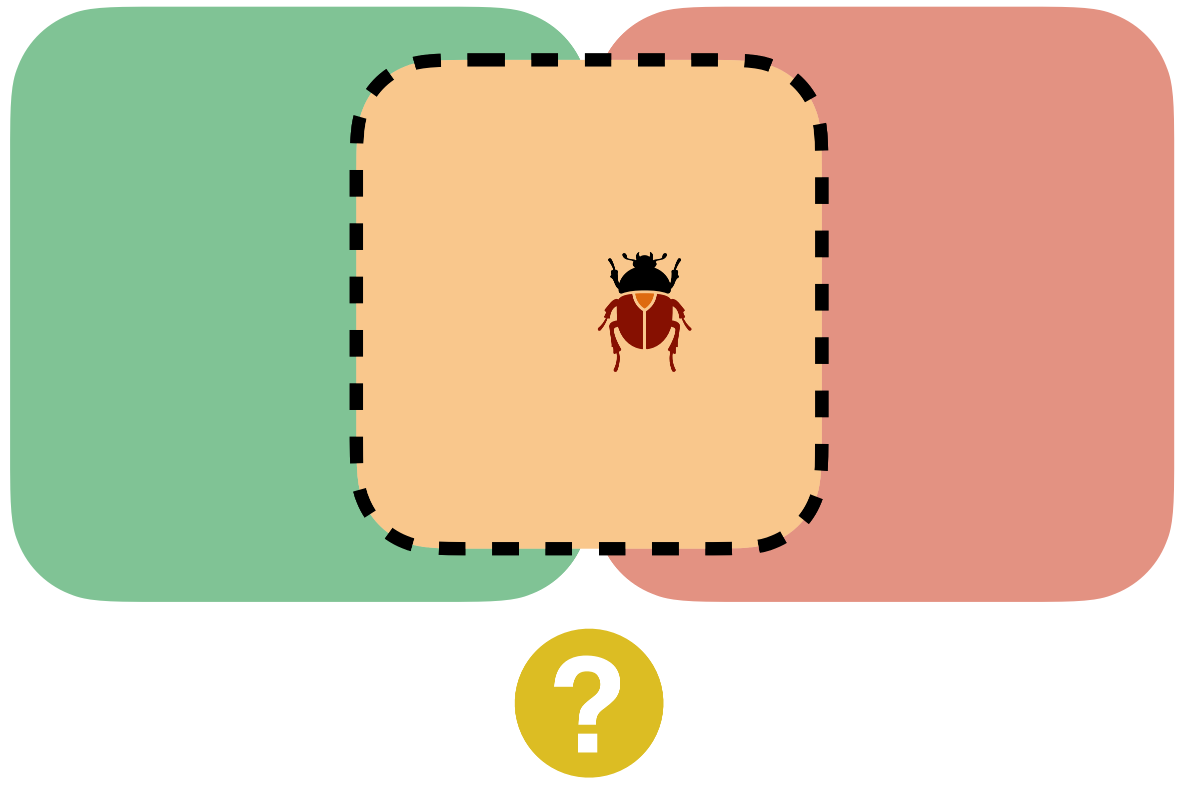
C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and

Over- vs Under-approximations





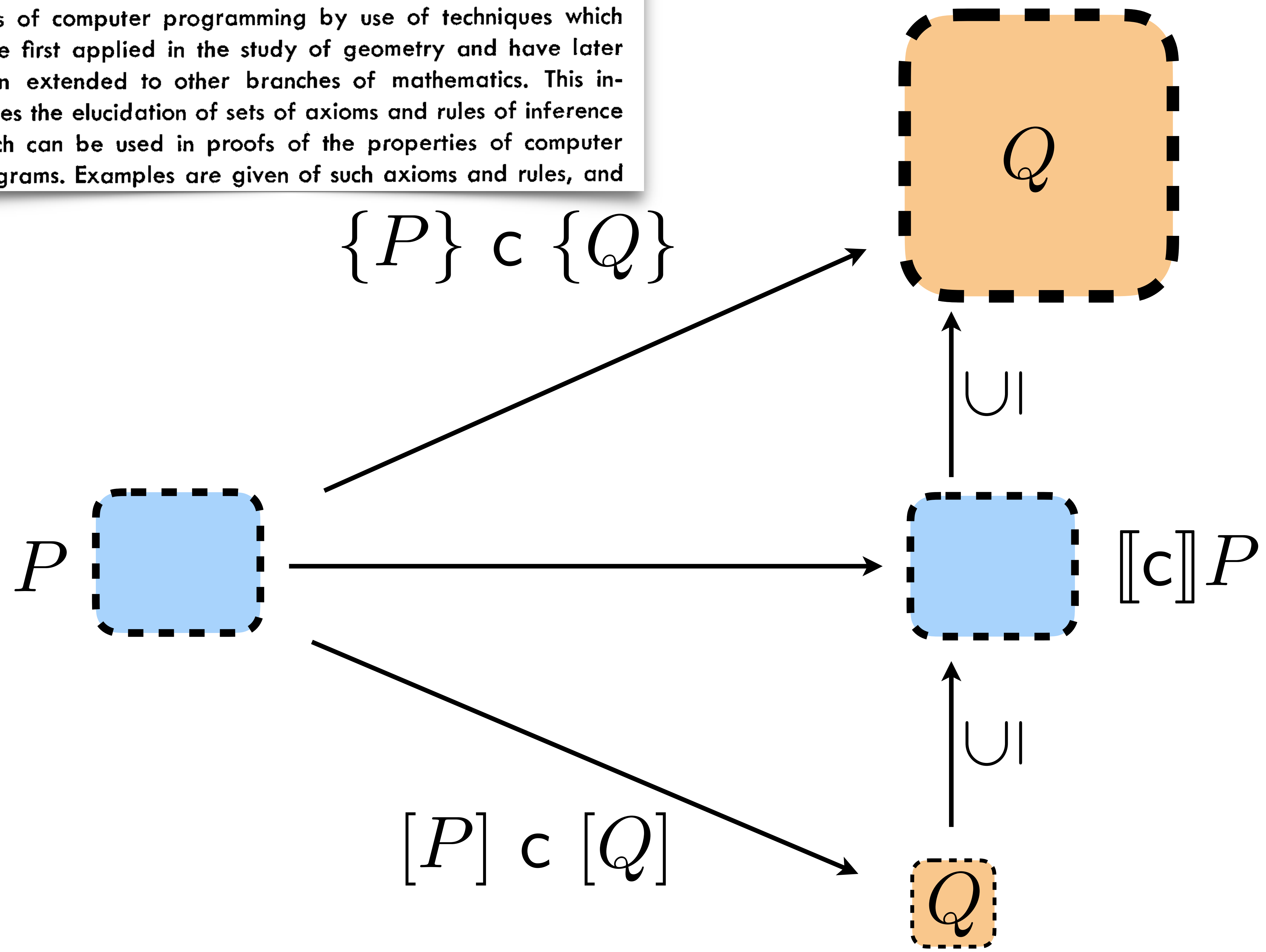


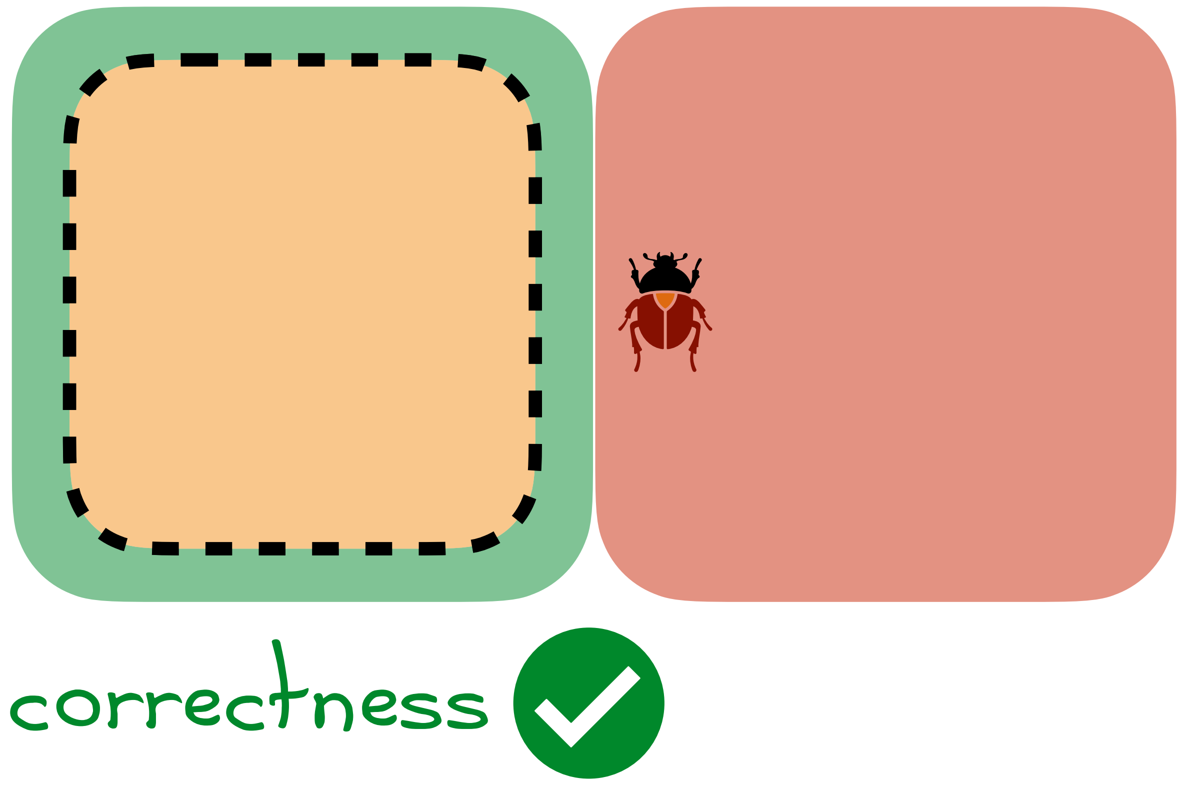
An Axiomatic Basis for Computer Programming

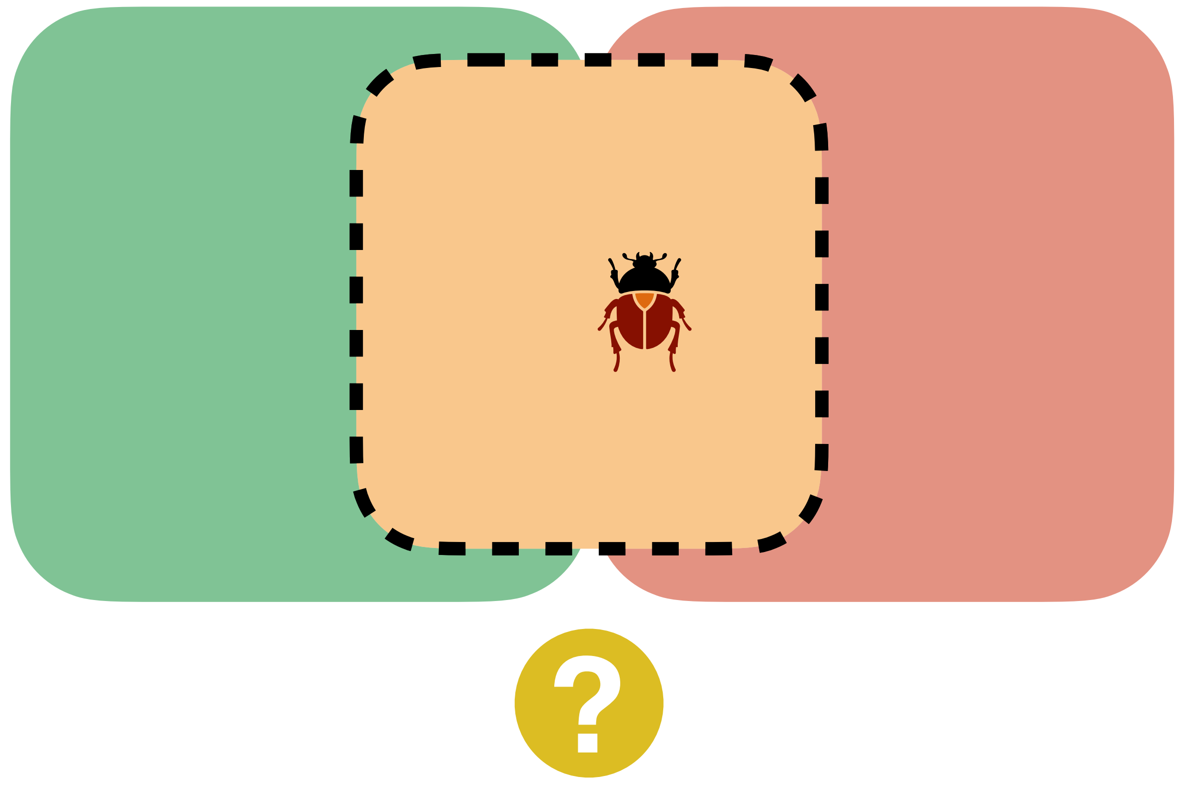
C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and

Over- vs Under-approximations







Incorrectness Logic

PETER W. O'HEARN, Facebook and University College London, UK

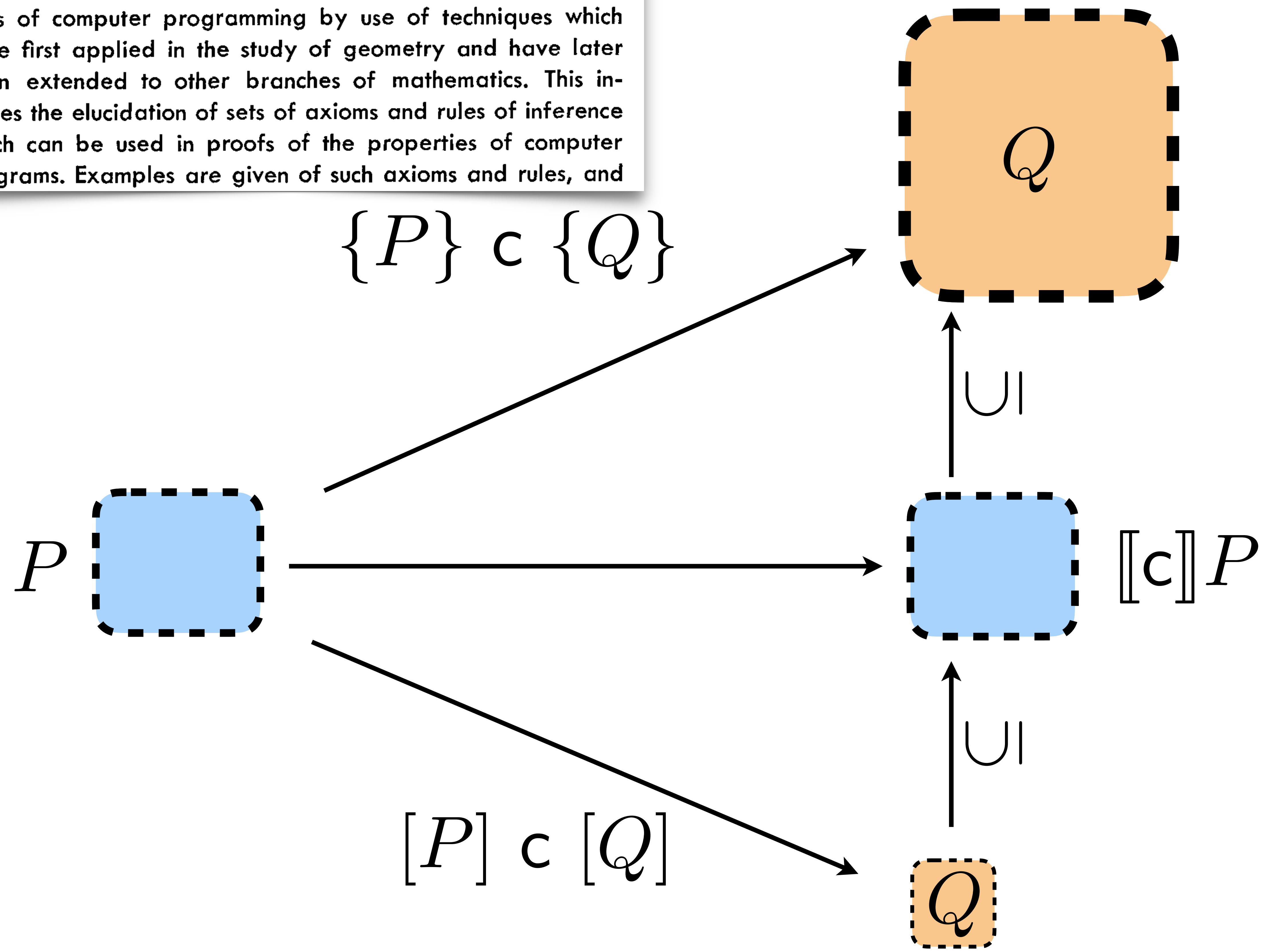
Program correctness and incorrectness are two sides of the same coin. As a programmer, even if you would like to have correctness, you might find yourself spending most of your time reasoning about incorrectness. This includes informal reasoning that people do while looking at or thinking about their code, as well as that supported by automated testing and static analysis tools. This paper describes a simple logic for program incorrectness which is, in a sense, the other side of the coin to Hoare's logic of correctness.

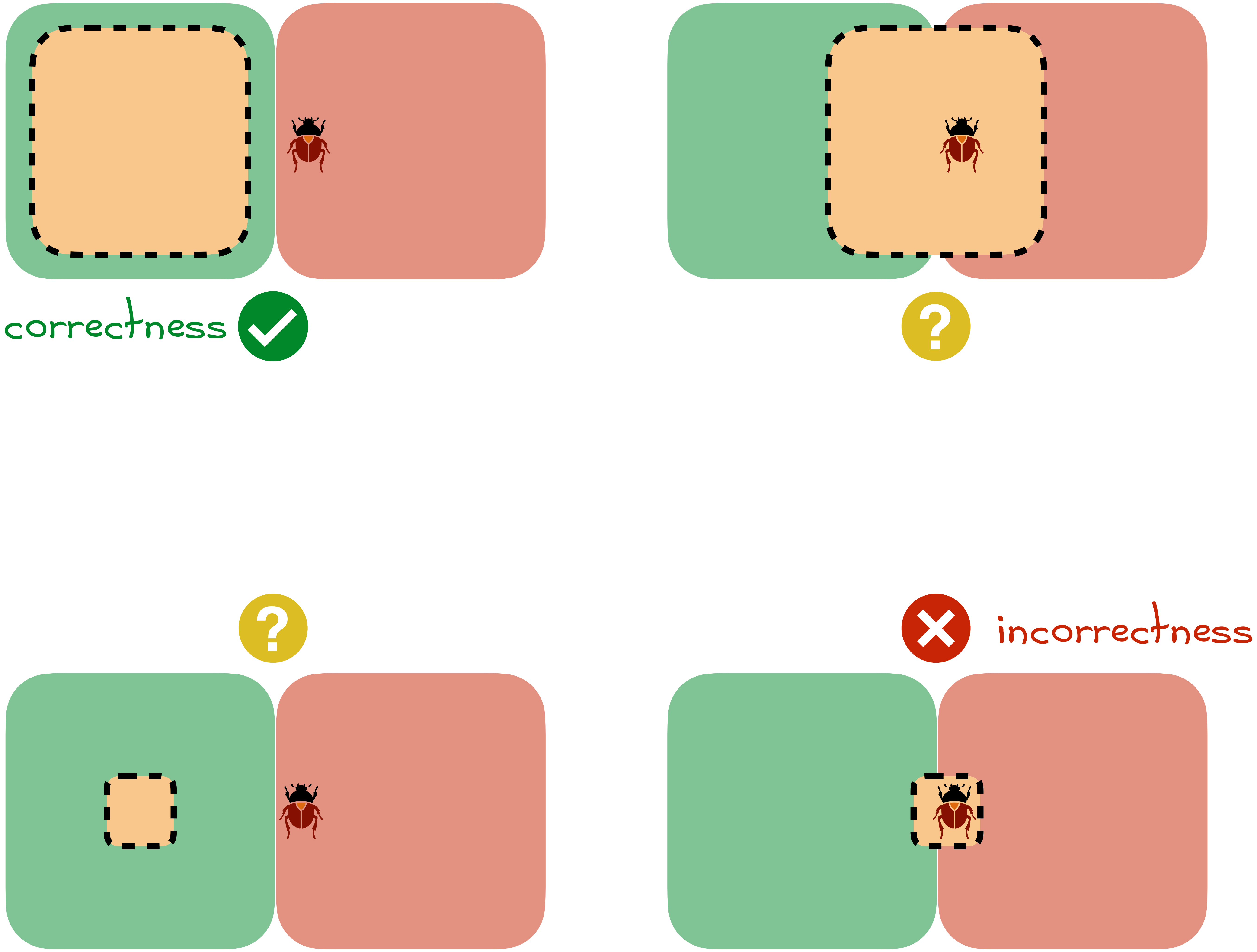
Over- vs Under-approximations

An Axiomatic Basis for Computer Programming

C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and





Incorrectness Logic

PETER W. O'HEARN, Facebook and University College London, UK

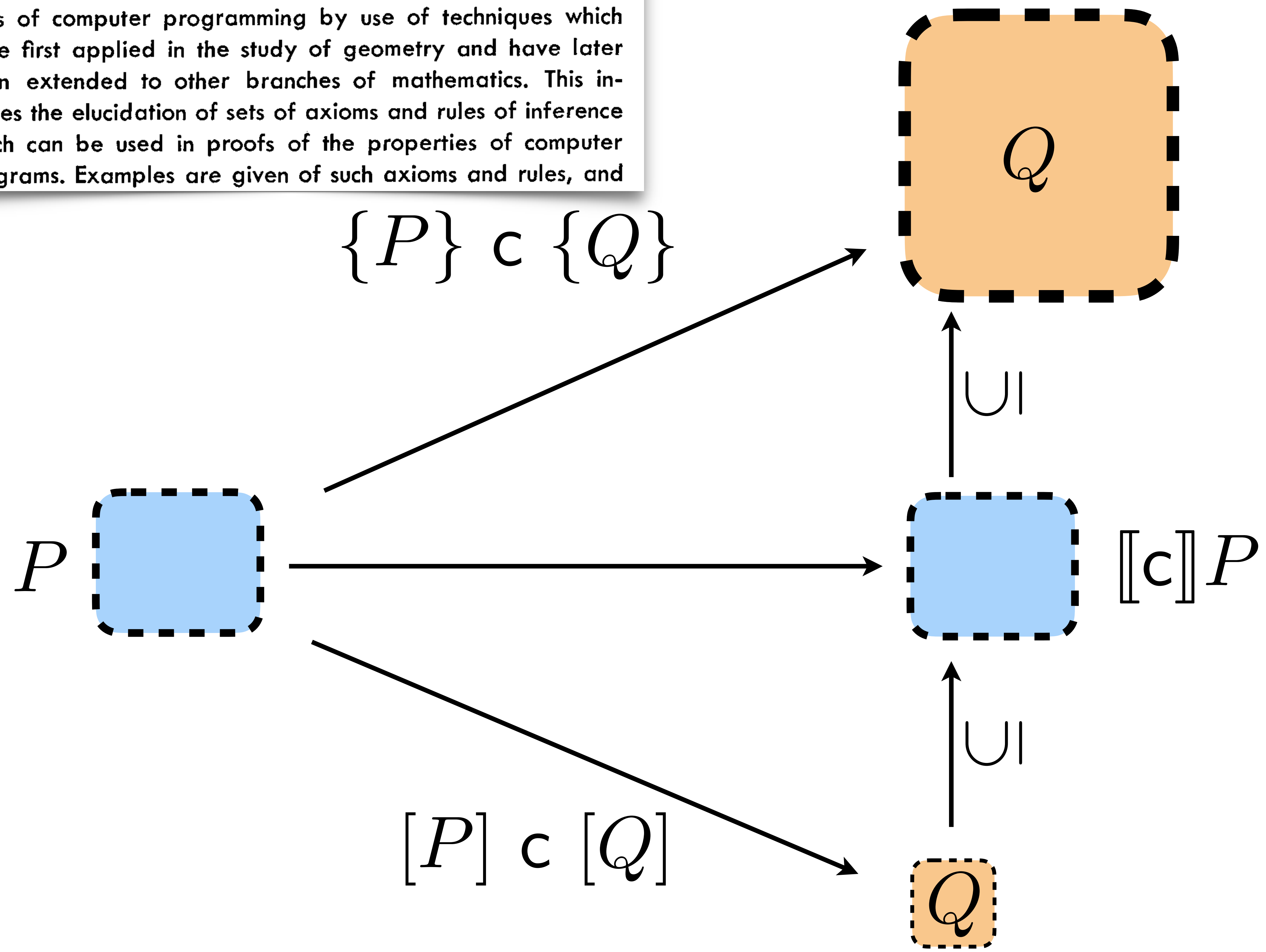
Program correctness and incorrectness are two sides of the same coin. As a programmer, even if you would like to have correctness, you might find yourself spending most of your time reasoning about incorrectness. This includes informal reasoning that people do while looking at or thinking about their code, as well as that supported by automated testing and static analysis tools. This paper describes a simple logic for program incorrectness which is, in a sense, the other side of the coin to Hoare's logic of correctness.

An Axiomatic Basis for Computer Programming

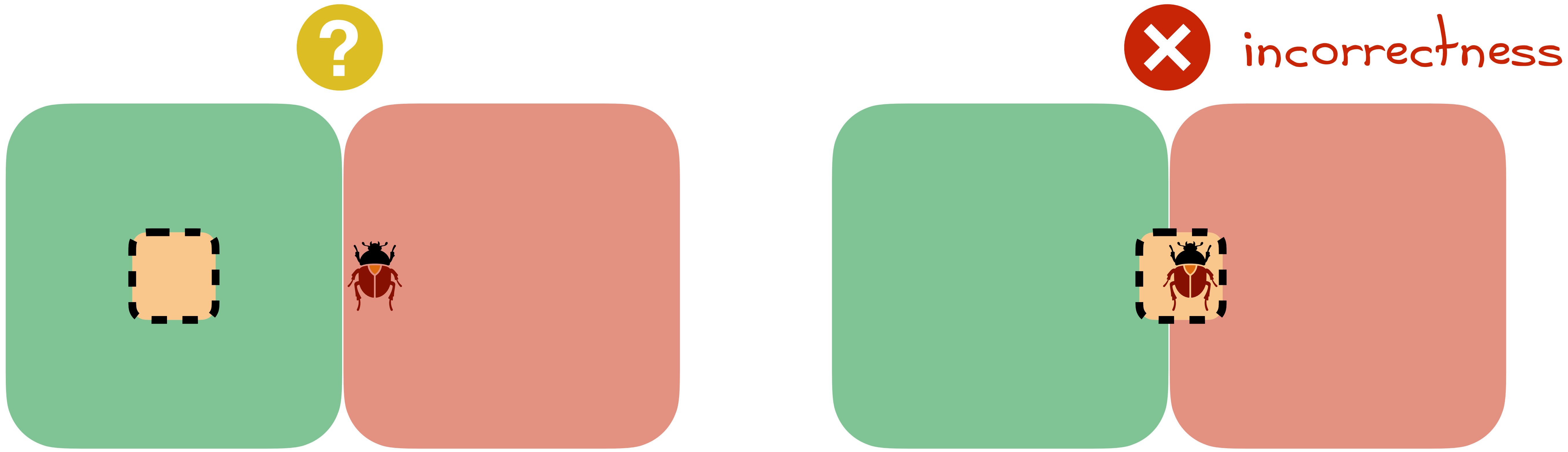
C. A. R. HOARE
The Queen's University of Belfast, Northern Ireland*

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and

Over- vs Under-approximations



"Correctness and incorrectness are two sides of the same coin" Peter O'Hearn



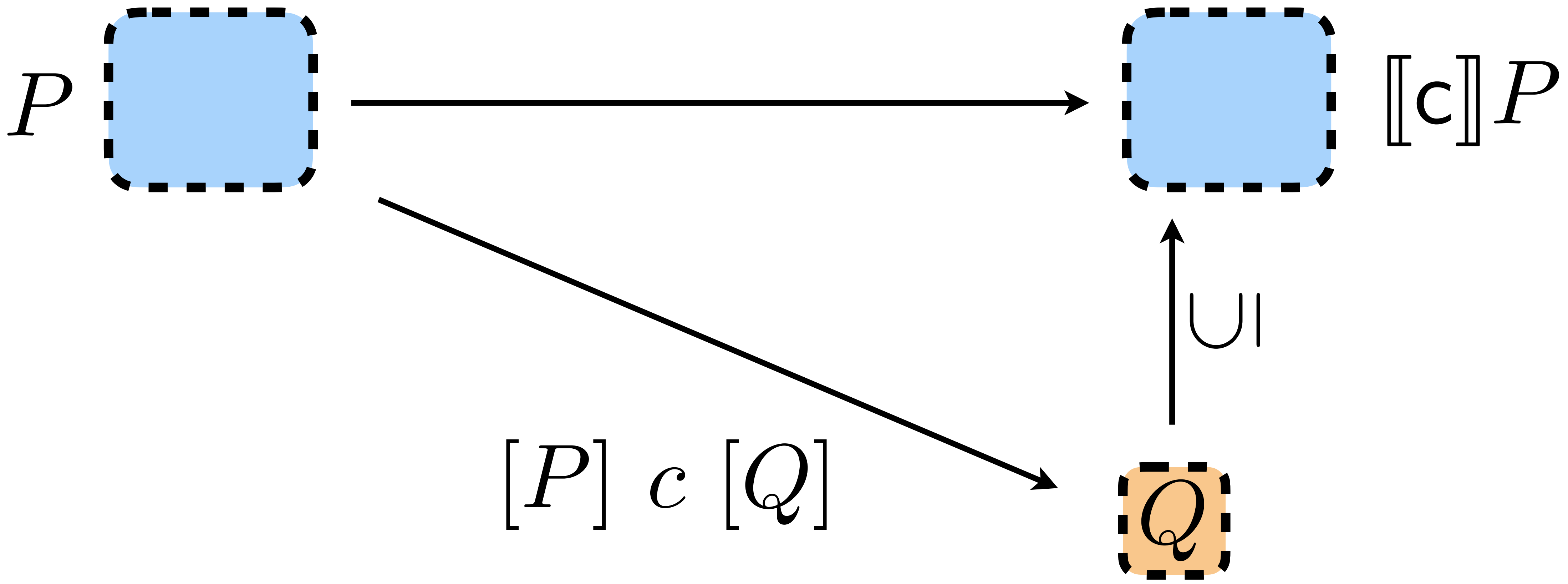
Incorrectness Logic

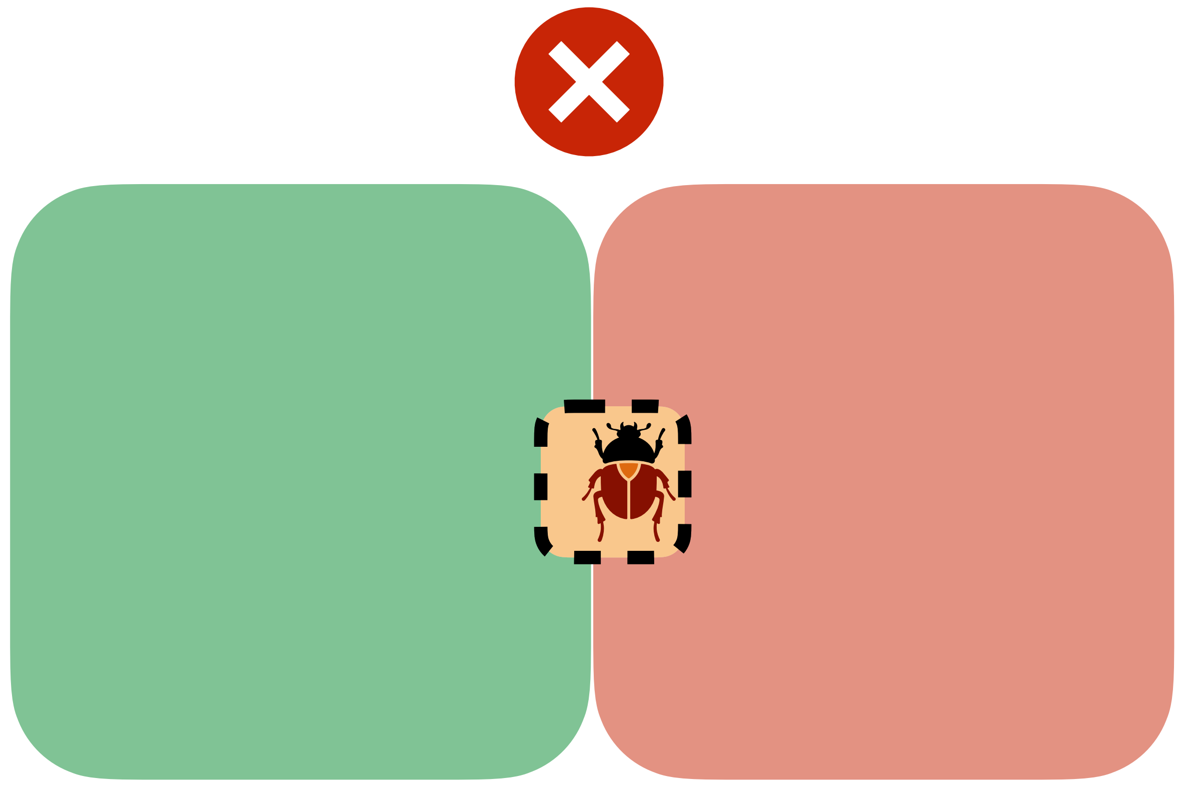
PETER W. O'HEARN, Facebook and University College London, UK

Program correctness and incorrectness are two sides of the same coin. As a programmer, even if you would like to have correctness, you might find yourself spending most of your time reasoning about incorrectness. This includes informal reasoning that people do while looking at or thinking about their code, as well as that supported by automated testing and static analysis tools. This paper describes a simple logic for program incorrectness which is, in a sense, the other side of the coin to Hoare's logic of correctness.

The idea

$$Q \subseteq [\![\mathsf{c}]\!]P$$





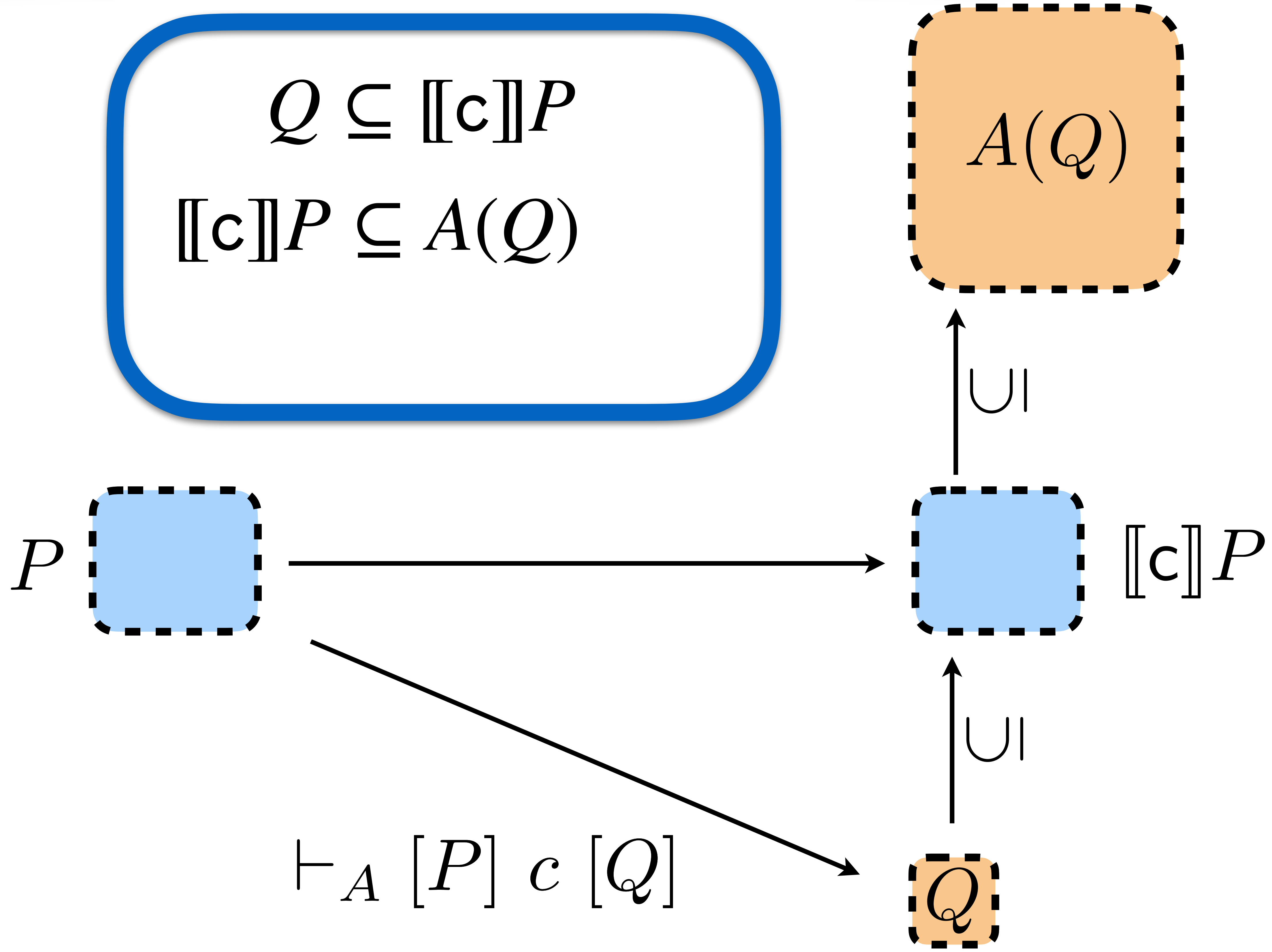
The idea

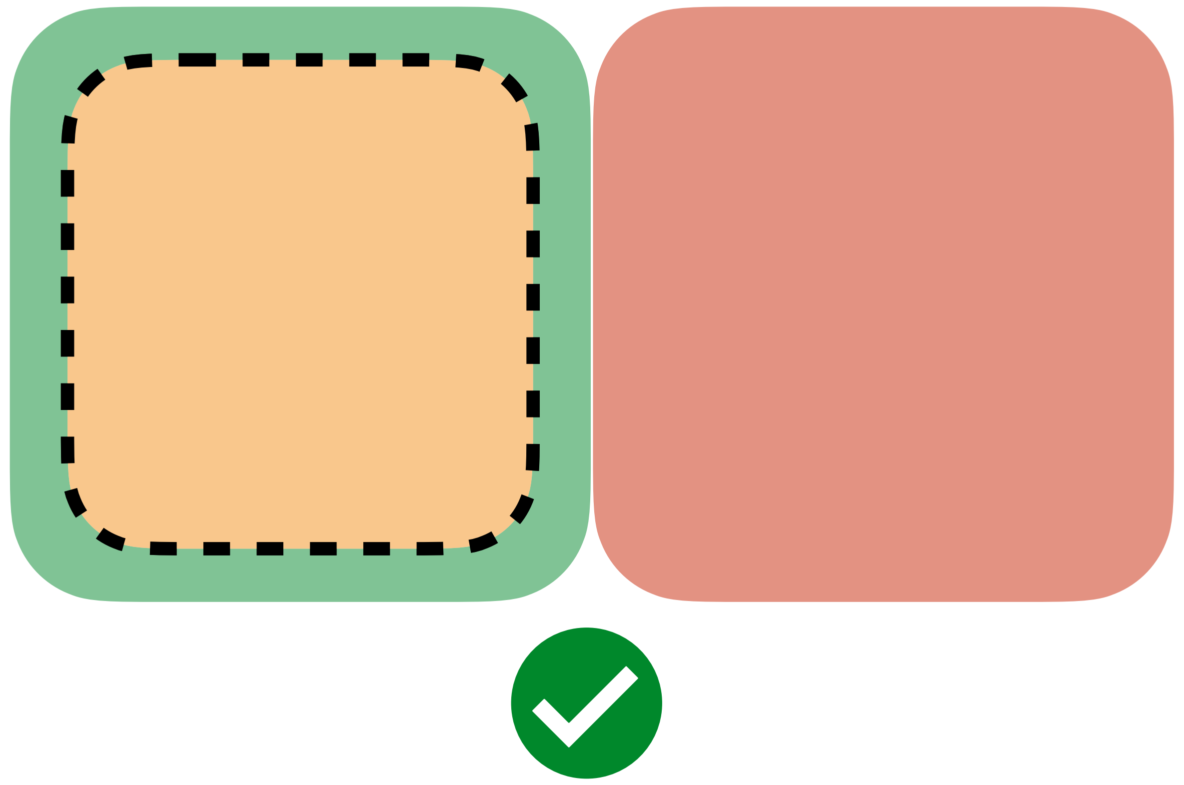
ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

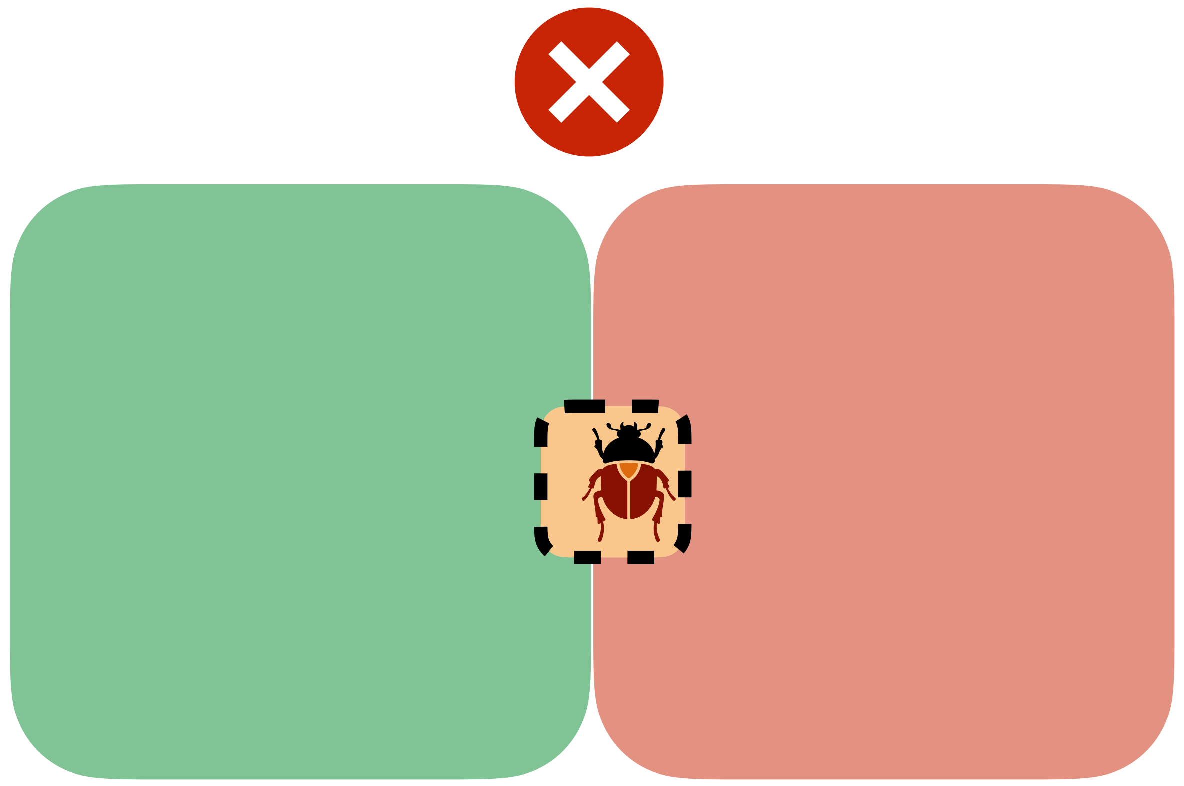
Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

$$Q \subseteq [\![\mathsf{c}]\!]P$$
$$[\![\mathsf{c}]\!]P \subseteq A(Q)$$







ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

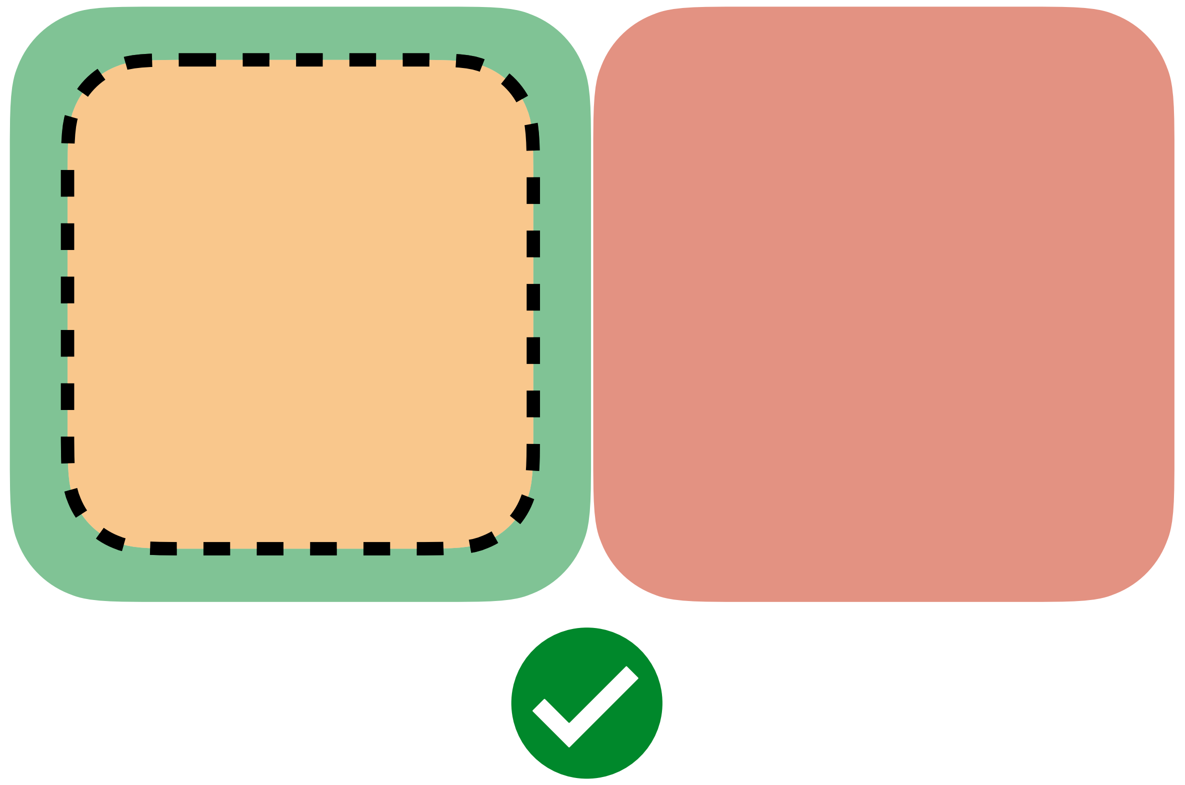
Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

The idea

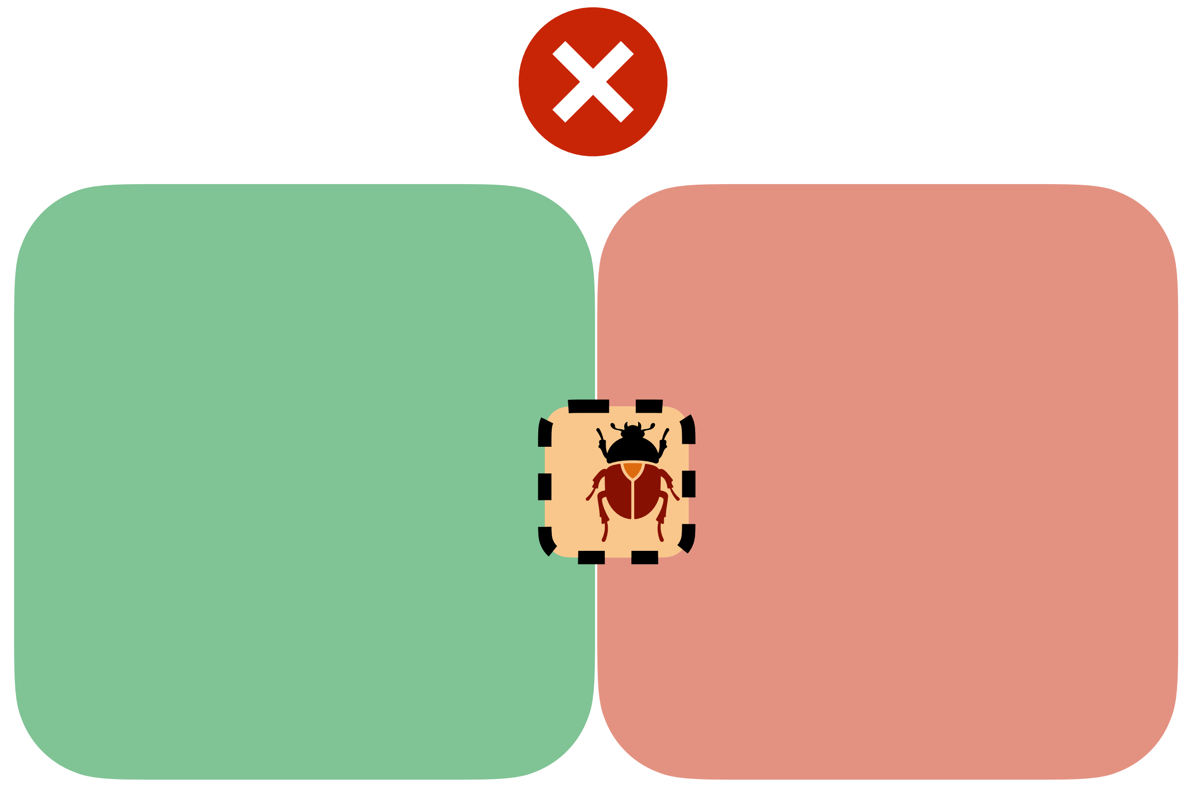
$$Q \subseteq [\![c]\!]P$$
$$[\![c]\!]P \subseteq A(Q)$$
$$A(Spec) = Spec$$

$$\begin{tikzcd}
 & A(Q) \\
P \arrow[r] \arrow[rd, "\vdash_A [P]\ c\ [Q]"'] & [\![c]\!]P \arrow[u, "\cup\!\mid"'] \\
 & Q \arrow[u, "\cup\!\mid"']
\end{tikzcd}$$



Locally Complete Abstraction

$$A(Q) \subseteq Spec$$
$$\Leftrightarrow$$
$$[\![c]\!]P \subseteq Spec$$
$$\Leftrightarrow$$
$$Q \subseteq Spec$$



ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

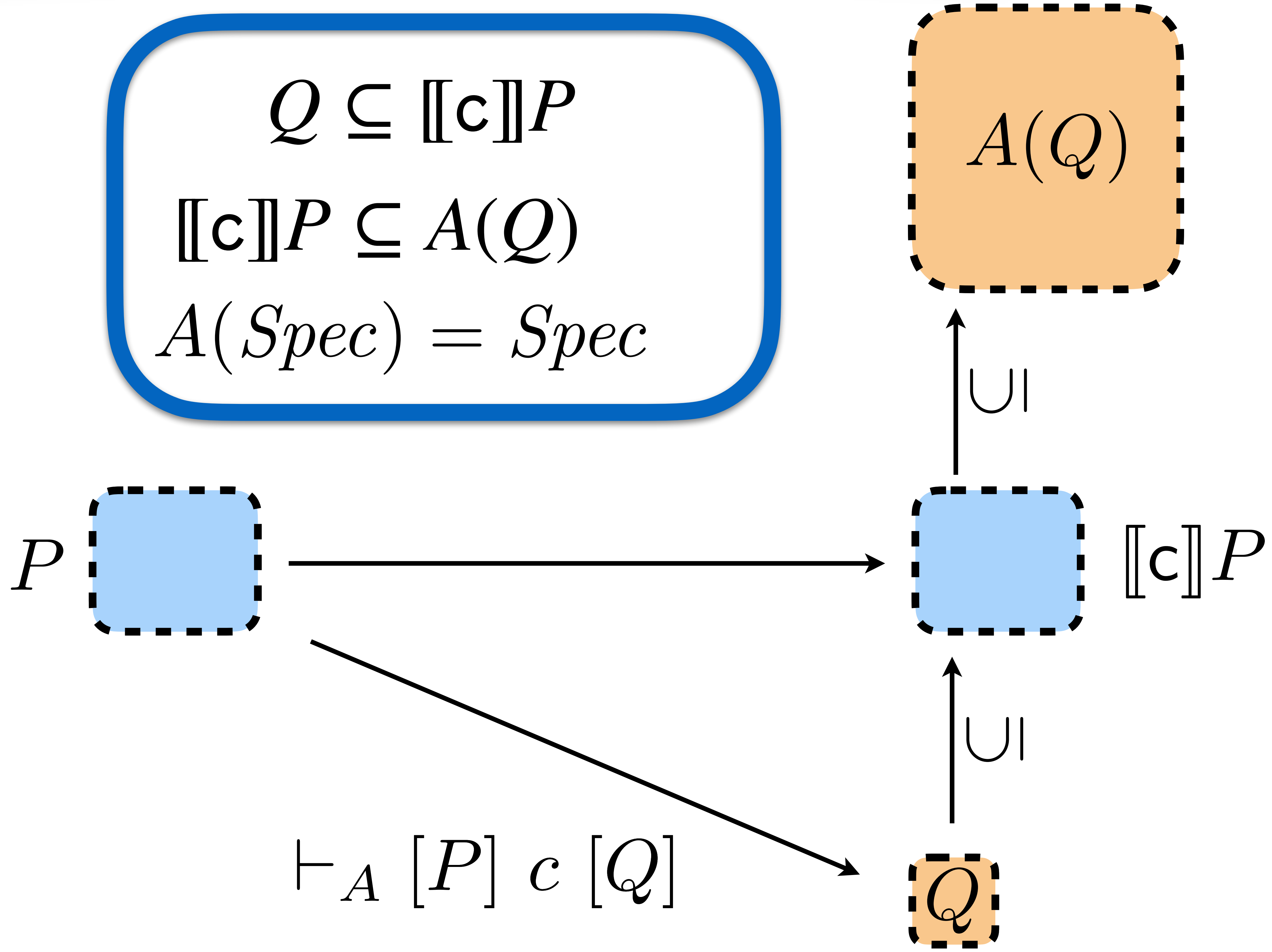
Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

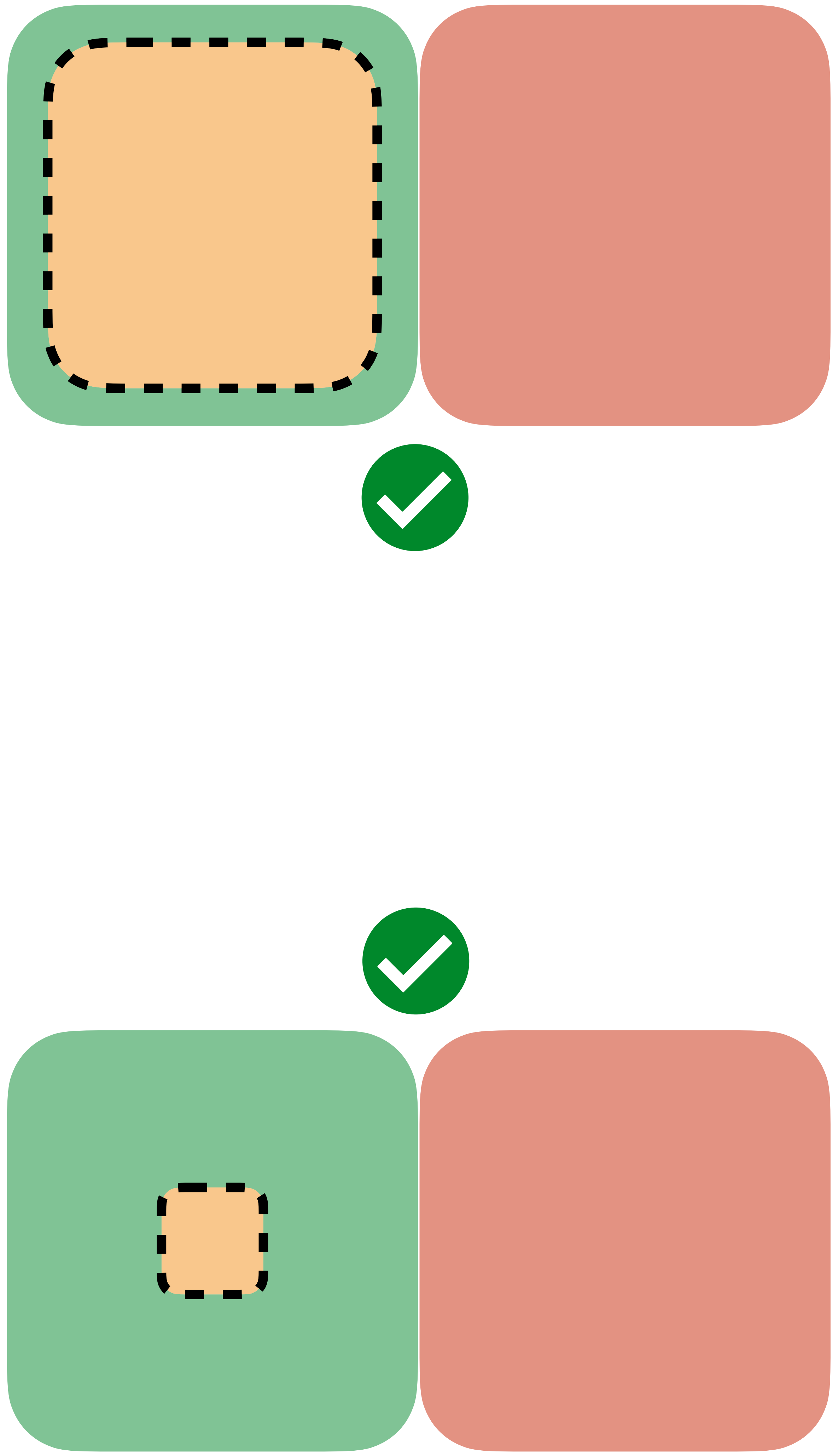
The idea

$$Q \subseteq [\![c]\!]P$$

$$[\![c]\!]P \subseteq A(Q)$$

$$A(Spec) = Spec$$





Locally Complete Abstraction

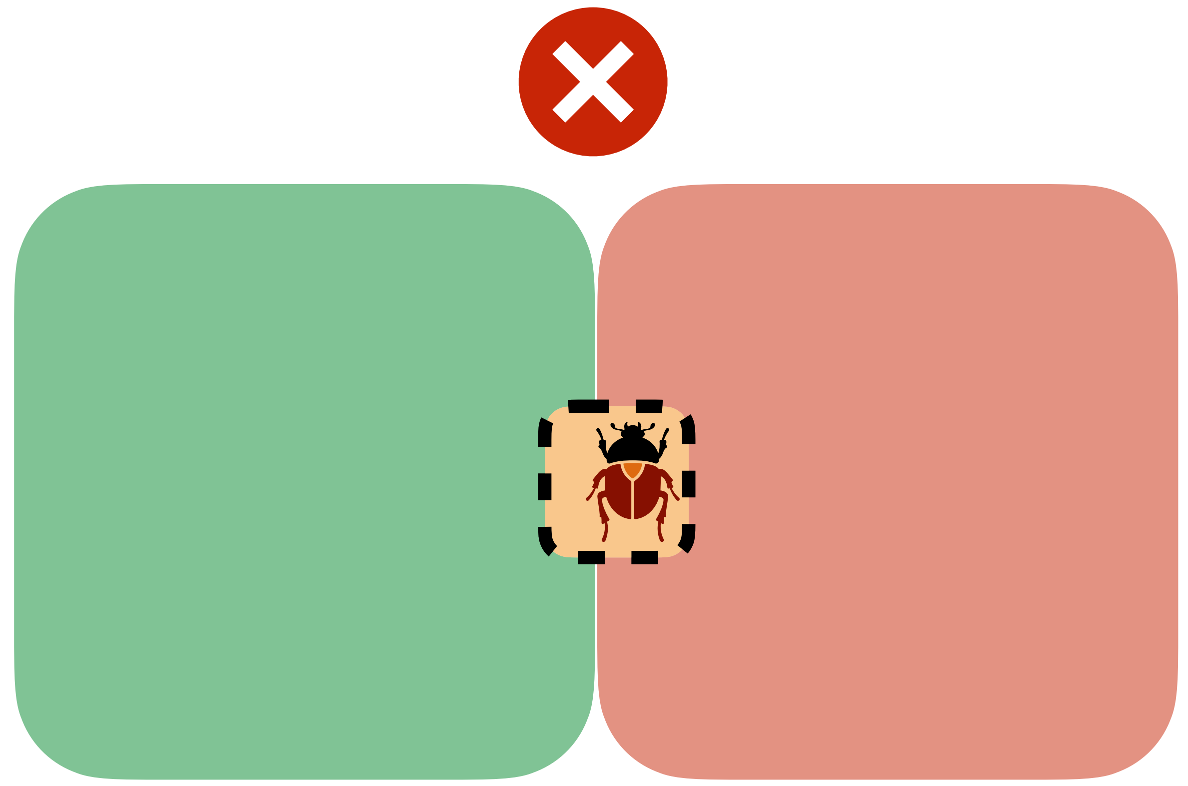
$$A(Q) \subseteq Spec$$

$$\Leftrightarrow$$

$$[\![c]\!]P \subseteq Spec$$

$$\Leftrightarrow$$

$$Q \subseteq Spec$$



ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S. ... 53
38041 Grenoble cedex

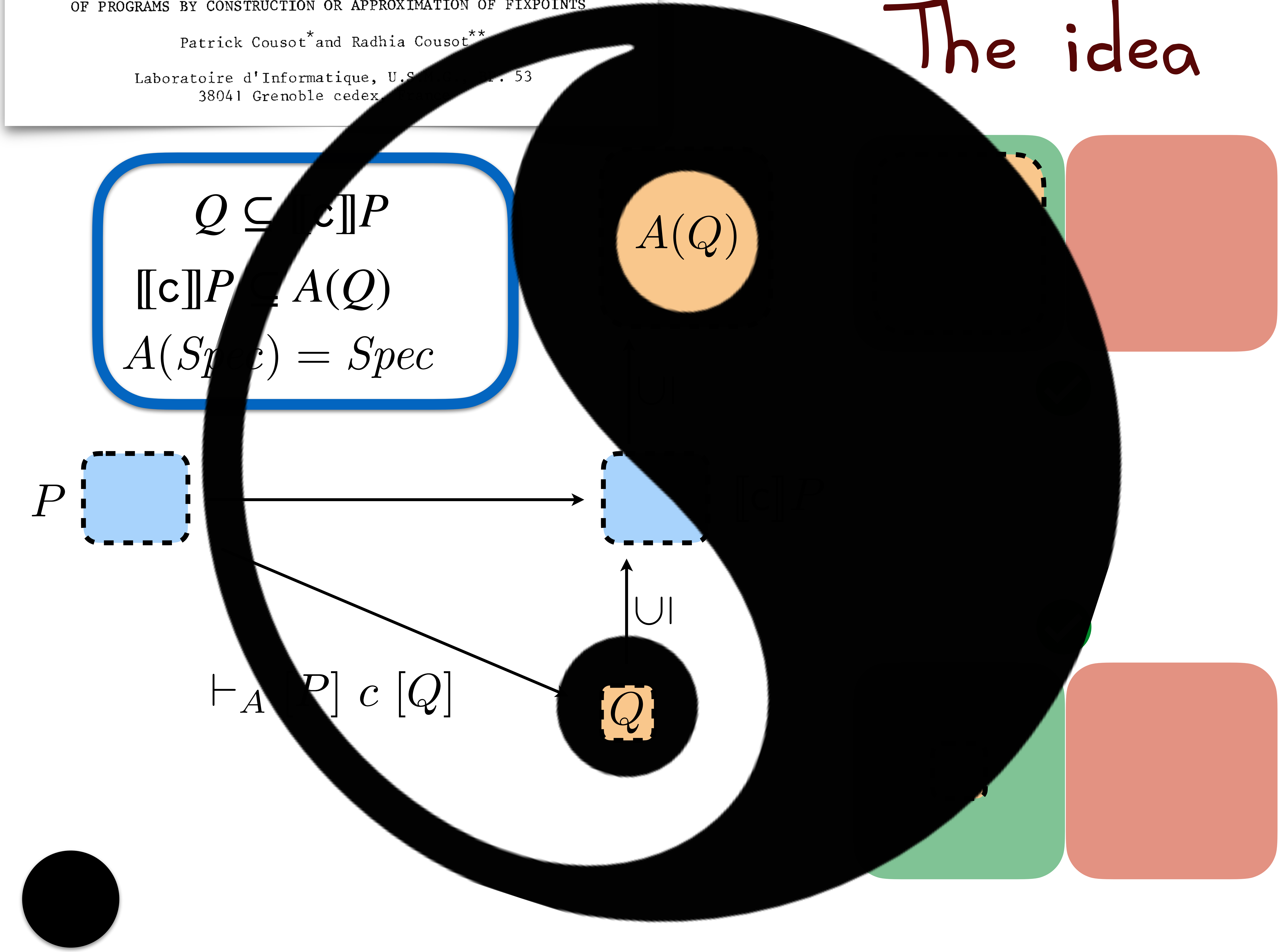
The idea

Locally Complete Abstraction

$Q \subseteq [\![c]\!]P$

$[\![c]\!]P \subseteq A(Q)$

$A(Spec) = Spec$



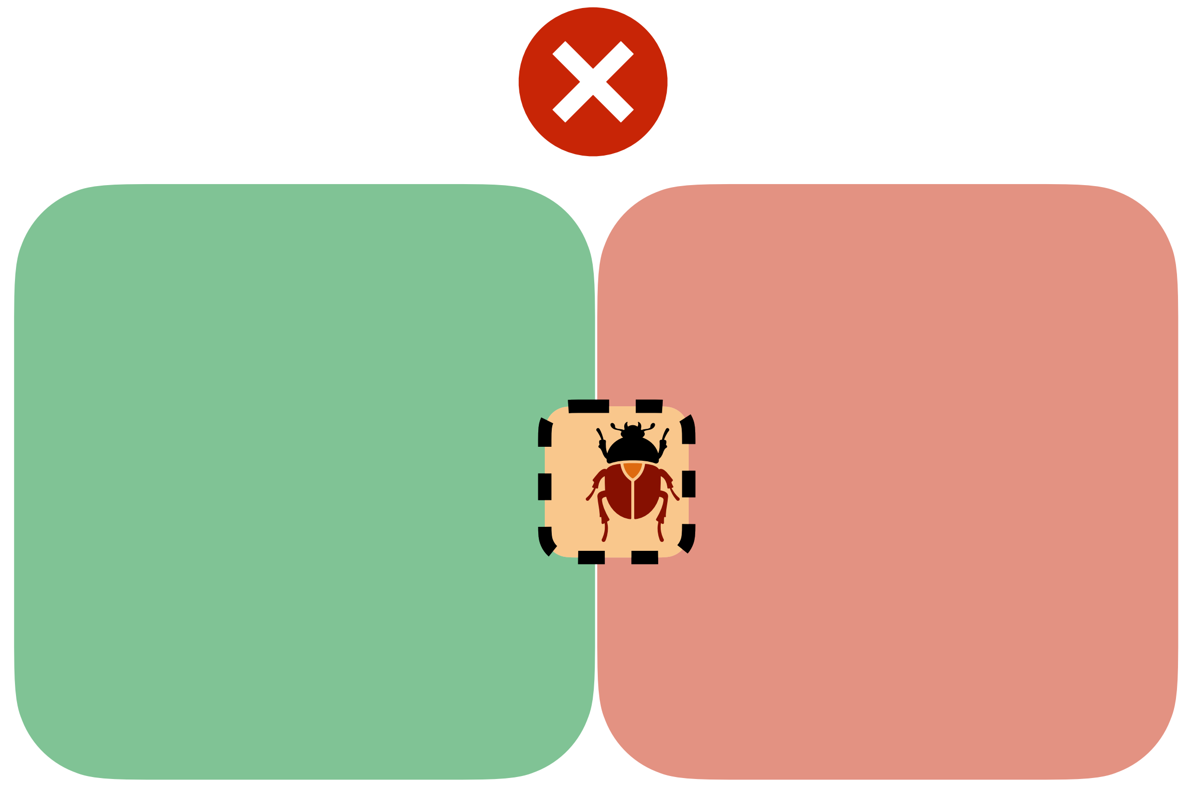
$A(Q) \subseteq Spec$

$\Leftrightarrow$

$[\![c]\!]P \subseteq Spec$

$\Leftrightarrow$

$Q \subseteq Spec$



Completeness in Abstract Interpretation

Abstract Interpretation (Mathematically)

A concrete complete lattice

$$\begin{tikzcd} C \arrow[r, bend right, "\alpha"'] & A \arrow[l, bend right, "\gamma"'] \end{tikzcd}$$

An abstract complete lattice

A Galois connection $c \leq \gamma(a) \quad \Leftrightarrow \quad \alpha(c) \leq a$

A Galois insertion $id = \alpha \circ \gamma : A \to A$ (γ is injective)

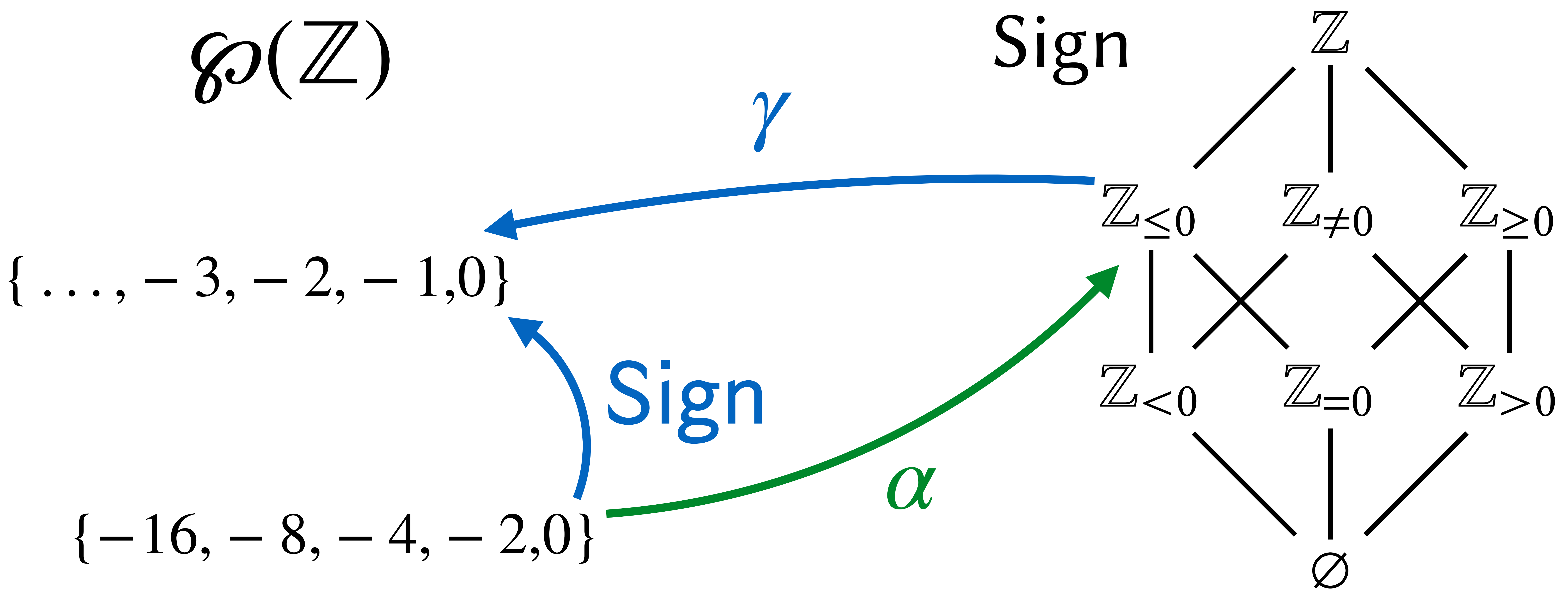
Closure operator $A = \gamma \circ \alpha : C \to C$ (A is idempotent)

Abstract Interpretation (by example)

A concrete complete lattice

$$\begin{tikzcd} C \arrow[r, bend right, "\alpha"'] & A \arrow[l, bend right, "\gamma"'] \end{tikzcd}$$

An abstract complete lattice



Abstract Interpretation (as closure)

$\wp(\{1,2,3,4,5\})$

Odd&Even

expressible elements

$\boxed{\{1,2,3,4,5\}}$

$\{1,2,3,4\}$ $\{1,2,3,5\}$ $\{1,2,4,5\}$ $\{1,3,4,5\}$ $\{2,3,4,5\}$

$\{1,2,3\}$ $\{1,2,4\}$ $\{1,2,5\}$ $\{1,3,4\}$ $\boxed{\{1,3,5\}}$ $\{1,4,5\}$ $\{2,3,4\}$ $\{2,3,5\}$ $\{2,4,5\}$ $\{3,4,5\}$

$\{1,2\}$ $\{1,3\}$ $\{1,4\}$ $\{1,5\}$ $\{2,3\}$ $\boxed{\{2,4\}}$ $\{2,5\}$ $\{3,4\}$ $\{3,5\}$ $\{4,5\}$

$\{1\}$ $\{2\}$ $\{3\}$ $\{4\}$ $\{5\}$

$\boxed{\emptyset}$

Abstract Interpretation (as closure)

$\wp(\{1,2,3,4,5\})$

LessMoreThan3

$\boxed{\{1,2,3,4,5\}}$

$\{1,2,3,4\}$ $\{1,2,3,5\}$ $\{1,2,4,5\}$ $\{1,3,4,5\}$ $\{2,3,4,5\}$

$\boxed{\{1,2,3\}}$ $\{1,2,4\}$ $\{1,2,5\}$ $\{1,3,4\}$ $\{1,3,5\}$ $\{1,4,5\}$ $\{2,3,4\}$ $\{2,3,5\}$ $\{2,4,5\}$ $\boxed{\{3,4,5\}}$

$\{1,2\}$ $\{1,3\}$ $\{1,4\}$ $\{1,5\}$ $\{2,3\}$ $\{2,4\}$ $\{2,5\}$ $\{3,4\}$ $\{3,5\}$ $\{4,5\}$

$\{1\}$ $\{2\}$ $\{3\}$ $\{4\}$ $\{5\}$

$\boxed{\emptyset}$

Abstract Interpretation (as closure)

$\wp(\{1,2,3,4,5\})$

LessMoreThan3

$\{1,2,3,4,5\}$

$\{1,2,3,4\}$ $\{1,2,3,5\}$ $\{1,2,4,5\}$ $\{1,3,4,5\}$ $\{2,3,4,5\}$

$\{1,2,3\}$ $\{1,2,4\}$ $\{1,2,5\}$ $\{1,3,4\}$ $\{1,3,5\}$ $\{1,4,5\}$ $\{2,3,4\}$ $\{2,3,5\}$ $\{2,4,5\}$ $\{3,4,5\}$

$\{1,2\}$ $\{1,3\}$ $\{1,4\}$ $\{1,5\}$ $\{2,3\}$ $\{2,4\}$ $\{2,5\}$ $\{3,4\}$ $\{3,5\}$ $\{4,5\}$

Must be closed under meet! (Moore closure)

$\{1\}$ $\{2\}$ $\{3\}$ $\{4\}$ $\{5\}$

$\emptyset$

Soundness (by construction)

$$\begin{tikzcd} A \arrow[r, bend left, "f^\sharp"] & A \\ C \arrow[u, bend left, "\alpha"] \arrow[r, bend right, "f"'] \arrow[ur, dotted, "\leq"] & C \arrow[u, bend right, "\alpha"'] \end{tikzcd}$$

$$\alpha \circ f \leq f^\sharp \circ \alpha$$

Abstract interpretation
computes over-approximations!
(ok to prove absence of bugs)

Best correct approximation (bca)

$$\begin{tikzcd} A \arrow[r, bend left, "f^A"] \arrow[d, bend right, "\gamma"'] \arrow[dr, phantom, "\triangleq"] & A \\ C \arrow[r, bend right, "f"'] & C \arrow[u, bend right, "\alpha"'] \end{tikzcd}$$

$$f^A \leq f^\sharp$$

As much precise as possible!

Completeness (maximal precision)

$$\begin{tikzcd}
A \arrow[r, bend left, "f^\sharp"] & A \\
C \arrow[u, bend left, "\alpha"] \arrow[r, bend right, "f"'] \arrow[ur, dotted, "="] & C \arrow[u, bend right, "\alpha"']
\end{tikzcd}$$

$$f^A \circ \alpha$$

$$\leq \quad \leq$$

$$\alpha \circ f = f^\sharp \circ \alpha$$

It must be the bca!

in terms of closures, requires:

$$A\,f = A\,f\,A$$

Program Analysis with Abstract Interpretation

Regular commands

$$\mathsf{Reg} \ni \mathsf{r} ::= \mathsf{e} \mid \mathsf{r};\mathsf{r} \mid \mathsf{r} \oplus \mathsf{r} \mid \mathsf{r}^*$$

$$\mathsf{AExp} \ni \mathsf{a} ::= v \in \mathbb{Z} \mid x \in \mathit{Var} \mid \mathsf{a} + \mathsf{a} \mid \mathsf{a} - \mathsf{a} \mid \mathsf{a} * \mathsf{a}$$

$$\mathsf{BExp} \ni \mathsf{b} ::= \mathbf{tt} \mid \mathbf{ff} \mid \mathsf{a} = \mathsf{a} \mid \mathsf{a} < \mathsf{a} \mid \mathsf{a} \leq \mathsf{a} \mid \mathsf{b} \wedge \mathsf{b} \mid \neg \mathsf{b}$$

$$\mathsf{Exp} \ni \mathsf{e} ::= \mathbf{skip} \mid x := \mathsf{a} \mid \mathsf{b}?$$

$$\textbf{if } (\mathsf{b}) \textbf{ then } \mathsf{c}_1 \textbf{ else } \mathsf{c}_2 \triangleq (\mathsf{b}?;\mathsf{c}_1) \oplus (\neg\mathsf{b}?;\mathsf{c}_2)$$

$$\textbf{while } (\mathsf{b}) \textbf{ do } \mathsf{c} \triangleq (\mathsf{b}?;\mathsf{c})^* ; \neg\mathsf{b}?$$

Collecting semantics

$$(\!|\mathbf{skip}|\!)S \triangleq S \qquad \{\!|\mathsf{a}|\!\} : \Sigma \to \mathbb{Z}$$

$$(\!|x := \mathsf{a}|\!)S \triangleq \{\sigma[x \mapsto \{\!|\mathsf{a}|\!\}\sigma] \mid \sigma \in S\}$$

$$(\!|\mathsf{b}?|\!)S \triangleq \{\sigma \in S \mid \{\!|\mathsf{b}|\!\}\sigma = \mathbf{tt}\}$$

$$\{\!|\mathsf{b}|\!\} : \Sigma \to \{\mathbf{tt}, \mathbf{ff}\}$$

$$[\![\mathsf{e}]\!]c \triangleq (\!|\mathsf{e}|\!)c$$

$$[\![\mathsf{r}_1 ; \mathsf{r}_2]\!]c \triangleq [\![\mathsf{r}_2]\!]([\![\mathsf{r}_1]\!]c)$$

$$[\![\mathsf{r}_1 \oplus \mathsf{r}_2]\!]c \triangleq [\![\mathsf{r}_1]\!]c \vee [\![\mathsf{r}_2]\!]c$$

$$[\![\mathsf{r}^*]\!]c \triangleq \bigvee\{[\![\mathsf{r}]\!]^n c \mid n \in \mathbb{N}\}$$

Abstract semantics

$$\llbracket \mathsf{e} \rrbracket^\sharp_A a \triangleq \llbracket \mathsf{e} \rrbracket^A a = (\alpha \circ \llbracket \mathsf{e} \rrbracket \circ \gamma) a$$

$$\llbracket \mathsf{r}_1 ; \mathsf{r}_2 \rrbracket^\sharp_A a \triangleq \llbracket \mathsf{r}_2 \rrbracket^\sharp_A (\llbracket \mathsf{r}_1 \rrbracket^\sharp_A a)$$

$$\llbracket \mathsf{r}_1 \oplus \mathsf{r}_2 \rrbracket^\sharp_A a \triangleq \llbracket \mathsf{r}_1 \rrbracket^\sharp_A a \vee_A \llbracket \mathsf{r}_2 \rrbracket^\sharp_A a$$

$$\llbracket \mathsf{r}^* \rrbracket^\sharp_A a \triangleq \bigvee_A \{ (\llbracket \mathsf{r} \rrbracket^\sharp_A)^n a \mid n \in \mathbb{N} \}$$

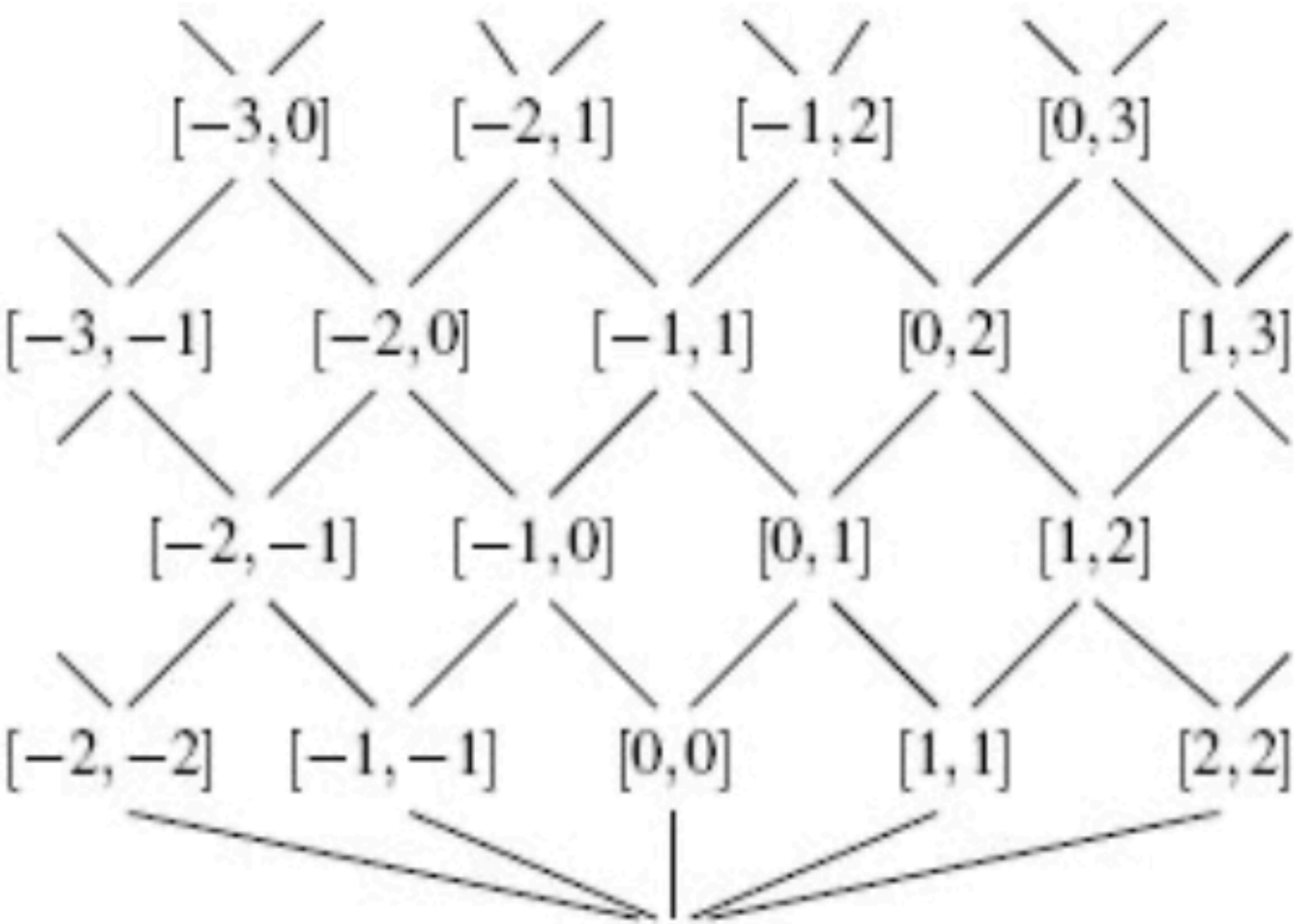
Just a composition of bcas!

An example of program analysis: Intervals domain

$[-\infty, +\infty]$

...

...



...

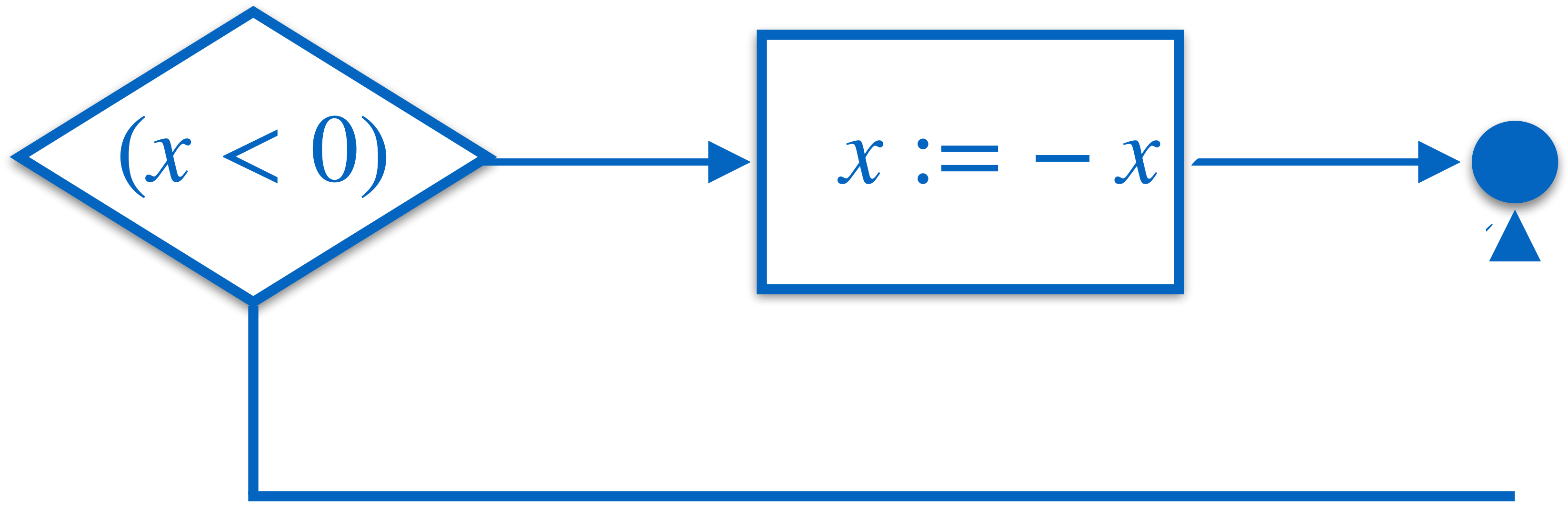
$\perp$

An example of program analysis

c = if $(x < 0)$ then $x := -x$

Concrete

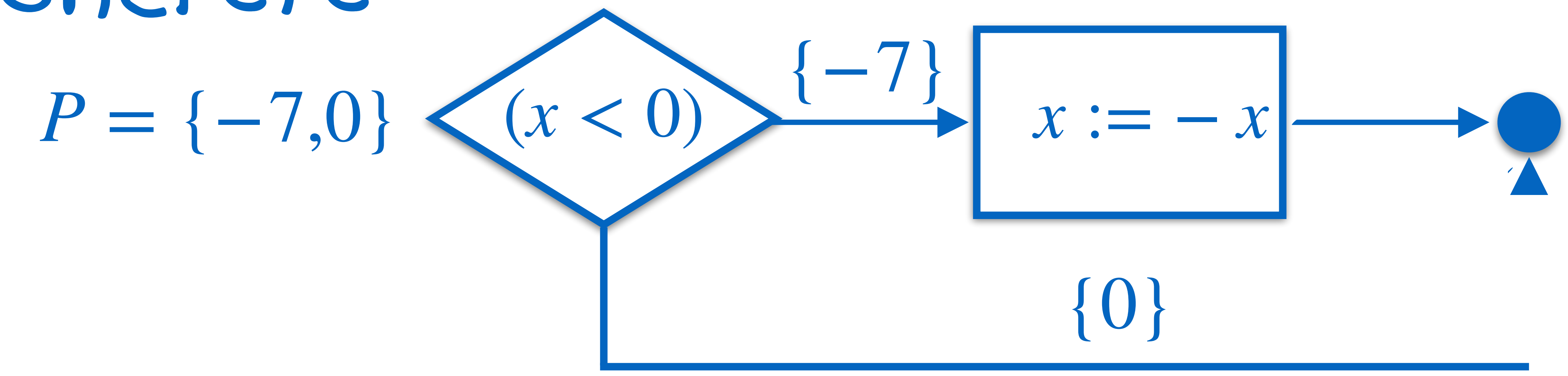
$P = \{-7, 0\}$



An example of program analysis

c = if $(x < 0)$ then $x := -x$

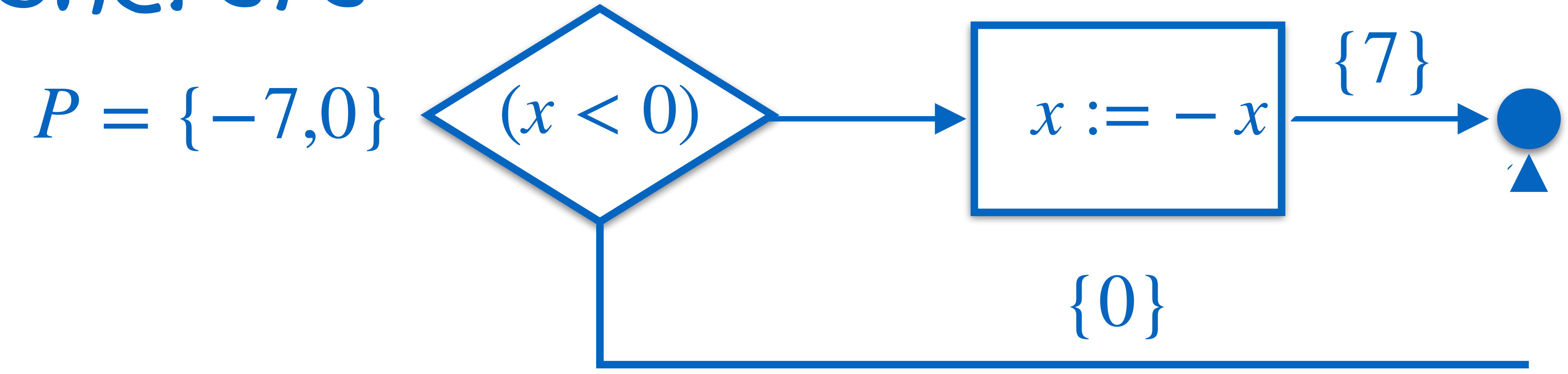
Concrete



An example of program analysis

c = if $(x < 0)$ then $x := -x$

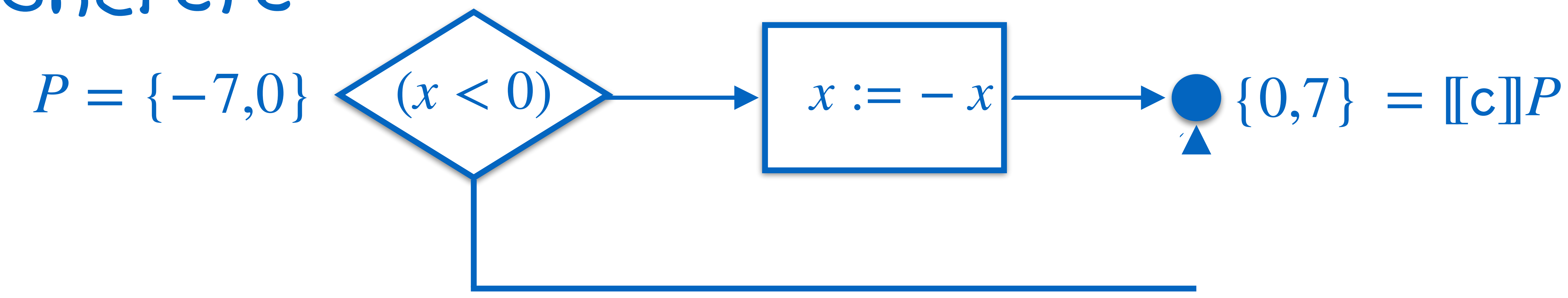
Concrete



An example of program analysis

$$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$$

Concrete



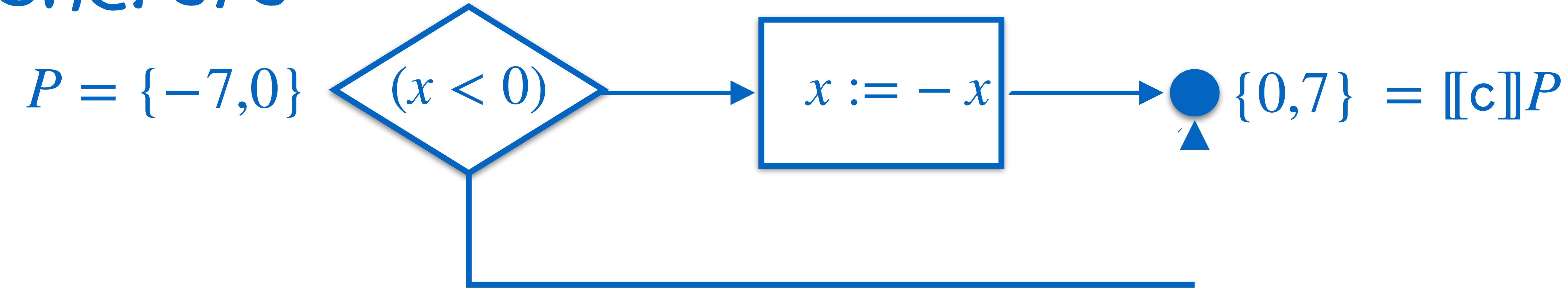
An example of program analysis

Abstract (ex. Intervals)

over-approximations are good for proving correctness but not for bug finding!

$$c = \text{if } (x < 0) \text{ then } x := -x$$

Concrete



An example of program analysis

Abstract (ex. Intervals)

over-approximations are good for proving correctness but not for bug finding!

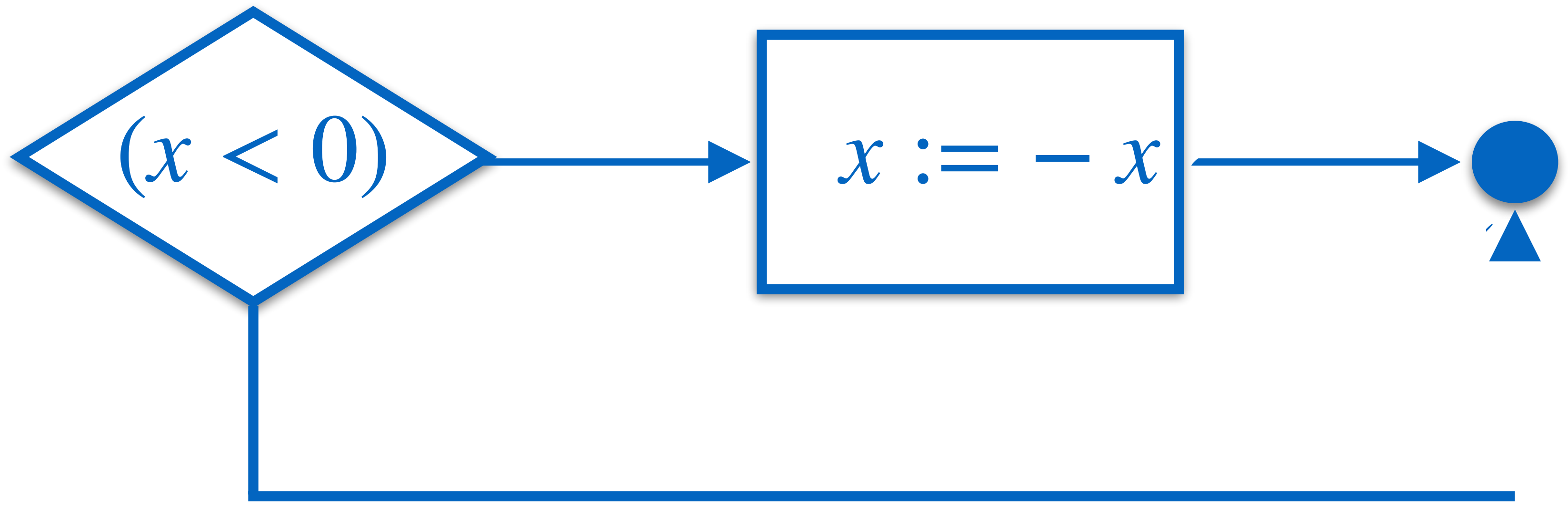
$A(P) = [-7,0]$

$P \subseteq A(P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

$P = \{-7,0\}$



$\{0,7\} = [\![\mathsf{c}]\!]P$

An example of program analysis

Abstract (ex. Intervals)

over-approximations are good for proving correctness but not for bug finding!

$A(P) = [-7,0]$

$[0,7] = A([\![\mathsf{c}]\!]P)$

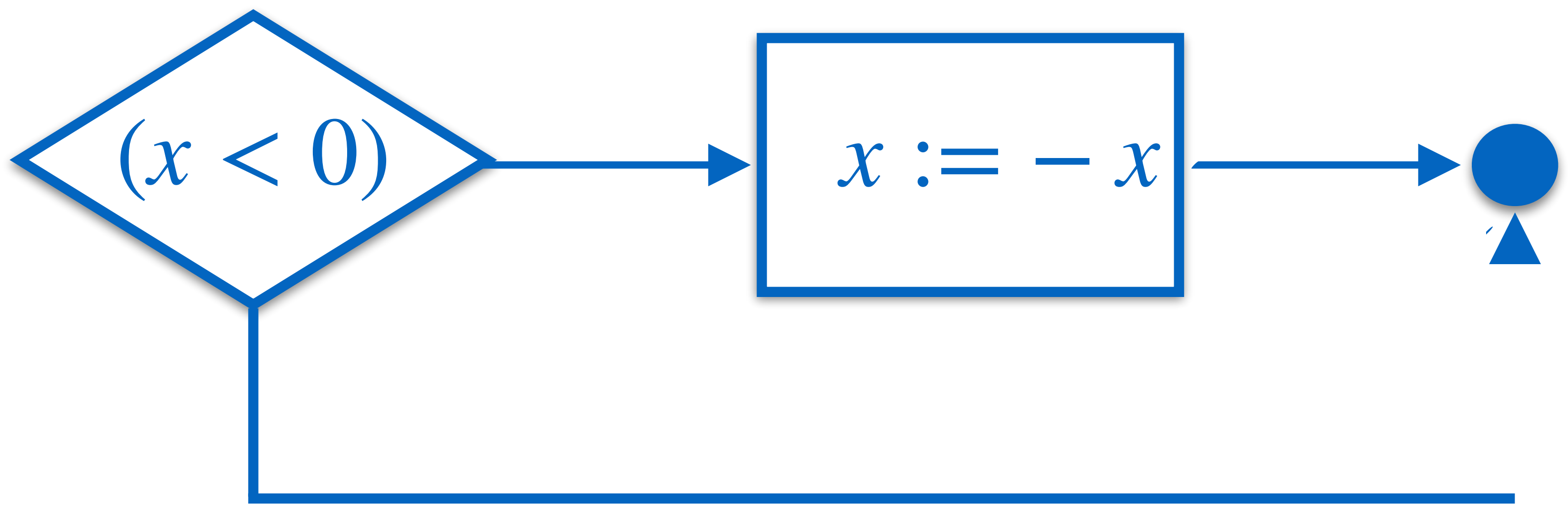
$P \subseteq A(P)$

$[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

$P = \{-7,0\}$



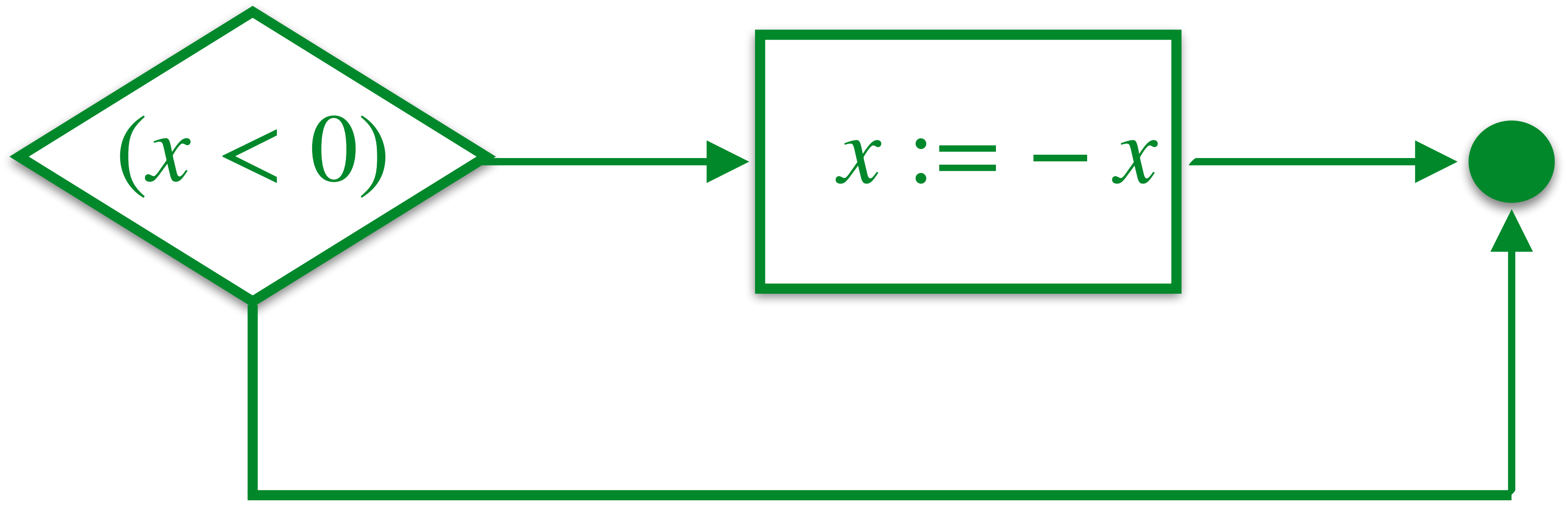
$\{0,7\} = [\![\mathsf{c}]\!]P$

An example of program analysis

over-approximations are
good for proving correctness
but not for bug finding!

Abstract (ex. Intervals)

$A(P) = [-7,0]$



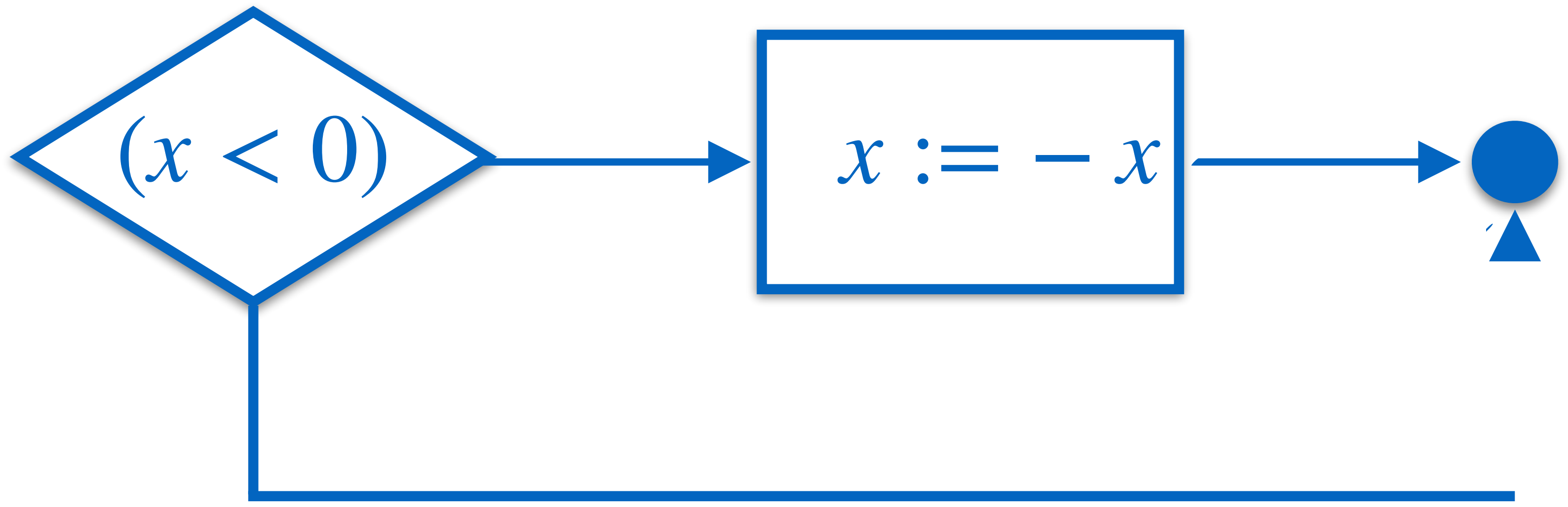
$[0,7] = A(\llbracket \mathsf{c} \rrbracket P)$

$P \subseteq A(P)$ $\qquad$ $\llbracket \mathsf{c} \rrbracket P \subseteq A(\llbracket \mathsf{c} \rrbracket P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

$P = \{-7,0\}$

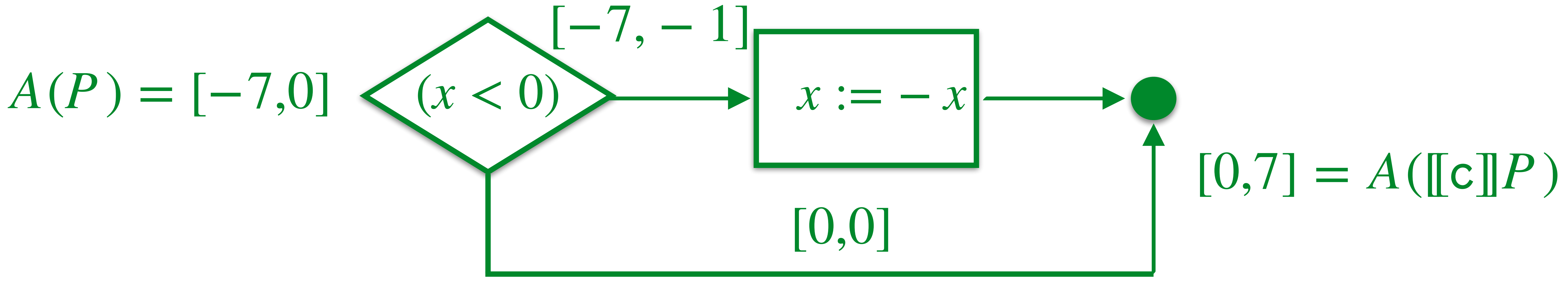


$\{0,7\} = \llbracket \mathsf{c} \rrbracket P$

An example of program analysis

Abstract (ex. Intervals)

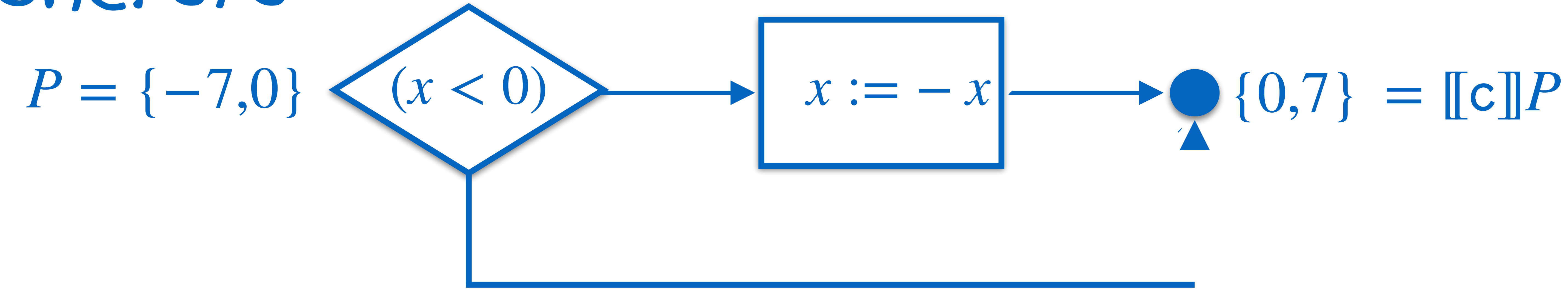
over-approximations are good for proving correctness but not for bug finding!



$P \subseteq A(P)$ $\quad$ $[\![c]\!]P \subseteq A([\![c]\!]P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

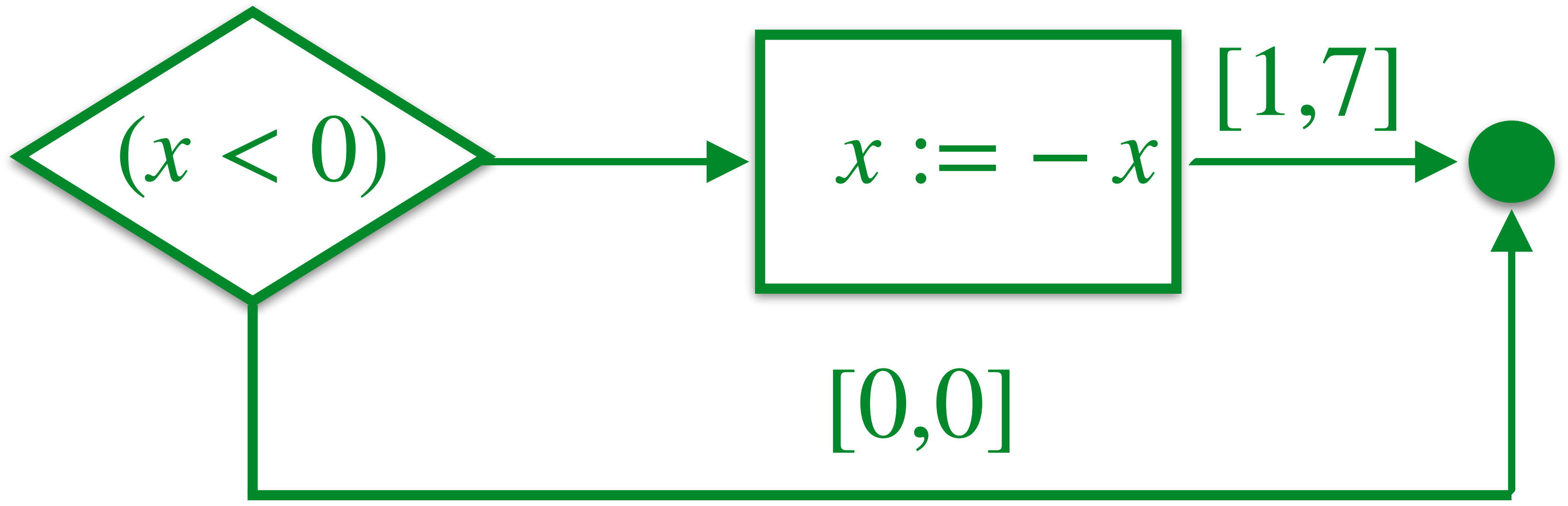


An example of program analysis

Abstract (ex. Intervals)

over-approximations are good for proving correctness but not for bug finding!

$A(P) = [-7,0]$



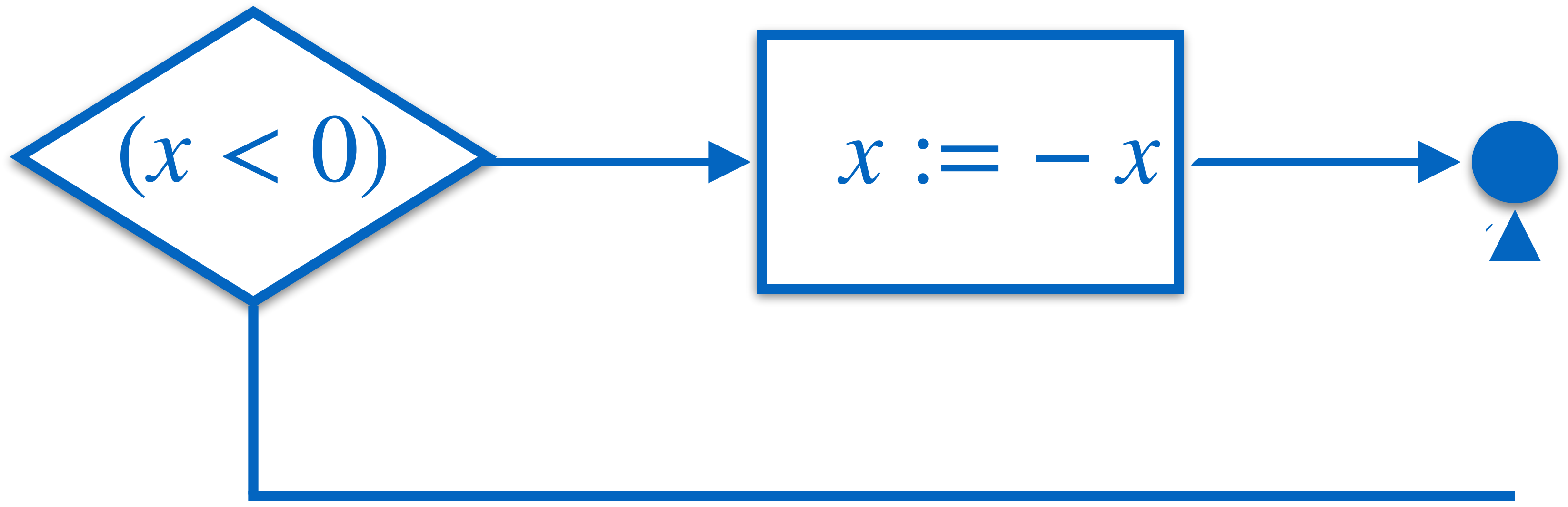
$[0,7] = A([\![\mathsf{c}]\!]P)$

$P \subseteq A(P)$ $\qquad$ $[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

$P = \{-7,0\}$



$\{0,7\} = [\![\mathsf{c}]\!]P$

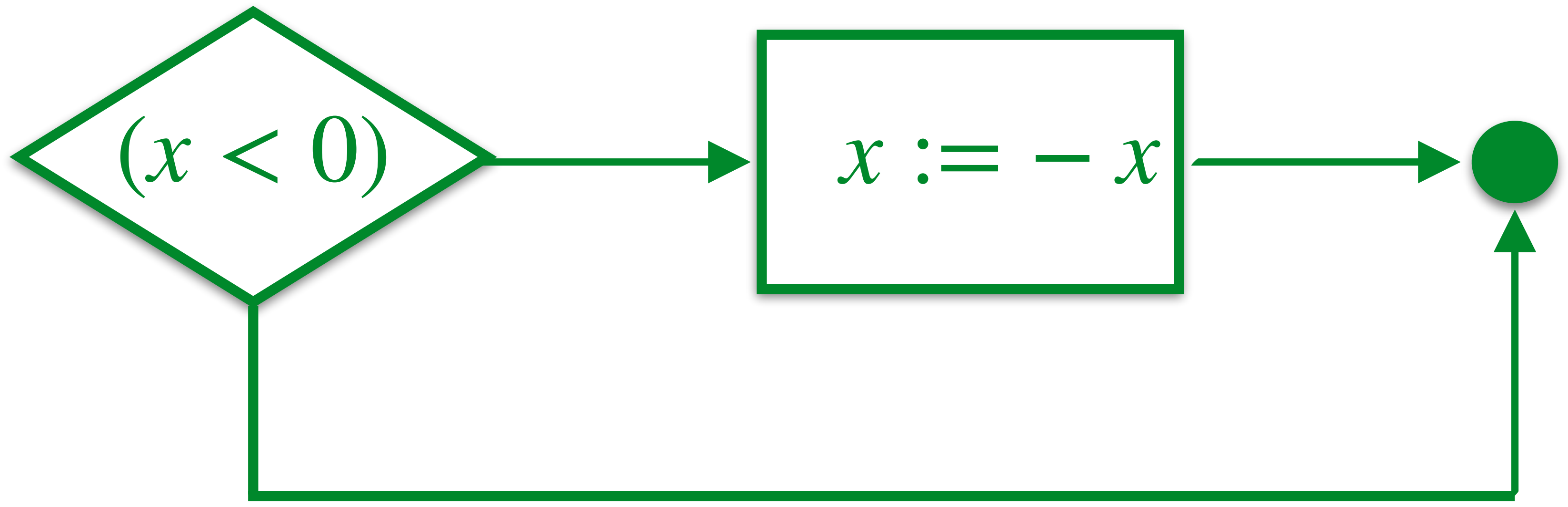
An example of program analysis

Abstract (ex. Intervals)

over-approximations are good for proving correctness

but not for bug finding!

$A(P) = [-7,0]$



$[0,7] = [\![\mathsf{c}]\!]^\sharp A(P)$

$[0,7] = A([\![\mathsf{c}]\!]P)$

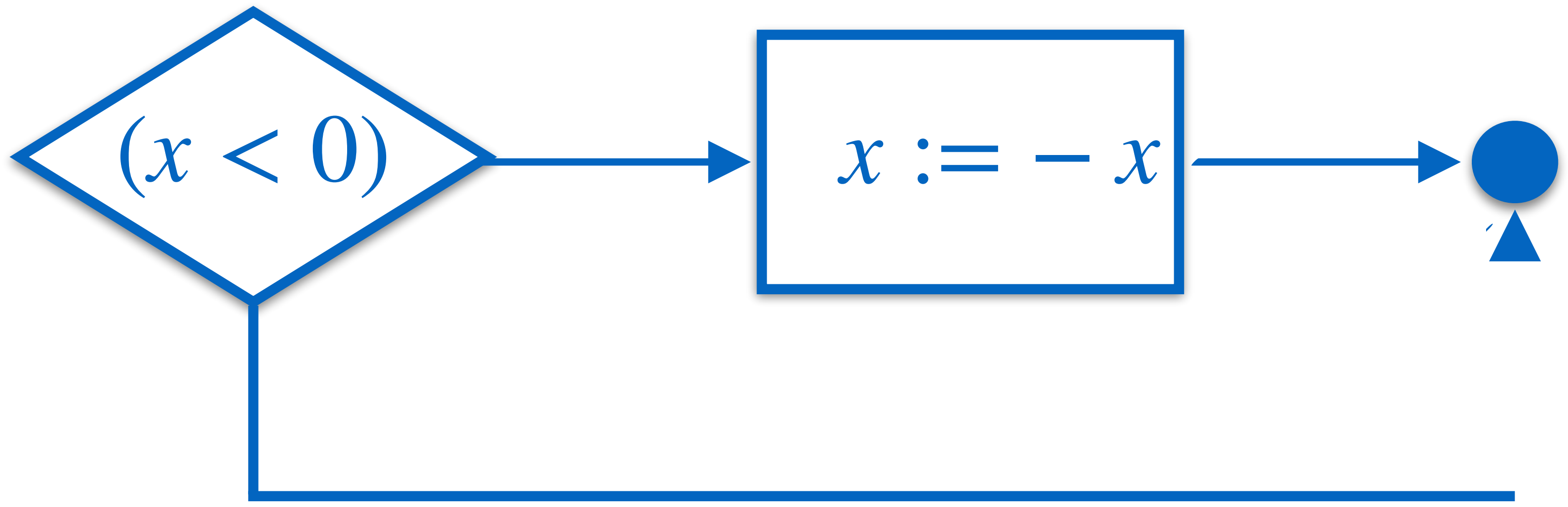
$P \subseteq A(P)$

$[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P) \subseteq [\![\mathsf{c}]\!]^\sharp A(P)$

$\mathsf{c} = \textsf{if } (x < 0) \textsf{ then } x := -x$

Concrete

$P = \{-7,0\}$



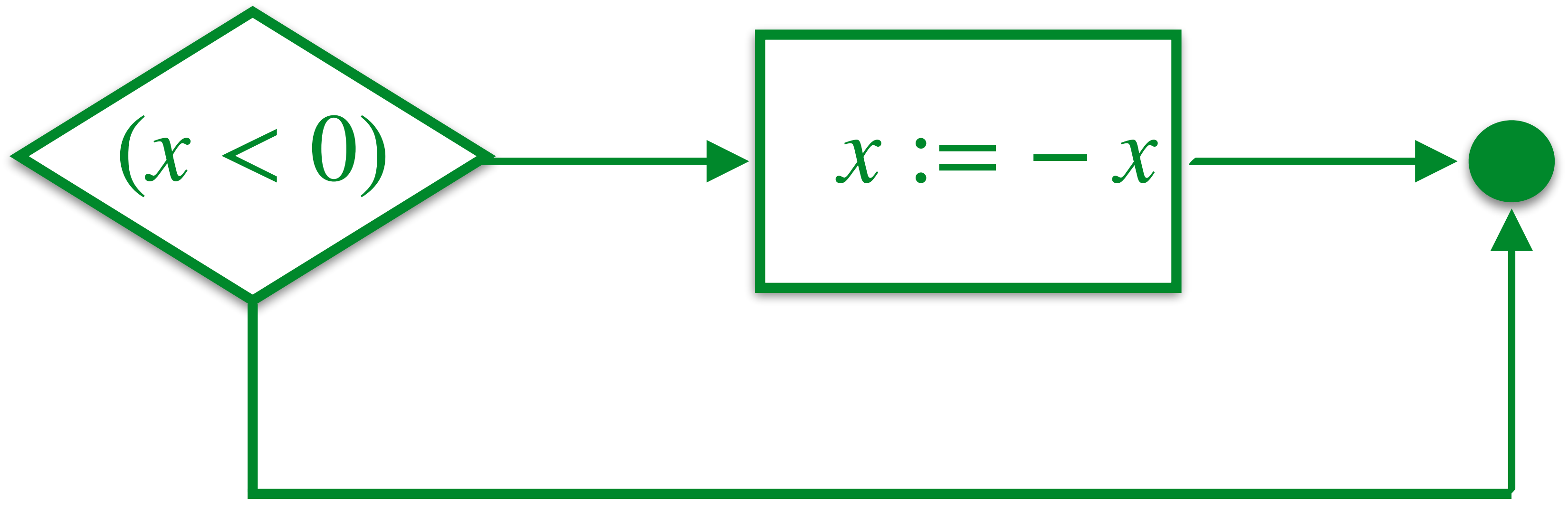
$\{0,7\} = [\![\mathsf{c}]\!]P$

An example of program analysis

Abstract (ex. Intervals)

over-approximations are good for proving correctness but not for bug finding!

$A(P) = [-7,0]$



$[0,7] = [\![\mathsf{c}]\!]^\sharp A(P)$

$[0,7] = A([\![\mathsf{c}]\!]P)$

$=$ Complete analysis!

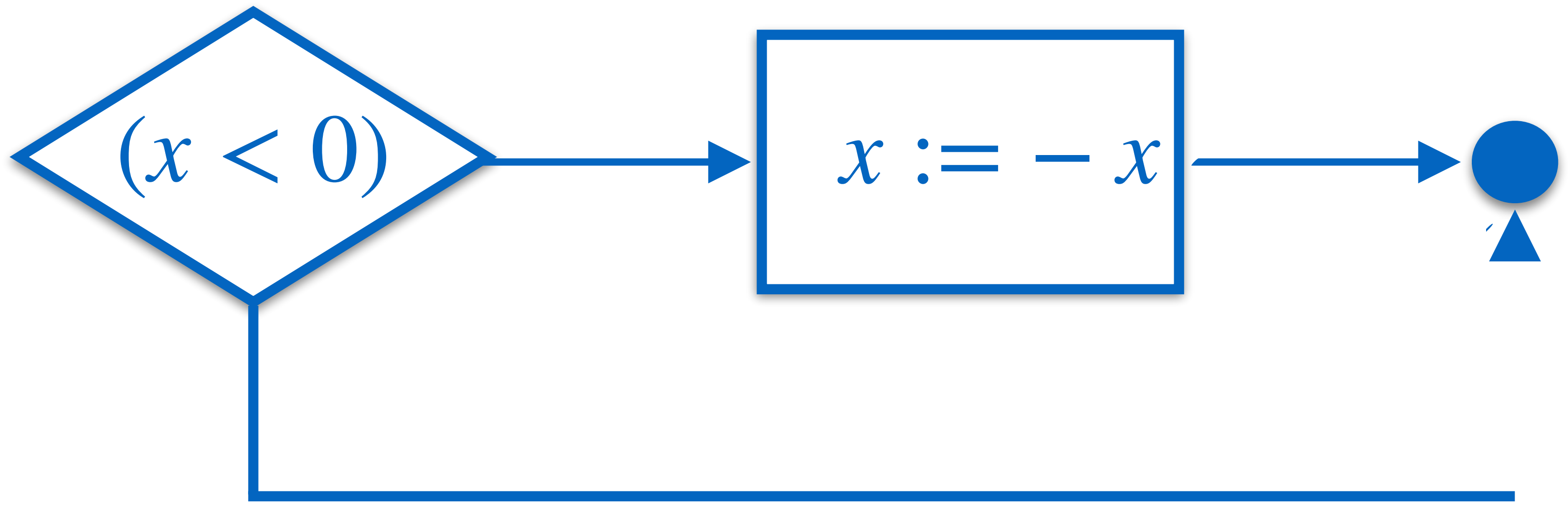
$P \subseteq A(P)$

$[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P) \subseteq [\![\mathsf{c}]\!]^\sharp A(P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

$P = \{-7,0\}$

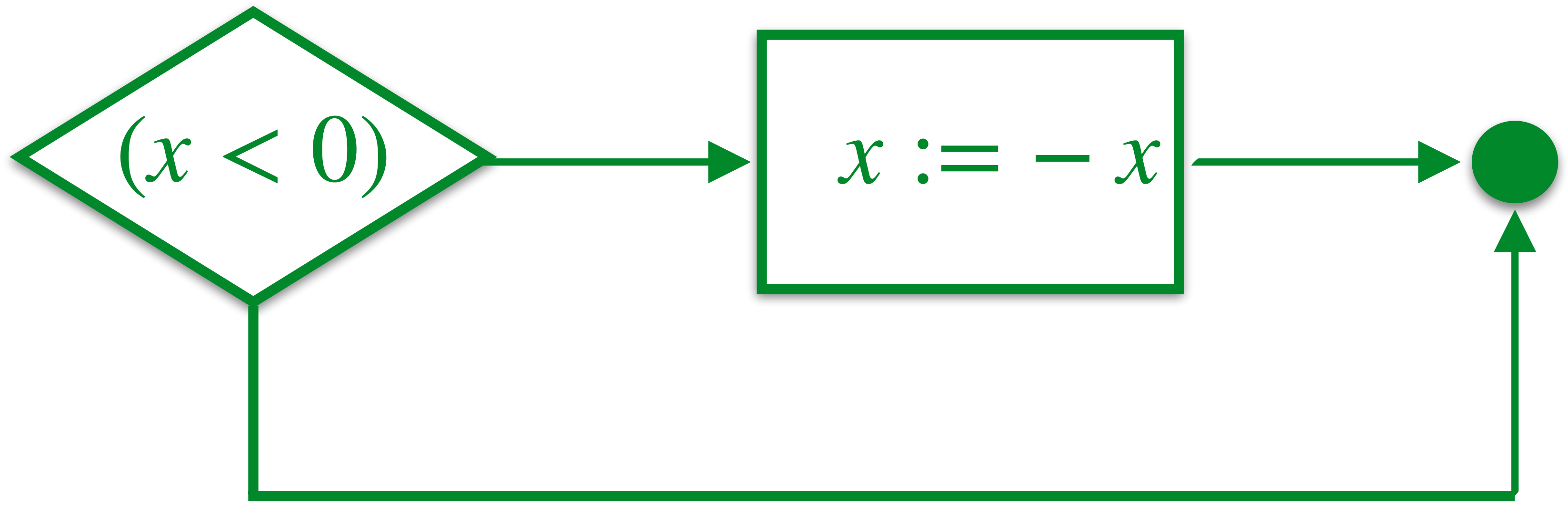


$\{0,7\} = [\![\mathsf{c}]\!]P$

Completeness vs Incompleteness

Abstract (ex. Intervals)

over-approximations are good for proving correctness but not for bug finding!

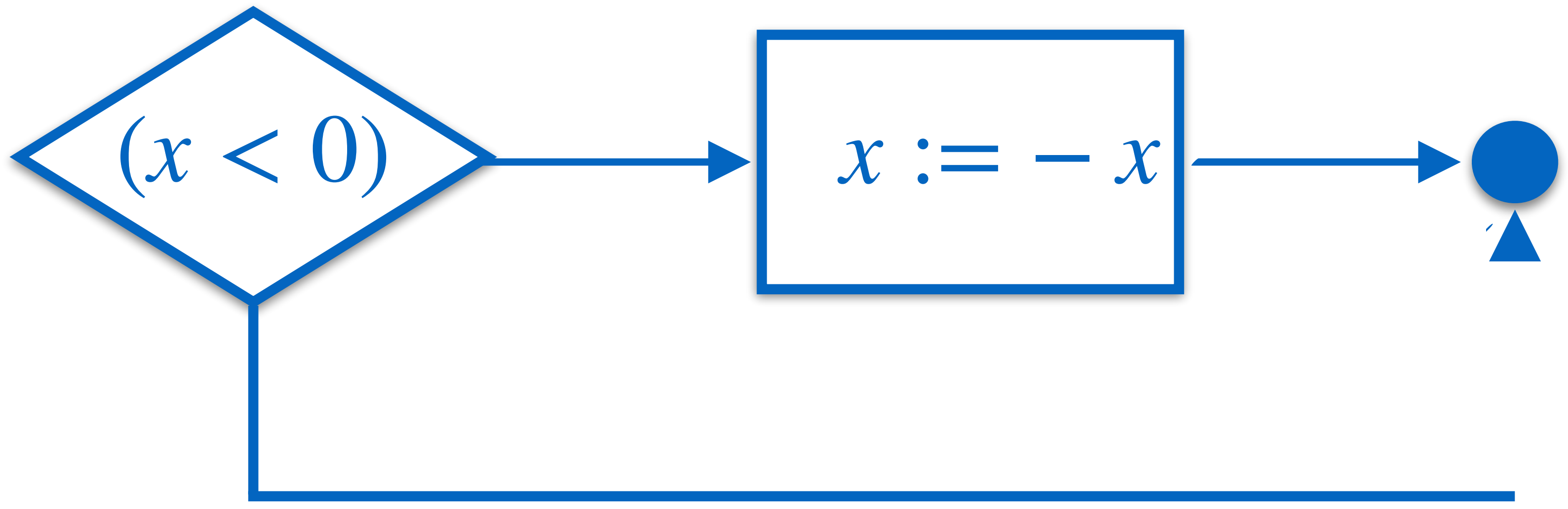


$P \subseteq A(P)$ $\quad$ $[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P) \subseteq [\![\mathsf{c}]\!]^\sharp A(P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

$P = \{-7,7\}$

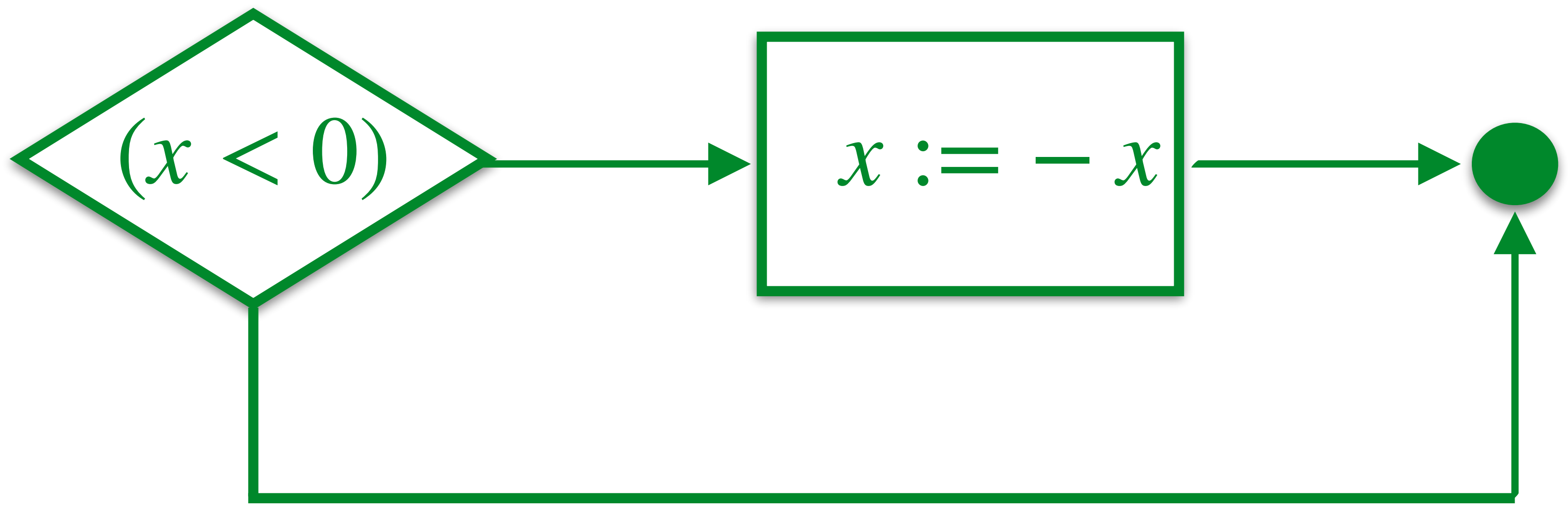


Completeness vs Incompleteness

Abstract (ex. Intervals)

over-approximations are good for proving correctness
but not for bug finding!

$A(P) = [-7,7]$



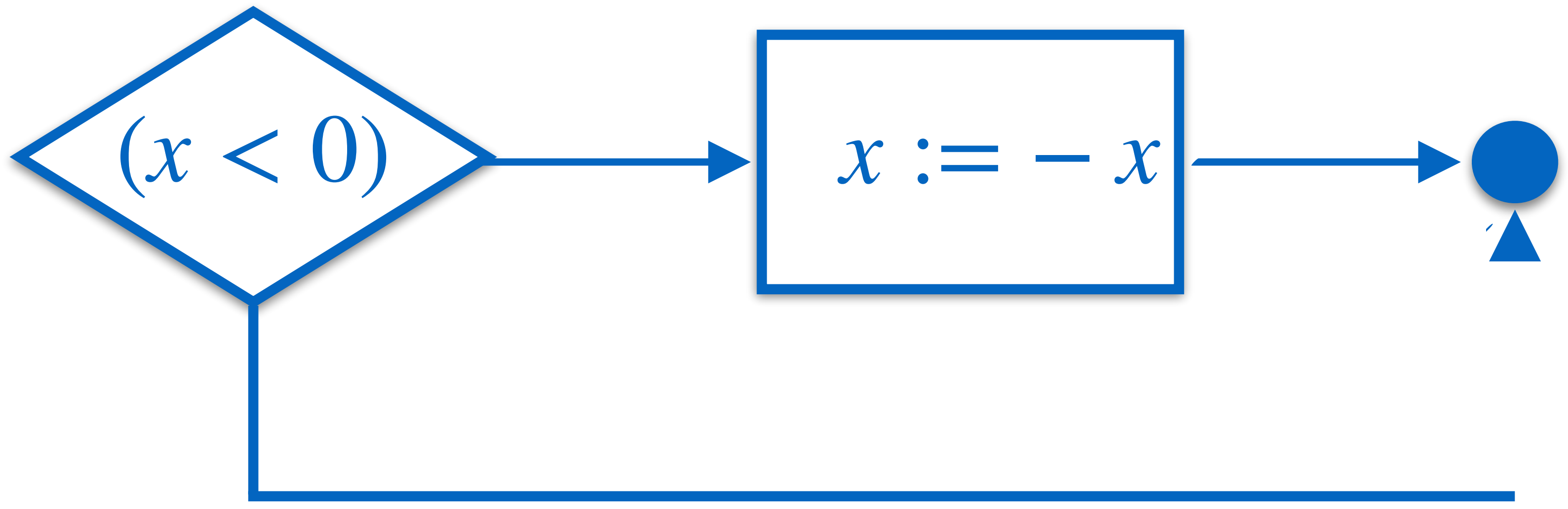
$P \subseteq A(P)$

$[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P) \subseteq [\![\mathsf{c}]\!]^\sharp A(P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

$P = \{-7,7\}$

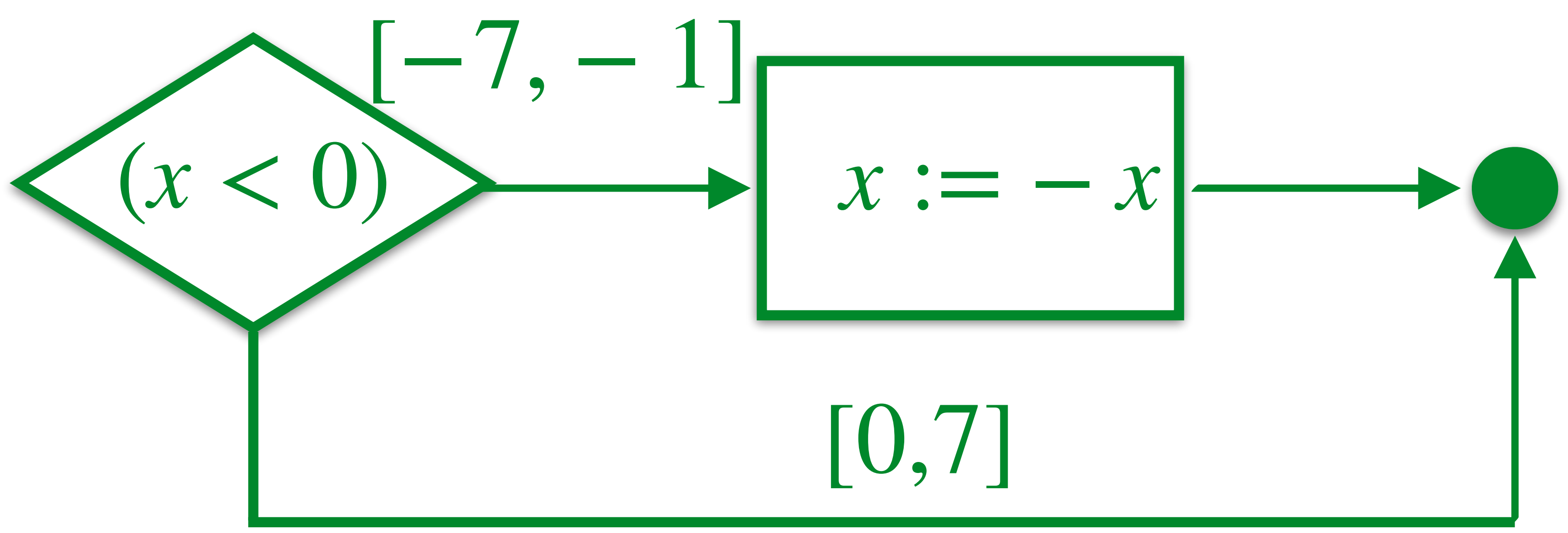


Completeness vs Incompleteness

Abstract (ex. Intervals)

over-approximations are good for proving correctness but not for bug finding!

$A(P) = [-7,7]$



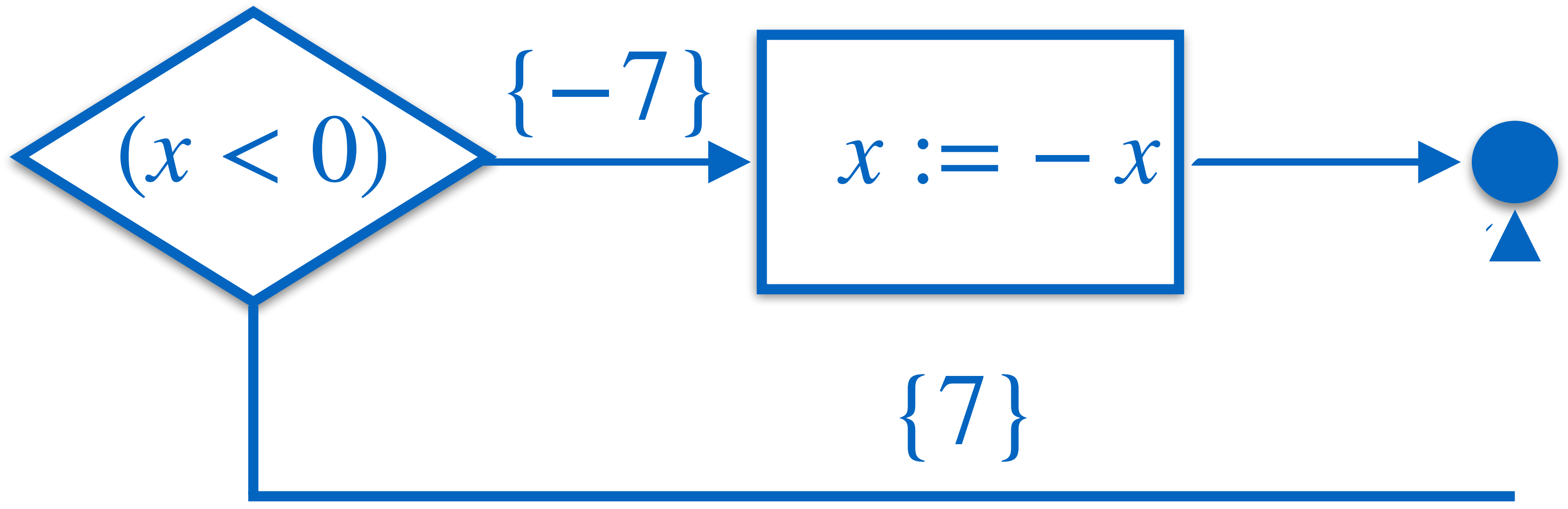
$P \subseteq A(P)$

$[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P) \subseteq [\![\mathsf{c}]\!]^{\sharp}A(P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

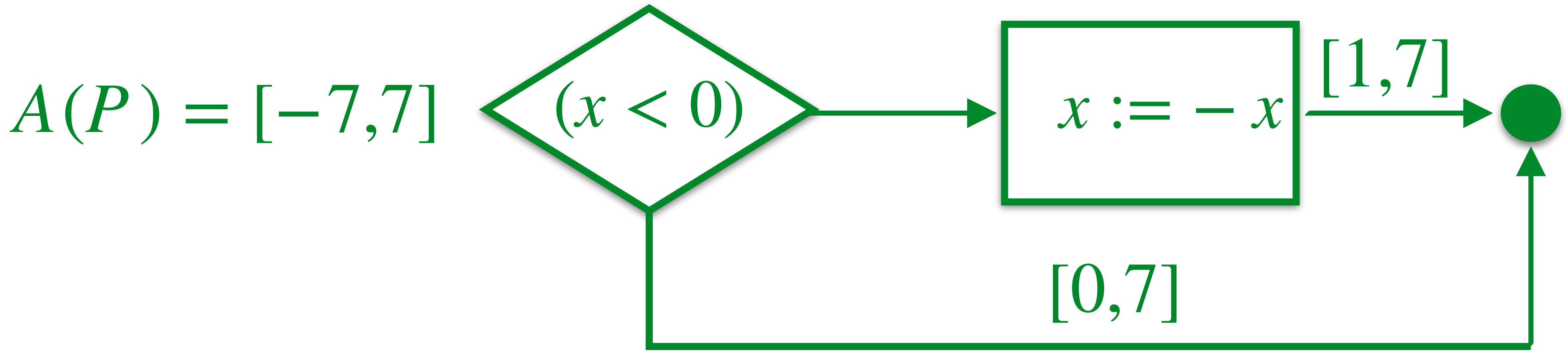
$P = \{-7,7\}$



Completeness vs Incompleteness

Abstract (ex. Intervals)

over-approximations are good for proving correctness
but not for bug finding!

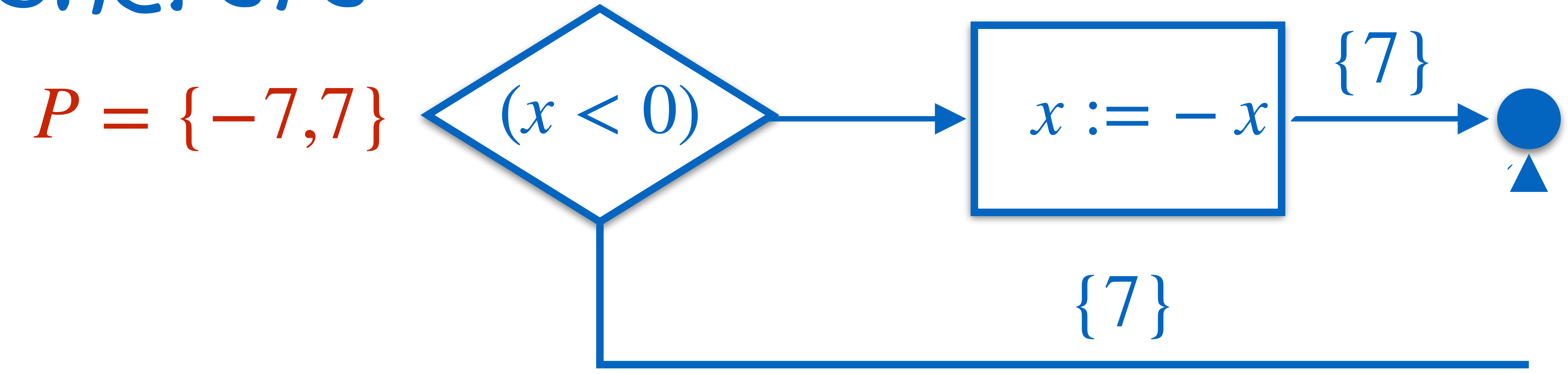


$P \subseteq A(P)$

$[\![c]\!]P \subseteq A([\![c]\!]P) \subseteq [\![c]\!]^\sharp A(P)$

$c = \text{if } (x < 0) \text{ then } x := -x$

Concrete

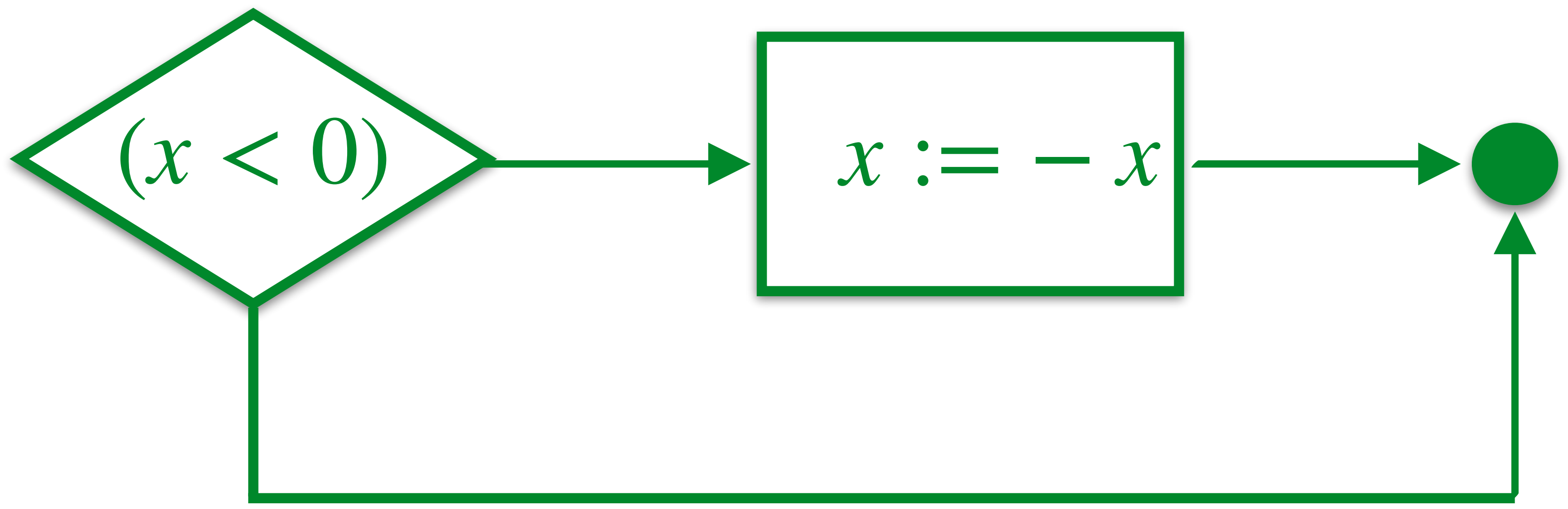


Completeness vs Incompleteness

over-approximations are good for proving correctness but not for bug finding!

Abstract (ex. Intervals)

$A(P) = [-7,7]$



$[0,7] = [\![\mathsf{c}]\!]^\sharp A(P)$

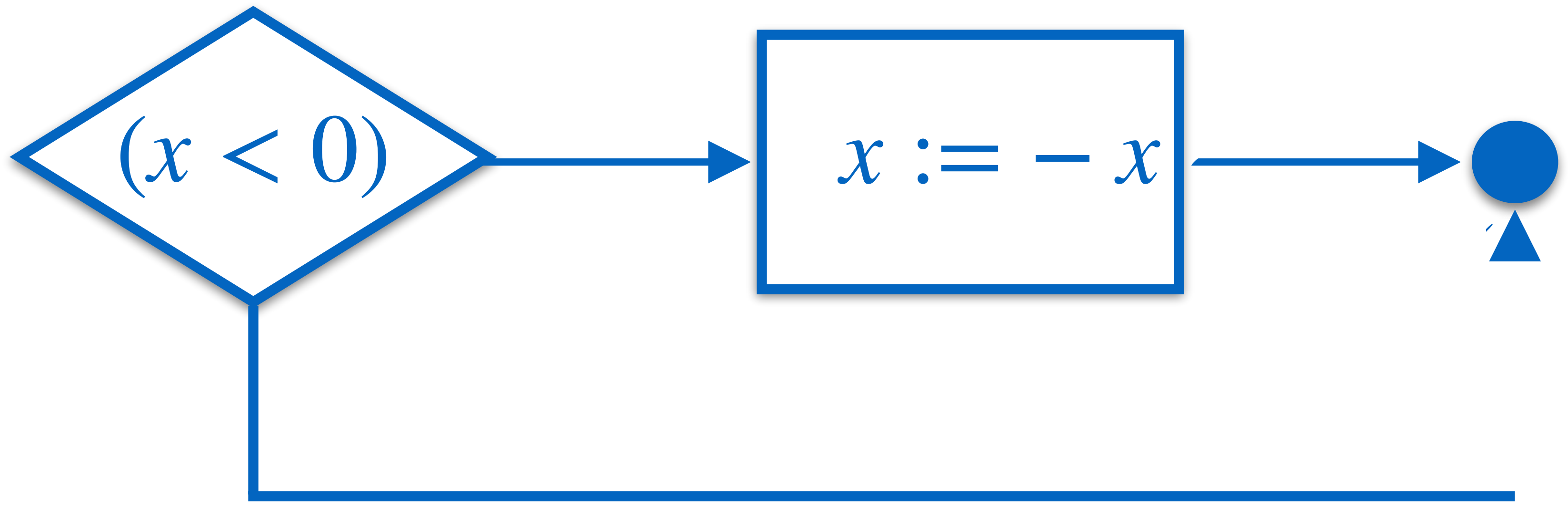
$P \subseteq A(P)$

$[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P) \subseteq [\![\mathsf{c}]\!]^\sharp A(P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

$P = \{-7,7\}$



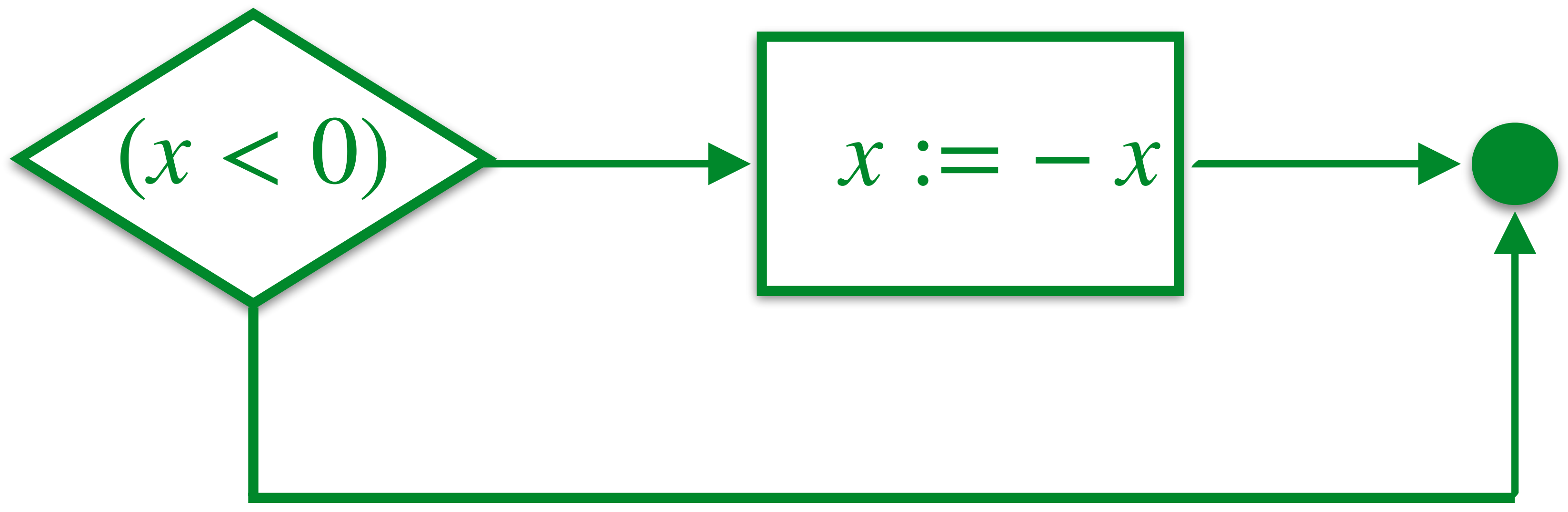
$\{7\} = [\![\mathsf{c}]\!]P$

Completeness vs Incompleteness

Abstract (ex. Intervals)

over-approximations are good for proving correctness but not for bug finding!

$A(P) = [-7,7]$



$[0,7] = [\![\mathsf{c}]\!]^\sharp A(P)$

$[7,7] = A([\![\mathsf{c}]\!]P)$

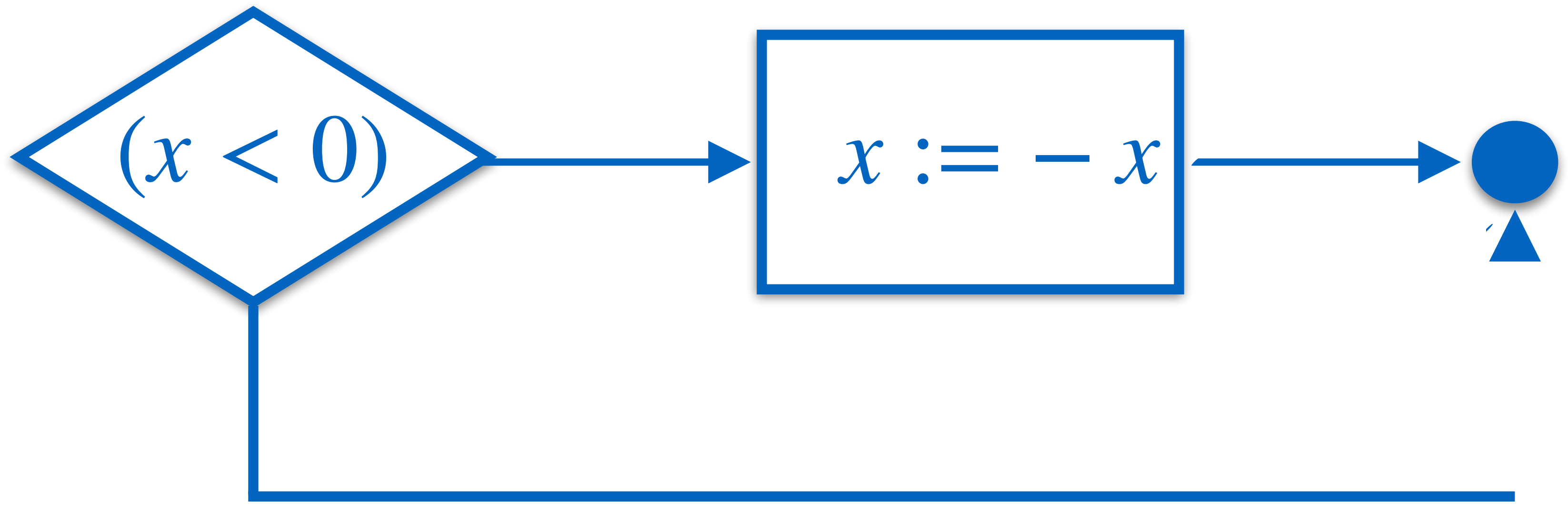
$P \subseteq A(P)$

$[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P) \subseteq [\![\mathsf{c}]\!]^\sharp A(P)$

$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$

Concrete

$P = \{-7,7\}$



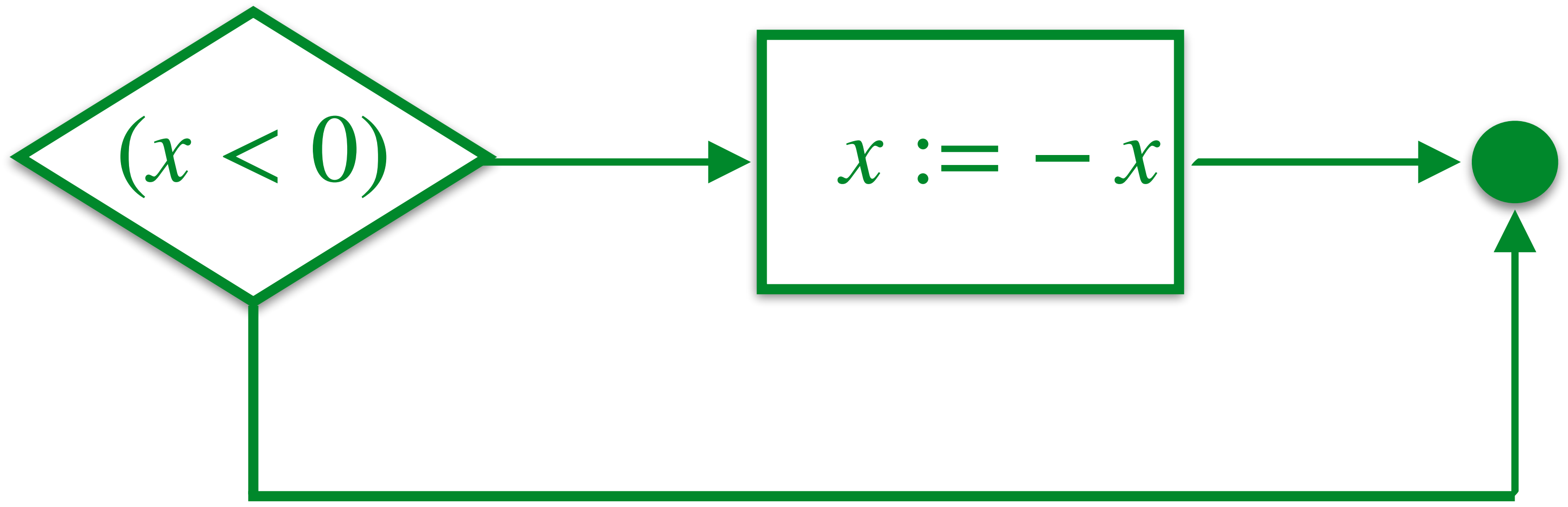
$\{7\} \quad = [\![\mathsf{c}]\!]P$

Completeness vs Incompleteness

Abstract (ex. Intervals)

over-approximations are good for proving correctness but not for bug finding!

$A(P) = [-7,7]$



$[0,7] = [\![\mathsf{c}]\!]^\sharp A(P)$

$[7,7] = A([\![\mathsf{c}]\!]P)$

$\neq$ Incomplete analysis!

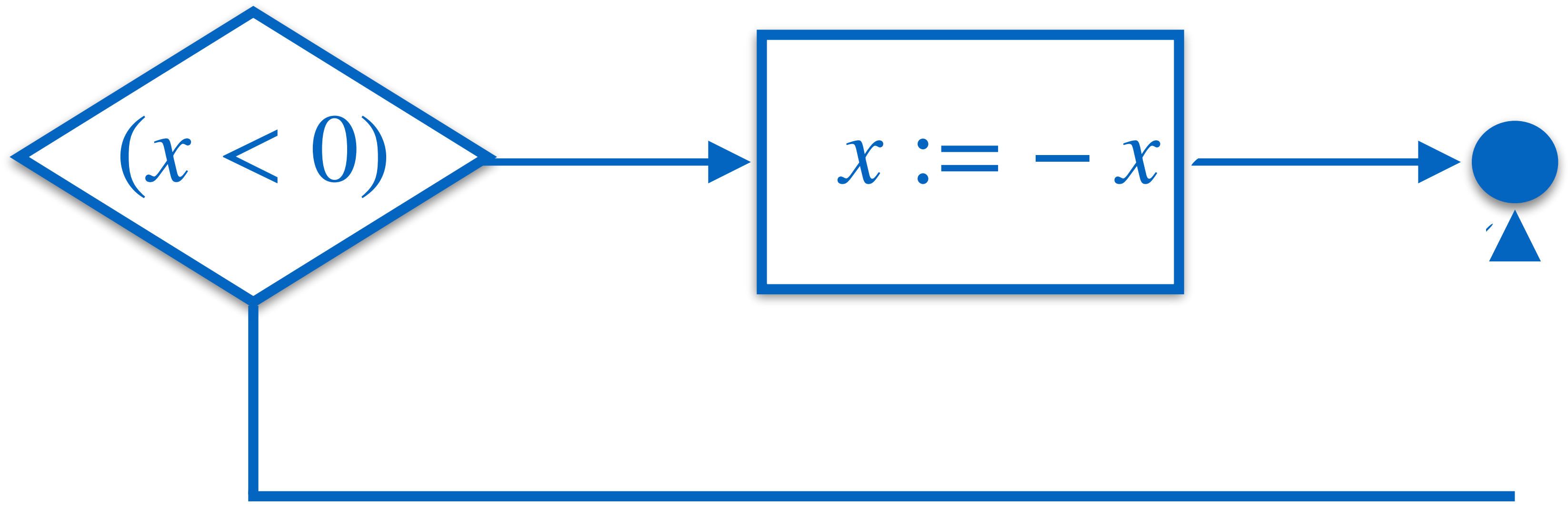
$P \subseteq A(P)$

$$[\![\mathsf{c}]\!]P \subseteq A([\![\mathsf{c}]\!]P) \subseteq [\![\mathsf{c}]\!]^\sharp A(P)$$

$$\mathsf{c} = \mathsf{if}\ (x < 0)\ \mathsf{then}\ x := -x$$

Concrete

$P = \{-7,7\}$



$\{7\} \quad = [\![\mathsf{c}]\!]P$

Completeness + Turing Equivalence = Trivial domains

$A = Id \ \lor \ A = \lambda S.\top$

Th. Only trivial abstractions are complete for Turing equivalent programming languages

Th. Completeness depends only on the basic expressions allowed in the syntax (guards and assignments)

e.g. Int is (non-trivial and) complete for any assignment

POPL 2015
Mumbai, India
January 12-18

Roberto Giacobazzi, Francesco Logozzo, Francesco Ranzato: **Analyzing Program Analyses**. POPL 2015: 261-273

Sources of Incompleteness and Local Completeness

Guards are hard to handle

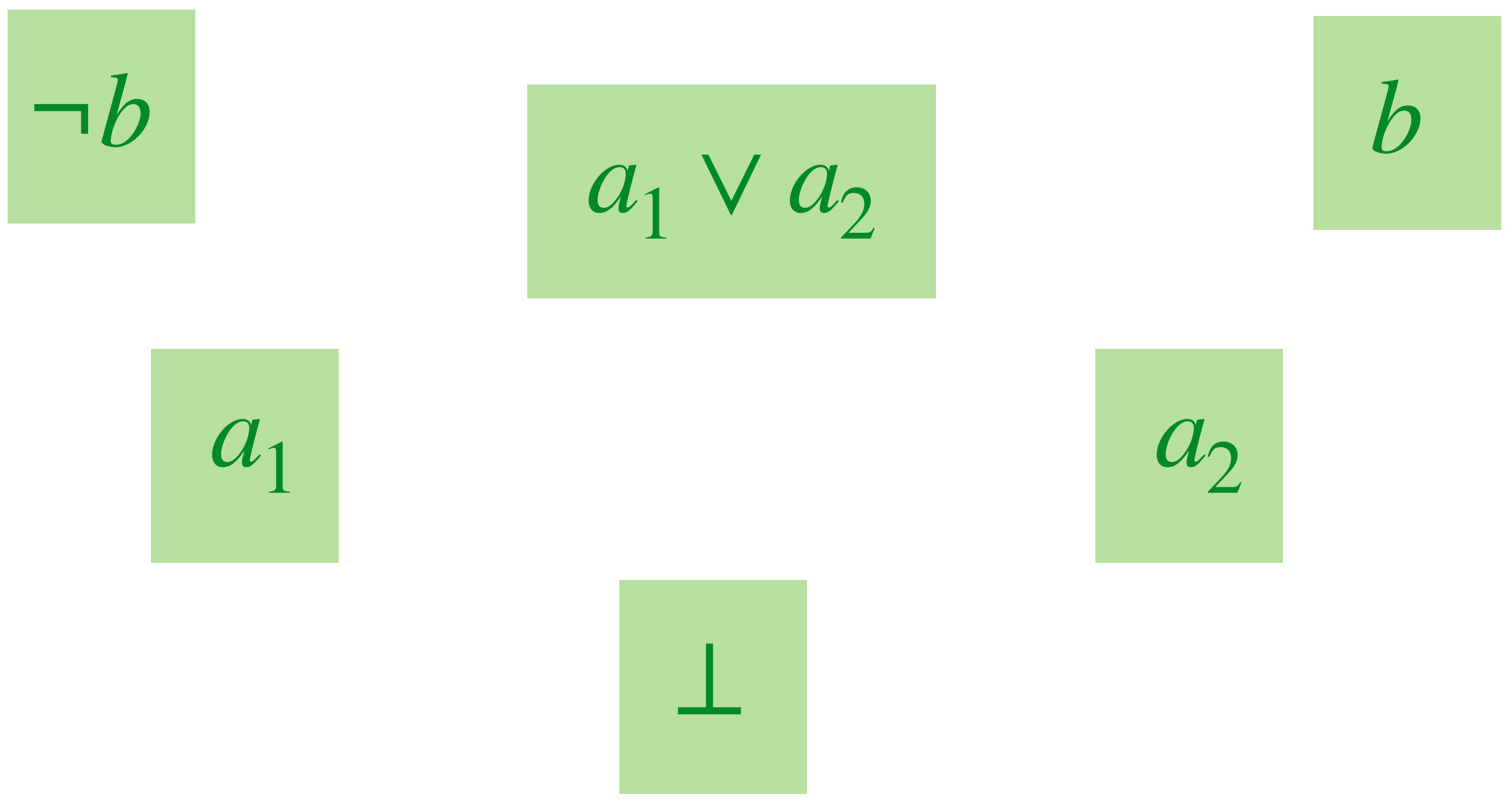
Th. A domain can be complete only for guards that are expressible in it

e.g. Int cannot be complete for $x=0$
(because $(x\neq 0)$ is not expressible in Int)

What if we add the abstract point $x\neq 0$?
After Moore closure: intervals with or without holes in zero
The domain $\mathrm{Int}_{\neq 0}$ is no longer complete for sums!

Necessary and sufficient conditions for guard completeness

Th. A domain is complete for b?/¬b? iff it contains any union of abstract points below b/¬b



Necessary and sufficient conditions for guard completeness

Th. A domain is complete for $b?/\neg b?$ iff it contains any union of abstract points below $b/\neg b$

e.g. Int is not complete for $x\leq 0/x>0$
(because $x<-2$ is expressible and below $x\leq 0$
$x>2$ is expressible and below $x>0$
but ($x<-2$ or $x>2$) is not expressible in Int)

Local Completeness

We must abandon the ambition of being complete
for every program and for every input!

Local completeness is about a given program and a given input

Global completeness

$$A\,f = A\,f\,A$$

$$\forall c\,.\,A\,f\,(c) = A\,f\,A\,(c)$$

Local completeness at c

$$A\,f\,(c) = A\,f\,A\,(c)$$

$$\mathbb{C}^A_c(f)$$

Locally Complete Analysis

Th. Verification

under-approx + local completeness + spec expressible

$$Q \leq [\![\mathsf{c}]\!]P \;\wedge\; [\![\mathsf{c}]\!]^\sharp A(P) = A(Q) \;\wedge\; S = A(S) \;\Rightarrow\; (\, [\![\mathsf{c}]\!]P \leq S \Leftrightarrow Q \leq S \,)$$

$$Q \leq [\![\mathsf{c}]\!]P \quad \Rightarrow \quad A(Q) \leq A([\![\mathsf{c}]\!]P) \leq [\![\mathsf{c}]\!]^\sharp A(P)$$

$$\wedge\; [\![\mathsf{c}]\!]^\sharp A(P) = A(Q) \quad \Rightarrow \quad A([\![\mathsf{c}]\!]P) = [\![\mathsf{c}]\!]^\sharp A(P)$$

$$\wedge\; S = A(S) \quad \Rightarrow \quad [\![\mathsf{c}]\!]P \leq S \;\Leftrightarrow\; A([\![\mathsf{c}]\!]P) \leq A(S) = S$$

$$\Leftrightarrow \quad A(Q) \leq A(S) = S \;\Leftrightarrow\; Q \leq S$$

Local Completeness Logic
(LCL)

The Rules of LCL

Local completeness proof obligations

$$\frac{\mathbb{C}^A_P(\mathsf{e})}{\vdash_A [P]\ \mathsf{e}\ [[\![\mathsf{e}]\!]P]}\ (\text{transfer})$$

$$\frac{\vdash_A [P]\ \mathsf{r}_1\ [R] \quad \vdash_A [R]\ \mathsf{r}_2\ [Q]}{\vdash_A [P]\ \mathsf{r}_1;\mathsf{r}_2\ [Q]}\ (\text{seq})$$

$$\frac{\vdash_A [P]\ \mathsf{r}\ [R] \quad \vdash_A [P \vee R]\ \mathsf{r}^*\ [Q]}{\vdash_A [P]\ \mathsf{r}^*\ [Q]}\ (\text{rec})$$

Key (consequence) rule

$$\frac{P' \leq P \leq A(P') \quad \vdash_A [P']\ \mathsf{r}\ [Q'] \quad Q \leq Q' \leq A(Q)}{\vdash_A [P]\ \mathsf{r}\ [Q]}\ (\text{relax})$$

$$\frac{\vdash_A [P]\ \mathsf{r}_1\ [Q_1] \quad \vdash_A [P]\ \mathsf{r}_2\ [Q_2]}{\vdash_A [P]\ \mathsf{r}_1 \oplus \mathsf{r}_2\ [Q_1 \vee Q_2]}\ (\text{join})$$

$$\frac{\vdash_A [P]\ \mathsf{r}\ [Q] \quad Q \leq A(P)}{\vdash_A [P]\ \mathsf{r}^*\ [P \vee Q]}\ (\text{iterate})$$

Fixpoint acceleration by abstract interpretation

Th. Logical Soundness

$$\vdash_A [P]\ \mathsf{c}\ [Q] \ \Rightarrow\ Q \leq [\![\mathsf{c}]\!]P \leq [\![\mathsf{c}]\!]^\sharp A(P) = A(Q)$$

The Proof System (compositional reasoning!)

Proof obligations: local completeness for basic expressions
(all the other rules preserve local completeness)

$$\frac{A(\llbracket \mathsf{e} \rrbracket P) = A(\llbracket \mathsf{e} \rrbracket A(P))}{\vdash_A [P]\ \mathsf{e}\ [\llbracket \mathsf{e} \rrbracket P]}\ \text{(transfer)}$$

$$\frac{\text{locally complete!}}{\vdash_{\mathsf{Int}} [\{-7,7\}]\ x := x+1\ [\{-6,8\}]}\ \text{(transfer)}\ \textit{assignments}$$

$$\frac{\text{not locally complete!}}{\vdash_{\mathsf{Int}} [\{-7,7\}]\ \mathsf{x} < 0?\ [\{-7\}]}\ \text{(transfer)}\ \textit{guards}$$

The Proof System (compositional reasoning!)

Proof obligations: local completeness for basic expressions
(all the other rules preserve local completeness)

$$\frac{A([\![\mathsf{e}]\!]P) = A([\![\mathsf{e}]\!]A(P))}{\vdash_A [P]\ \mathsf{e}\ [[\![\mathsf{e}]\!]P]}\ \text{(transfer)}$$

$$\frac{\text{locally complete!}}{\vdash_{\mathsf{Int}} [\{-7,7\}]\ x := x + 1\ [\{-6,8\}]}\ \text{(transfer)}$$

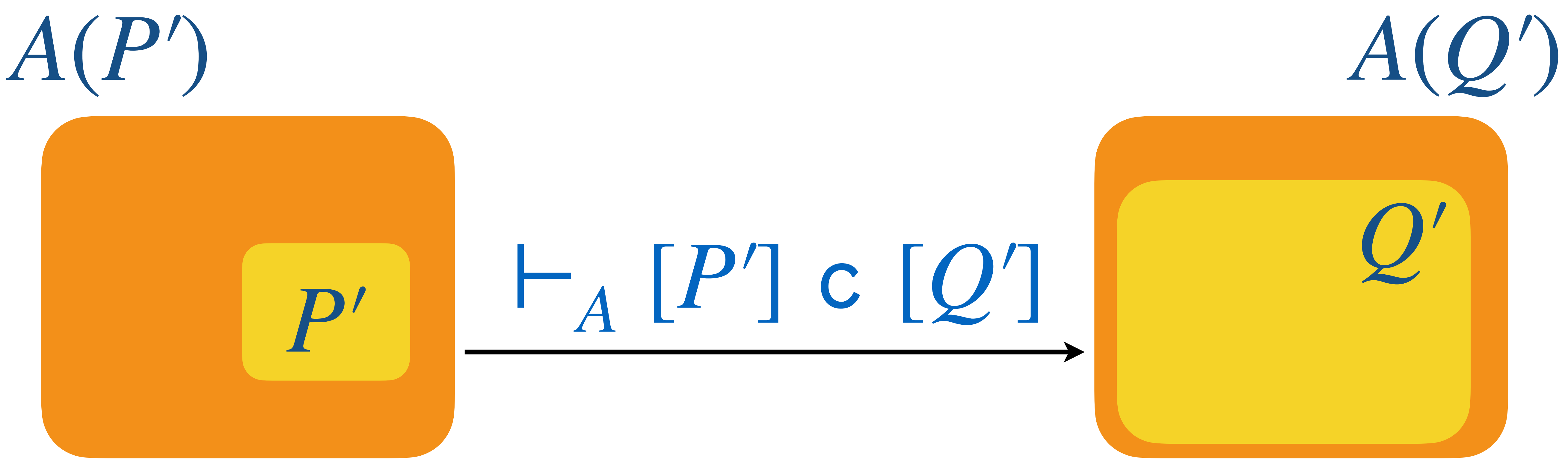
assignments

$$\frac{\text{locally complete!}}{\vdash_{\mathsf{Int}} [\{-7,-1,7\}]\ \mathsf{x} < 0?\ [\{-7,-1\}]}\ \text{(transfer)}$$

guards

The key new consequence rule

$$\frac{P' \leq P \leq A(P') \qquad \vdash_A [P']\ \mathsf{c}\ [Q'] \qquad Q \leq Q' \leq A(Q)}{\vdash_A [P]\ \mathsf{c}\ [Q]}\ \text{(relax)}$$

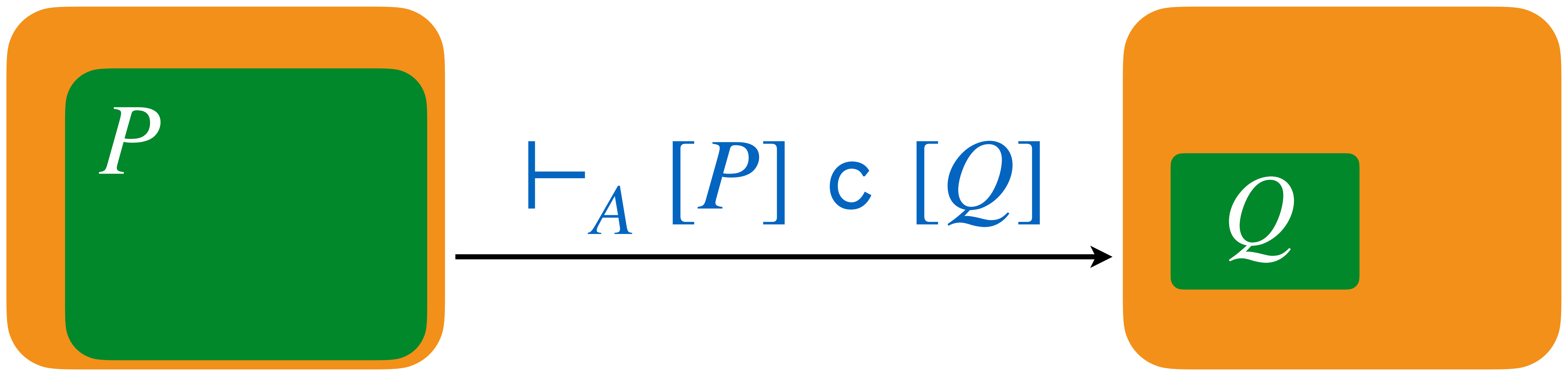


The key new consequence rule

$$\frac{P' \leq P \leq A(P') \qquad \vdash_A [P'] \; \mathsf{c} \; [Q'] \qquad Q \leq Q' \leq A(Q)}{\vdash_A [P] \; \mathsf{c} \; [Q]} \; \text{(relax)}$$

we can weaken the precondition... and strengthen the postcondition as far as we preserve their abstractions

$A(P') = A(P)$ $\qquad$ $A(Q) = A(Q')$



Fixpoint acceleration

$$\frac{\vdash_A [P]\ \mathsf{r}\ [Q] \quad Q \leq A(P)}{\vdash_A [P]\ \mathsf{r}^*\ [P \vee Q]}\ (\text{iterate})$$

$$\dfrac{\dfrac{\dfrac{\mathbb{C}^{\mathsf{Sign}}_{P}(\llbracket x \leq 0? \rrbracket)}{\vdash_{\mathsf{Sign}} [P]\ x \leq 0?\ [\{-10,-1\}]}\ (\text{transfer}) \quad \dfrac{\mathbb{C}^{\mathsf{Sign}}_{\{-10,-1\}}(\llbracket x := x * 10 \rrbracket)}{\vdash_{\mathsf{Sign}} [\{-10,-1\}]\ x := x * 10\ [\{-100,-10\}]}\ (\text{transfer})}{\vdash_{\mathsf{Sign}} [P]\ x \leq 0?; x := x * 10\ [\{-100,-10\}]}\ (\text{seq}) \quad \{-100,-10\} \subseteq \mathsf{Sign}(P) = \mathbb{Z}_{\neq 0}}{\vdash_{\mathsf{Sign}} [P]\ (x \leq 0?; x := x * 10)^*\ [\{-100,-10,-1,100\}]}\ (\text{iterate})$$

$$P \triangleq \{-10,-1,100\}$$

Main results

Th. Verification

any provable triple either shows the program correct or exposes some error

correctness + bug finding!

Th. Logical Completeness

if the abstraction is complete for every basic expressions in the program, then any valid triple is provable

Th. Intrinsic Incompleteness

for any Turing complete language and any non-trivial abstraction, there are valid triples that cannot be proved

Combine Abstract Domains

Proof obligations and domain refinement

Suppose $\vdash_{A_1} [P]\ \mathsf{r}_1\ [R]$ and $\vdash_{A_2} [R]\ \mathsf{r}_2\ [Q]$

Can we conclude $\vdash_{A} [P]\ \mathsf{r}_1\ ;\ \mathsf{r}_2\ [Q]$ for some A?

The reduced product domain $A = A_1 \sqcap A_2$ may not work

Idea: combine more abstract domains in the same derivation

The rule refine

take a more precise domain

preserve abstractions of pre-conditions

$$\frac{A' \preceq A \quad A'(P) = A(P) \quad \vdash_{A'} [P]\ \mathsf{c}\ [Q]}{\vdash_A [P]\ \mathsf{c}\ [Q]}\ (\text{refine})$$

Th. Logical Soundness

$$\vdash_A [P]\ \mathsf{c}\ [Q] \ \Rightarrow\ Q \leq [\![\mathsf{c}]\!]P \leq A(Q)$$

extensional analysis: cannot guarantee that

$$[\![\mathsf{c}]\!]^\sharp A(P) = A(Q)$$

Example

$$\mathsf{r}_1 \triangleq \quad \mathtt{y} := 2 * \mathtt{y} + 1; \quad \mathtt{y} := \mathtt{abs}(\mathtt{y})$$

$$\mathsf{r}_2 \triangleq \quad \mathtt{x} := \mathtt{y}; \quad \mathtt{while}(\mathtt{x} > 1)\{\mathtt{y} := \mathtt{y} - 1; \mathtt{x} := \mathtt{x} - 1\}$$

$$\dfrac{\dfrac{\vdash_{\mathsf{Int}_{\neq 0}} [P]\ \mathsf{r}_1\ [S]}{\vdash_{\mathsf{Int}} [P]\ \mathsf{r}_1\ [S]}\text{(refine)} \qquad \dfrac{\vdash_{\mathsf{Oct}} [S]\ \mathsf{r}_2\ [Q]}{\vdash_{\mathsf{Int}} [S]\ \mathsf{r}_2\ [Q]}\text{(refine)}}{\vdash_{\mathsf{Int}} [P]\ \mathsf{r}_1; \mathsf{r}_2\ [Q]}\text{(seq)}$$

$$P \triangleq (y \in [-100; 100]) \qquad S \triangleq (y \in \{1; 201\}) \qquad Q \triangleq (x = y = 1)$$

$[\![\mathsf{r}_1; \mathsf{r}_2]\!]^{\#}_{\mathsf{Int}}$ computes $x = 1 \wedge 0 \leq y \leq 100$

Abstract Interpretation Repair
(AIR)

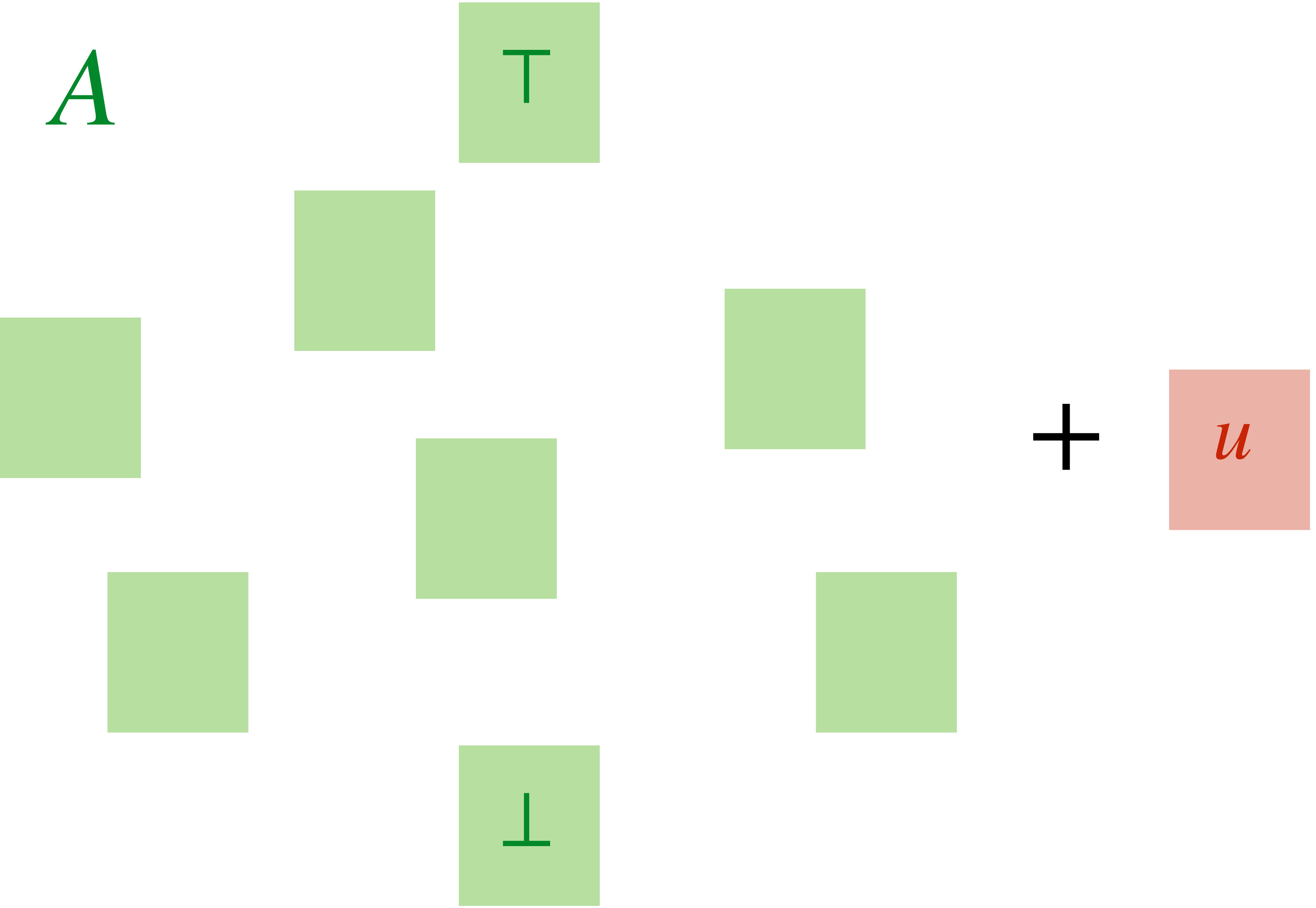
Proof obligations and domain refinement

What if a derivation fails?

e.g. a local completeness proof obligation $\mathbb{C}^A_R(e)$ fails

Idea: Refine the domain and restart the analysis

A



Add new element

+

Moore closure

$$A_u(c) = \begin{cases} u \wedge A(c) & \text{if } c \leq u \\ A(c) & \text{owise} \end{cases}$$

Refinement (as closure)

$\wp(\{1,2,3,4,5\})$

LessMoreThan3

$\{1,2,3,4,5\}$

$\{1,2,3,4\}$ $\{1,2,3,5\}$ $\{1,2,4,5\}$ $\{1,3,4,5\}$ $\{2,3,4,5\}$

$\{1,2,3\}$ $\{1,2,4\}$ $\{1,2,5\}$ $\{1,3,4\}$ $\{1,3,5\}$ $\{1,4,5\}$ $\{2,3,4\}$ $\{2,3,5\}$ $\{2,4,5\}$ $\{3,4,5\}$

$\{1,2\}$ $\{1,3\}$ $\{1,4\}$ $\{1,5\}$ $\{2,3\}$ $\{2,4\}$ $\{2,5\}$ $\{3,4\}$ $\{3,5\}$ $\{4,5\}$

$\{1\}$ $\{2\}$ $\{3\}$ $\{4\}$ $\{5\}$

$\emptyset$

Pointed shell

Which refinement when $\mathbb{C}^A_c(f)$ fails?

Idea: add the most abstract over-approximation of c that yields local completeness

(the most concrete would be c itself)

$$\max(\{x \in C \mid x \le A(c),\ \mathbb{C}^{A_x}_c(f)\}) = \{u\}$$

In the case of $\mathbb{C}^A_P(\mathsf{b}?)$ we set:

$$u \triangleq (A(P \cap \mathrm{b}) \cap \mathrm{b}) \cup (A(P \cap \neg\mathrm{b}) \cap \neg\mathrm{b})$$

A (forward) repair strategy

Given A, P, c try to find Q such that $\vdash_A [P]$ r $[Q]$

If a local completeness proof obligation fails, refine A with u_1 and retry

If a local completeness proof obligation fails, refine A_{u_1} with u_2 and retry

If a local completeness proof obligation fails, refine $A_{\{u_1,u_2\}}$ with u_3 and retry

…

Until $\vdash_{A_N} [P]$ r $[Q]$ for some $N = \{u_1, \ldots, u_n\}$ and Q

A (forward) repair strategy

Function fRepair$_A$$(N, P, \mathsf{r})$

  found := **false**;

  do

    out := find$_A$$(N, P, \mathsf{r})$;

    switch out **do**

      case Q **do** found := **true**;   `// underapprox.`

      case $\langle R, \mathsf{e}\rangle$ **do** N := refine$_A$$(N, R, \mathsf{e})$; `// incompl.`

  while $(\neg$found$)$;

  return $\langle N, \text{out}\rangle$;

given A, N, P, r try to find Q such that $\vdash_{A_N} [P]\ \mathsf{r}\ [Q]$

$\vdash_{A_N} [P]\ \mathsf{r}\ [Q]$

$\neg\mathbb{C}^{A_N}_{R}(\mathsf{e})$

update N and retry

the latest set of refinements N and Q are returned

Slogan

AIR is to program verification
what CEGAR is for model checking

(we have shown that CEGAR is an instance of AIR)

Concluding Remarks

What else?

AIR with backward repair or
how to find the most abstract domain refinement for proving correctness

Difficulties in swapping the roles of over- and under-approximations

LCL enhancements
(local variables, rewrite strategy languages)

What next?

Expressiveness hierarchy of (locally) complete domains?

Handling pointers and memory errors with ideas from separation logic

Theoretical foundations for scalable bug-catching and security tools Meta

References

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Isabel Garcia-Contreras, Dusko Pavlovic:
Abstract extensionality: on the properties of incomplete abstract interpretations.
Proc. ACM Program. Lang. 4(POPL): 28:1-28:28 (2020)

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Francesco Ranzato:
A Logic for Locally Complete Abstract Interpretations. LICS 2021: 1-13

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Francesco Ranzato:
A Correctness and Incorrectness Program Logic. J. ACM 70(2): 15:1-15:45 (2023)

Flavio Ascari, Roberto Bruni, Roberta Gori:
Limits and difficulties in the design of under-approximation abstract domains.
FoSSaCS 2022: 21-39

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Francesco Ranzato:
Abstract interpretation repair. PLDI 2022: 426-441

Roberto Bruni, Roberta Gori, Nicolas Manini:
Deciding Program Properties via Complete Abstractions on Bounded Domains.
SAS 2022: 175-200

Flavio Ascari, Roberto Bruni, Roberta Gori:
Logics for Extensional, Locally Complete Analysis via Domain Refinements.
ESOP 2023: 1-27

Thanks for the kind invitation
and for the attention!