Program analysis: from proving correctness to proving incorrectness

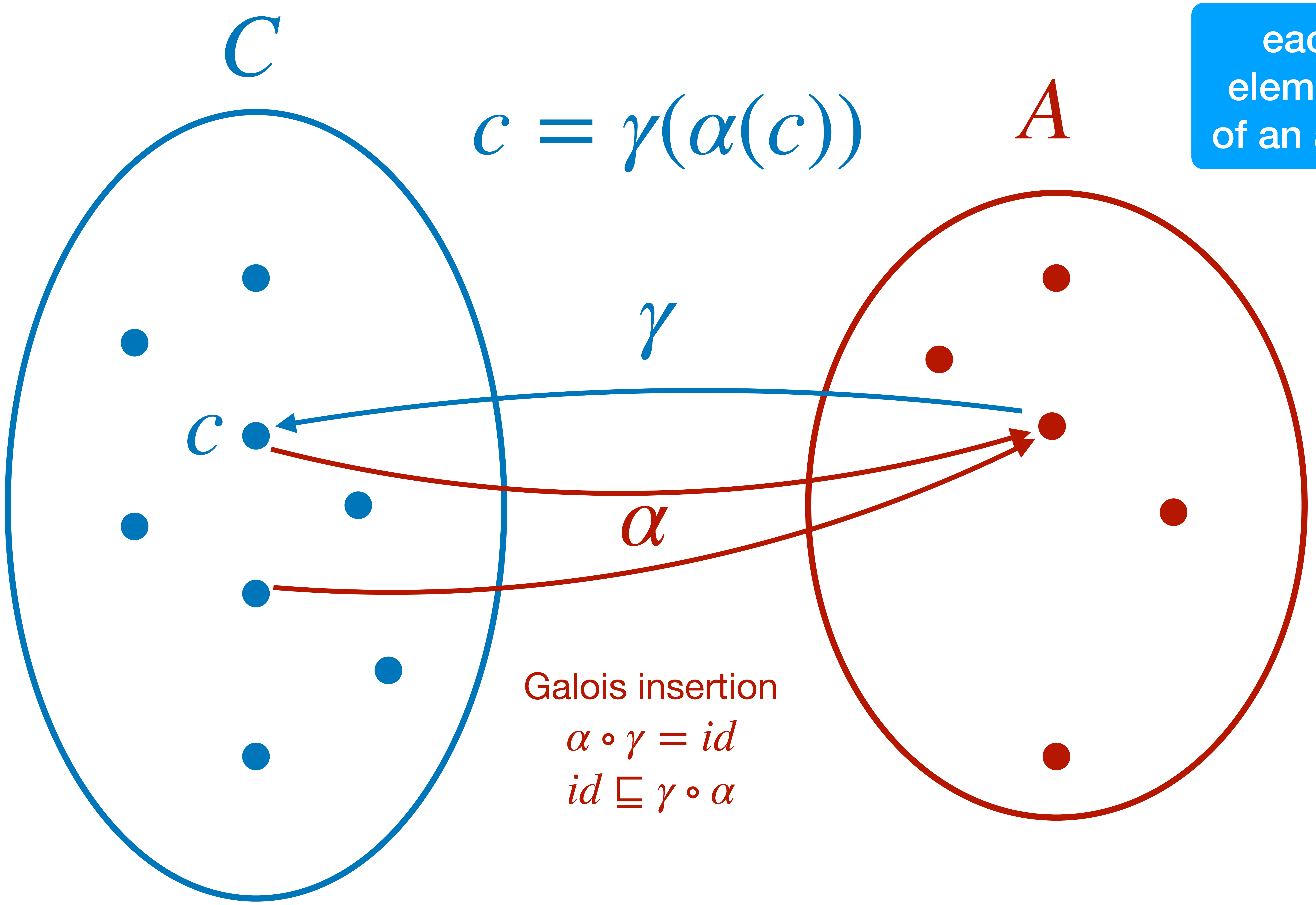
Roberto Bruni, Roberta Gori
(University of Pisa)

Lecture #06

BISS 2024
March 11-15, 2024

Addendum:
Abstract Interpretation as closure

Expressible elements



each expressible element is the image of an abstract element

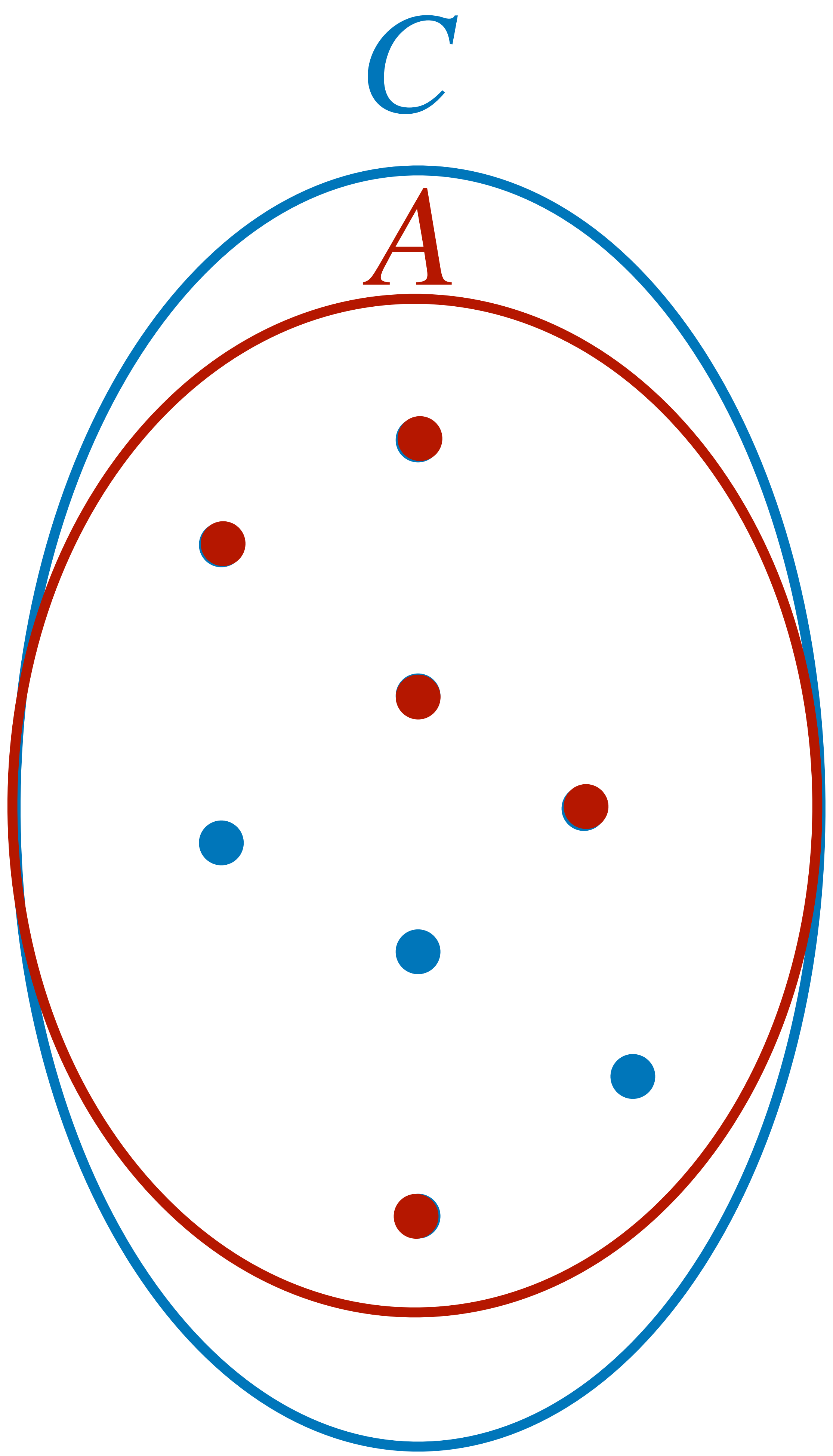
$$c = \gamma(\alpha(c))$$
$$\Rightarrow$$
$$\exists a \in A \,.\, c = \gamma(a)$$

$$\forall a \,.\, \alpha(\gamma(a)) = a$$
$$\Rightarrow$$
$$\forall a \,.\, \gamma(\alpha(\gamma(a))) = \gamma(a)$$
$$\Rightarrow$$
$\forall a \,.\, \gamma(a)$ is expressible

the image of an abstract element is expressible

$\gamma(A)$ is the set of expressible elements

Galois insertion as closures



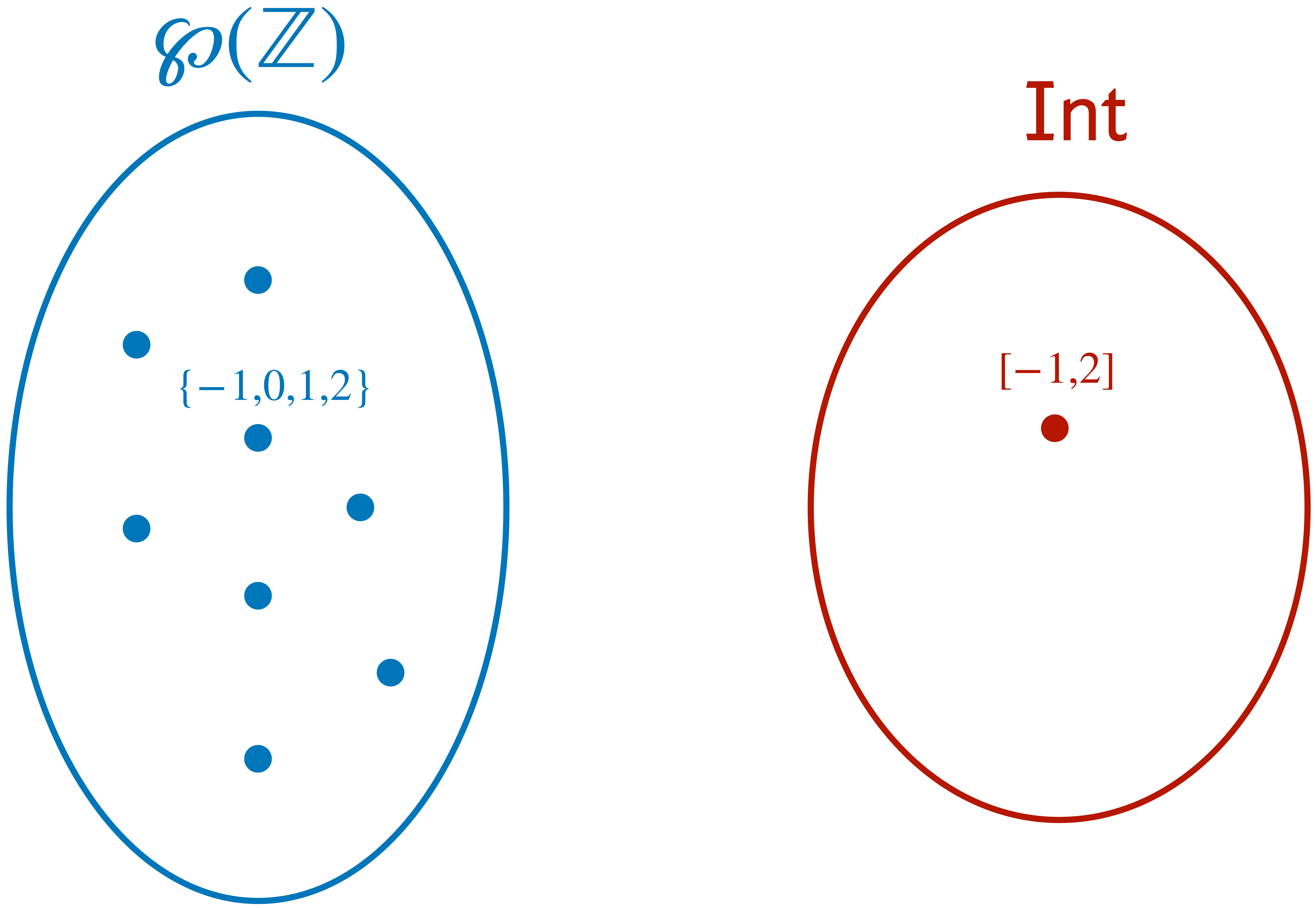
The abstract domain
can just be seen as
a subset of the concrete domain

We write $A(c)$
as a shorthand for $\gamma(\alpha(c))$

$A(C)$ is the set of
expressible elements

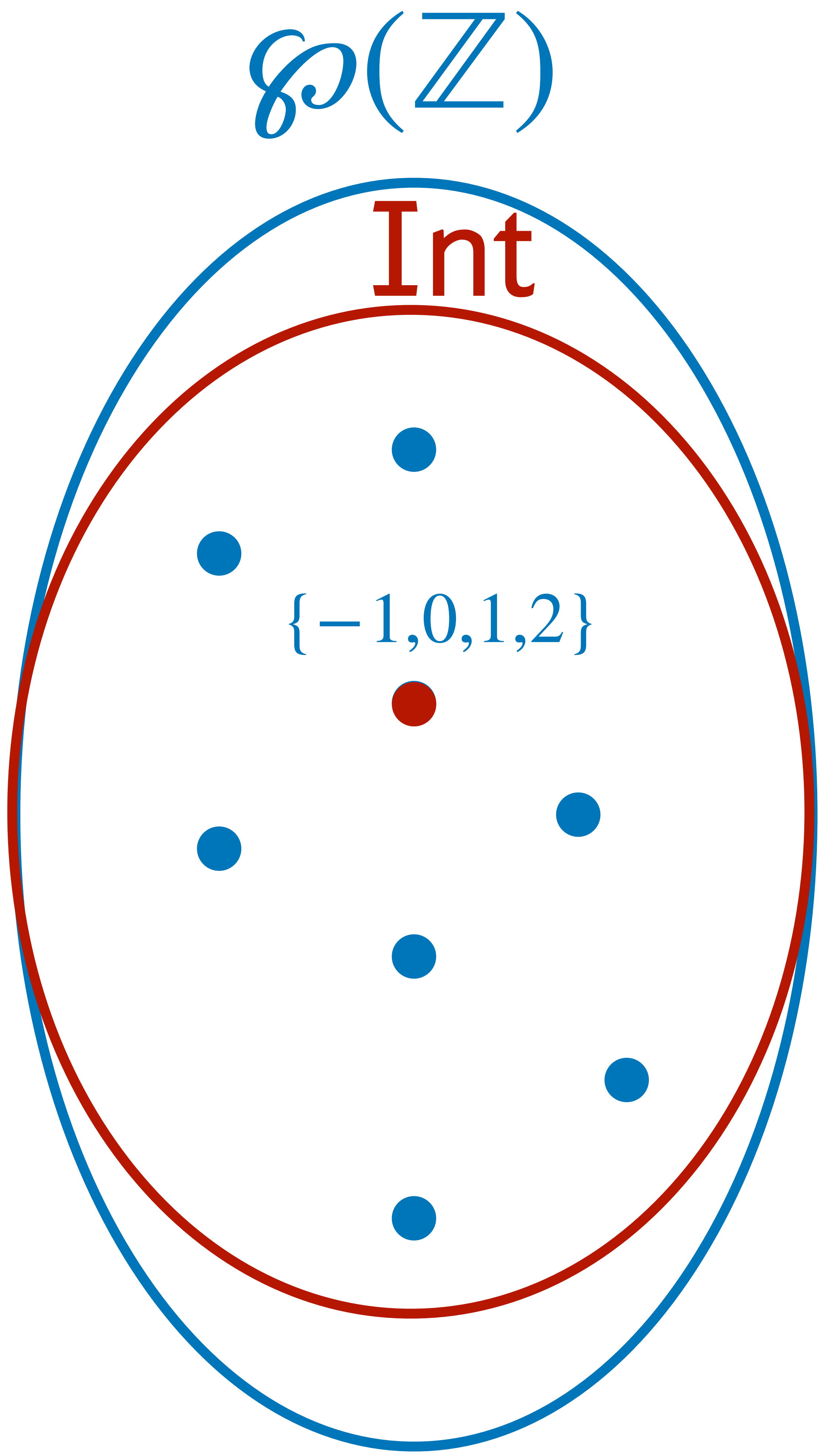
Since $A(A(c)) = A(c)$ the map $A : C \to C$ is a closure operator

Example



No need of symbolic representations

Example

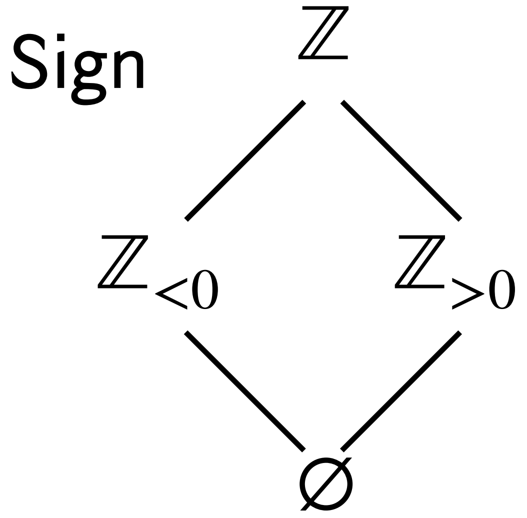


No need of symbolic representations

Examples

$\text{Int}(\{2,4,6,...\}) = \{2,3,4,5,6...\} = [2,\infty]$

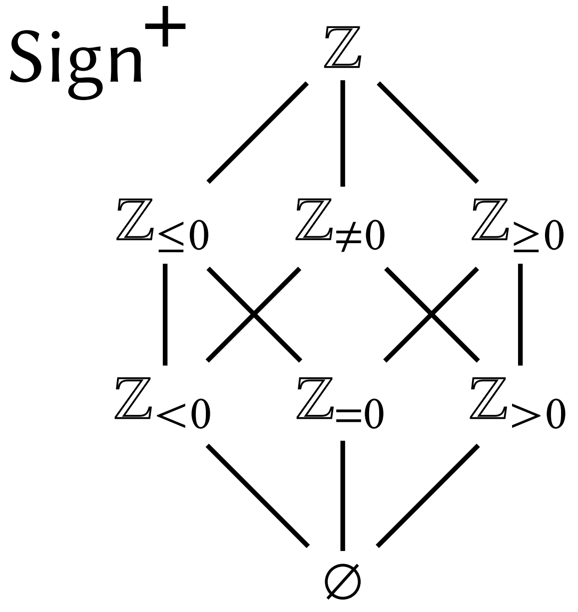
$\text{Sign}(\{2,4,6,...\}) = \{1,2,3,4,5,6...\} = \mathbb{Z}_{>0}$



$\text{Int}(\{0,2,4,6,...\}) = \{0,1,2,3,4,5,6...\} = [0,\infty]$

$\text{Sign}(\{0,2,4,6,...\}) = \{\ldots, -1,0,1,2,3,4,5,6...\} = \mathbb{Z}$

$\text{Sign}^+(\{0,2,4,6,...\}) = \{0,1,2,3,4,5,6...\} = \mathbb{Z}_{\geq 0}$



Completeness, revisited

$$\forall P \,.\, A(\llbracket c \rrbracket P) = \llbracket c \rrbracket^{\#}_{A} A(P)$$

$$\begin{array}{ccc} \subseteq & & \subseteq \end{array}$$

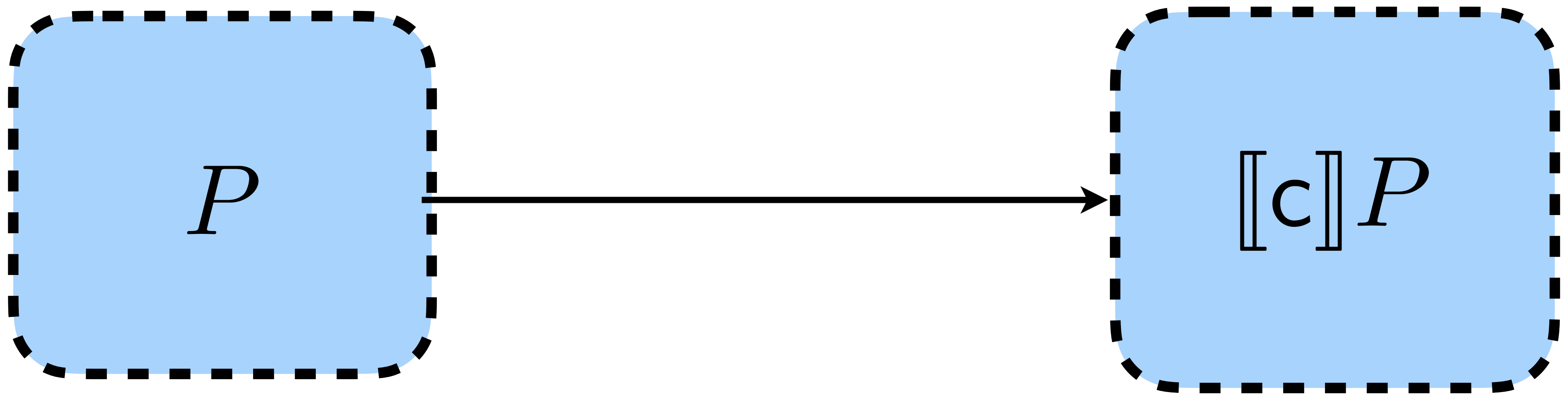
$$A(\llbracket c \rrbracket A(P))$$

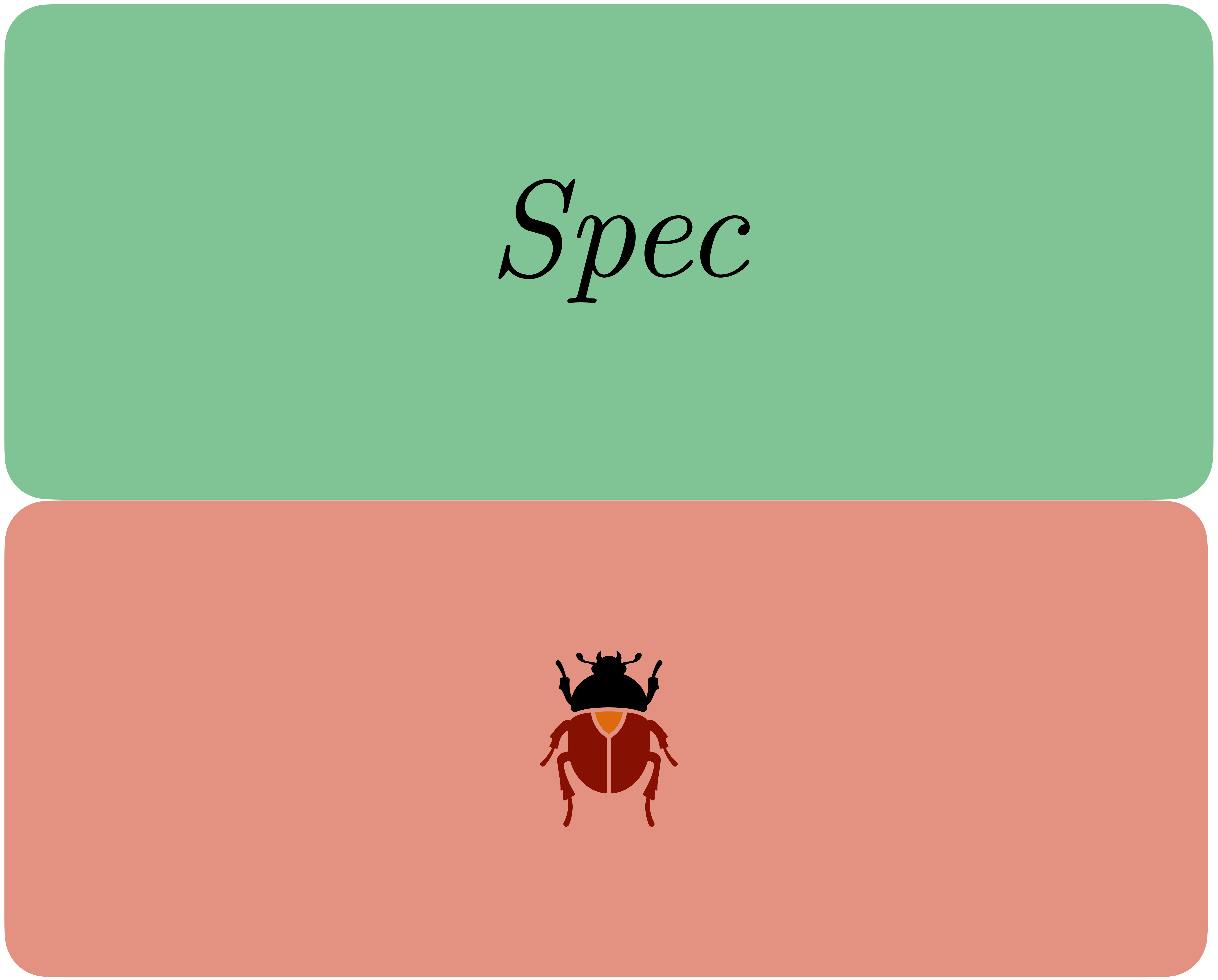
Completeness equation: $\forall P \,.\, A(\llbracket c \rrbracket P) = A(\llbracket c \rrbracket A(P))$

Recap:
to be correct or incorrect?

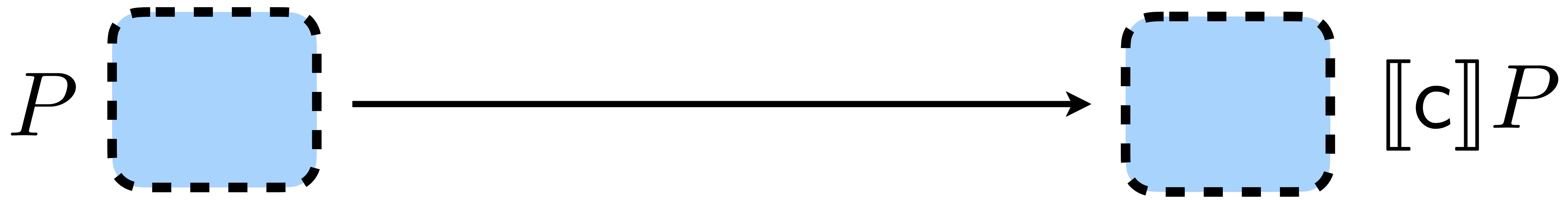
Verification problem

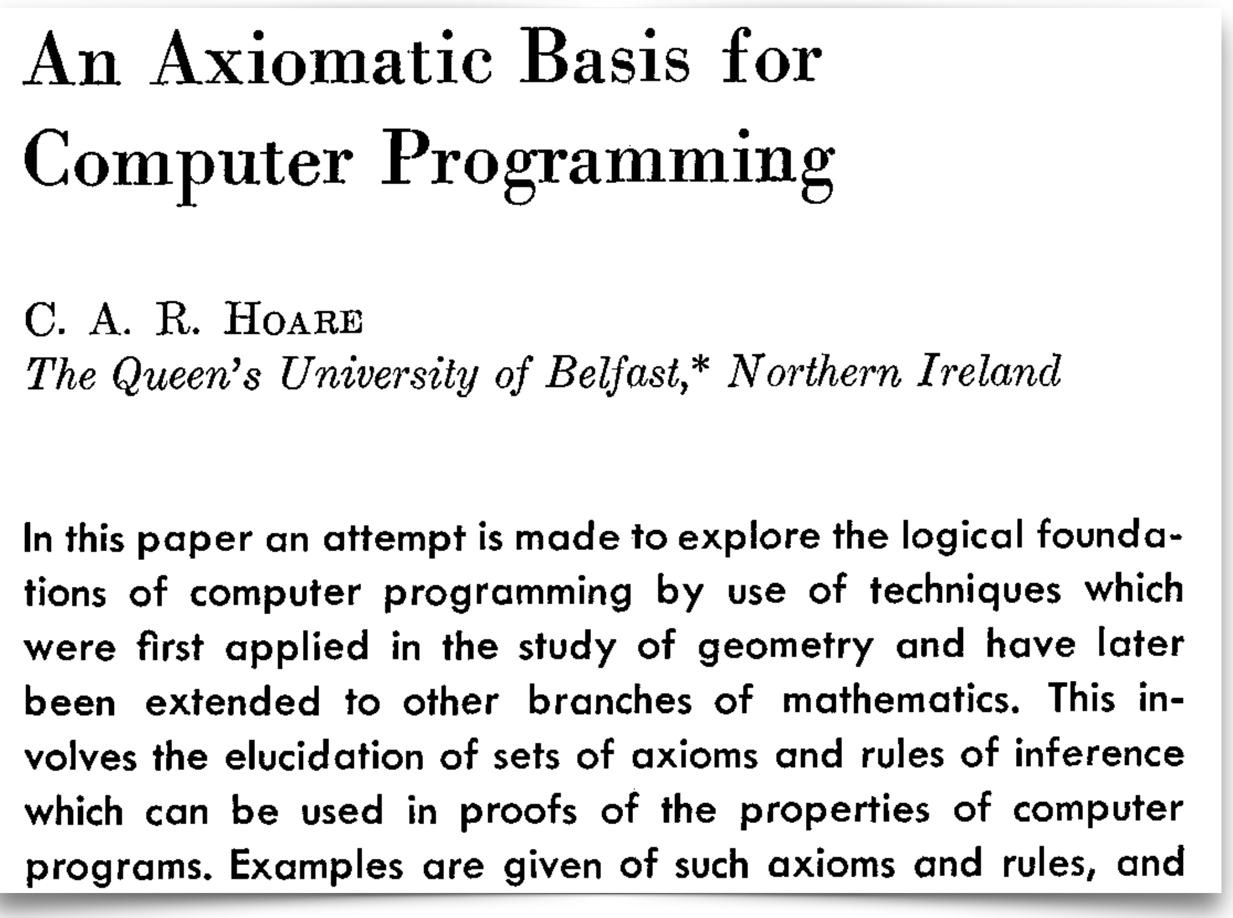
$$[\![\mathsf{c}]\!] P \overset{?}{\subseteq} \mathit{Spec}$$





Over vs Under



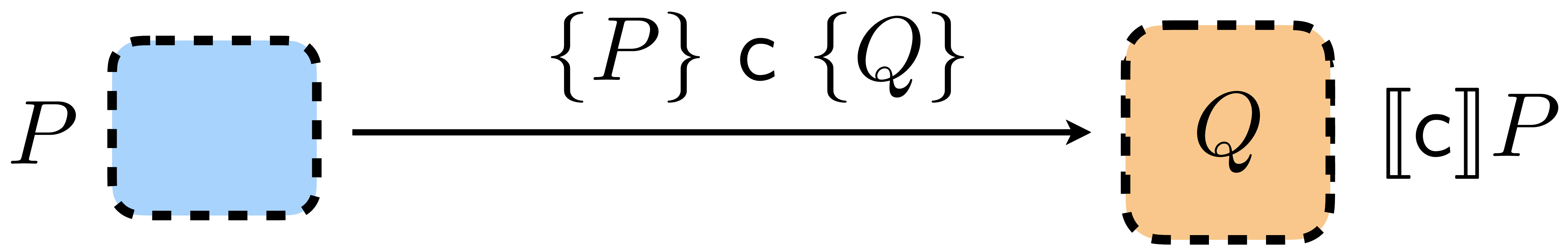


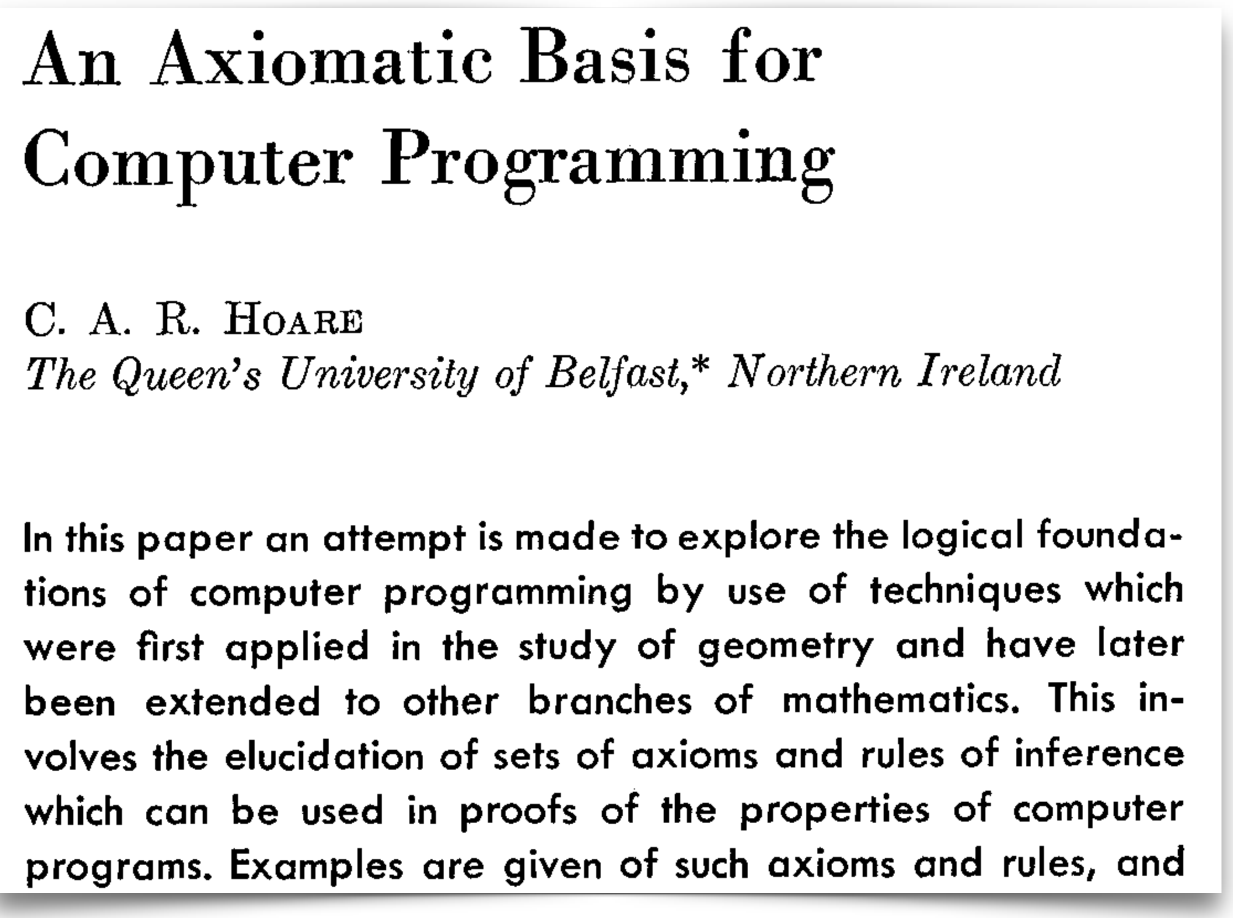
An Axiomatic Basis for Computer Programming

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

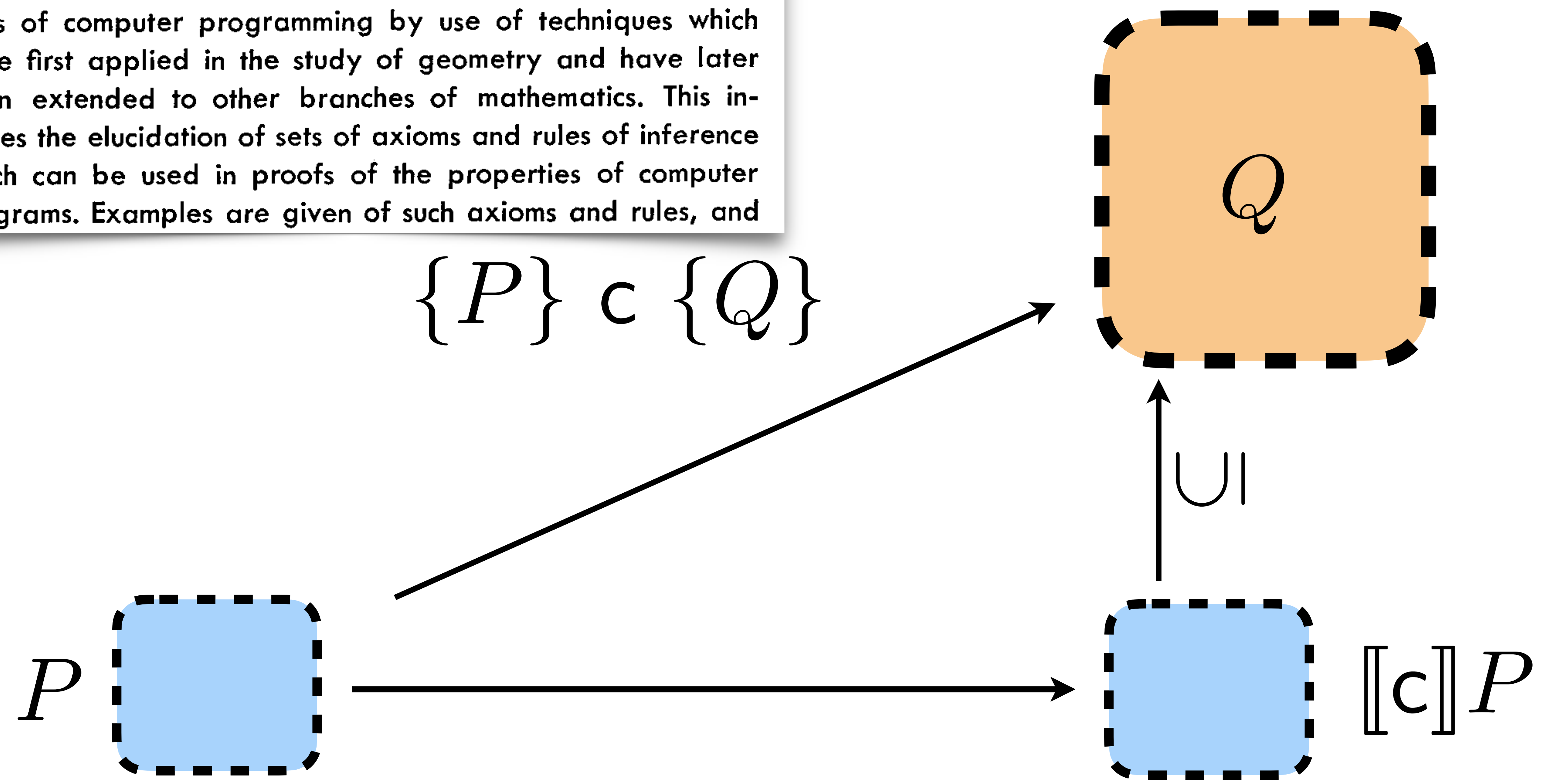


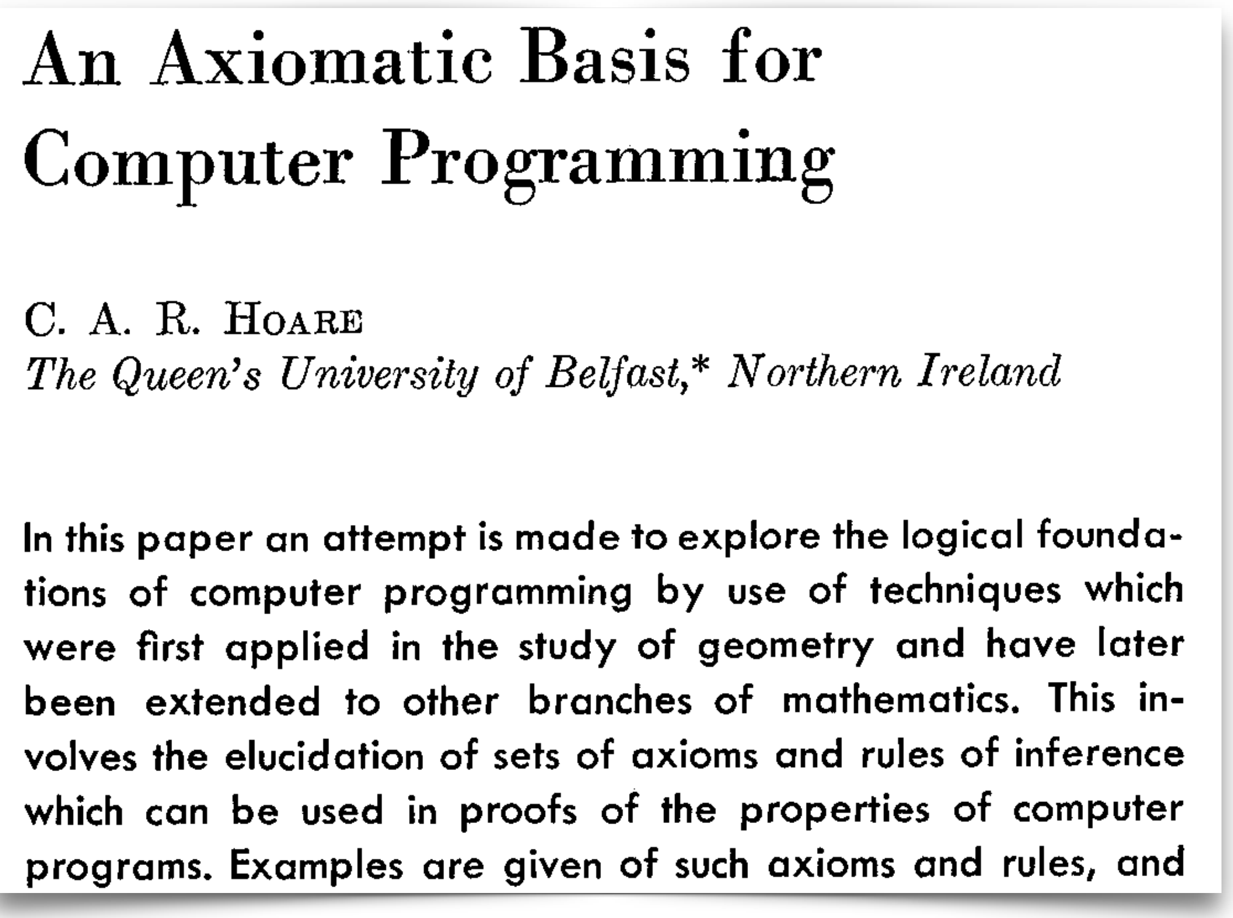

An Axiomatic Basis for
Computer Programming

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



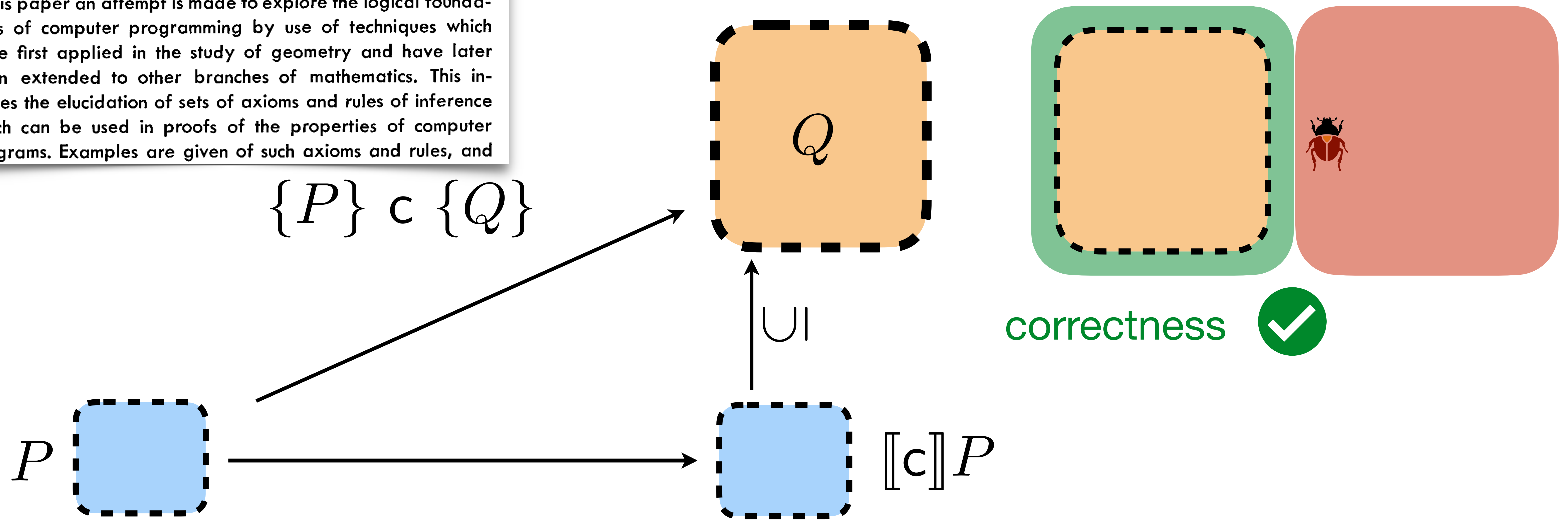


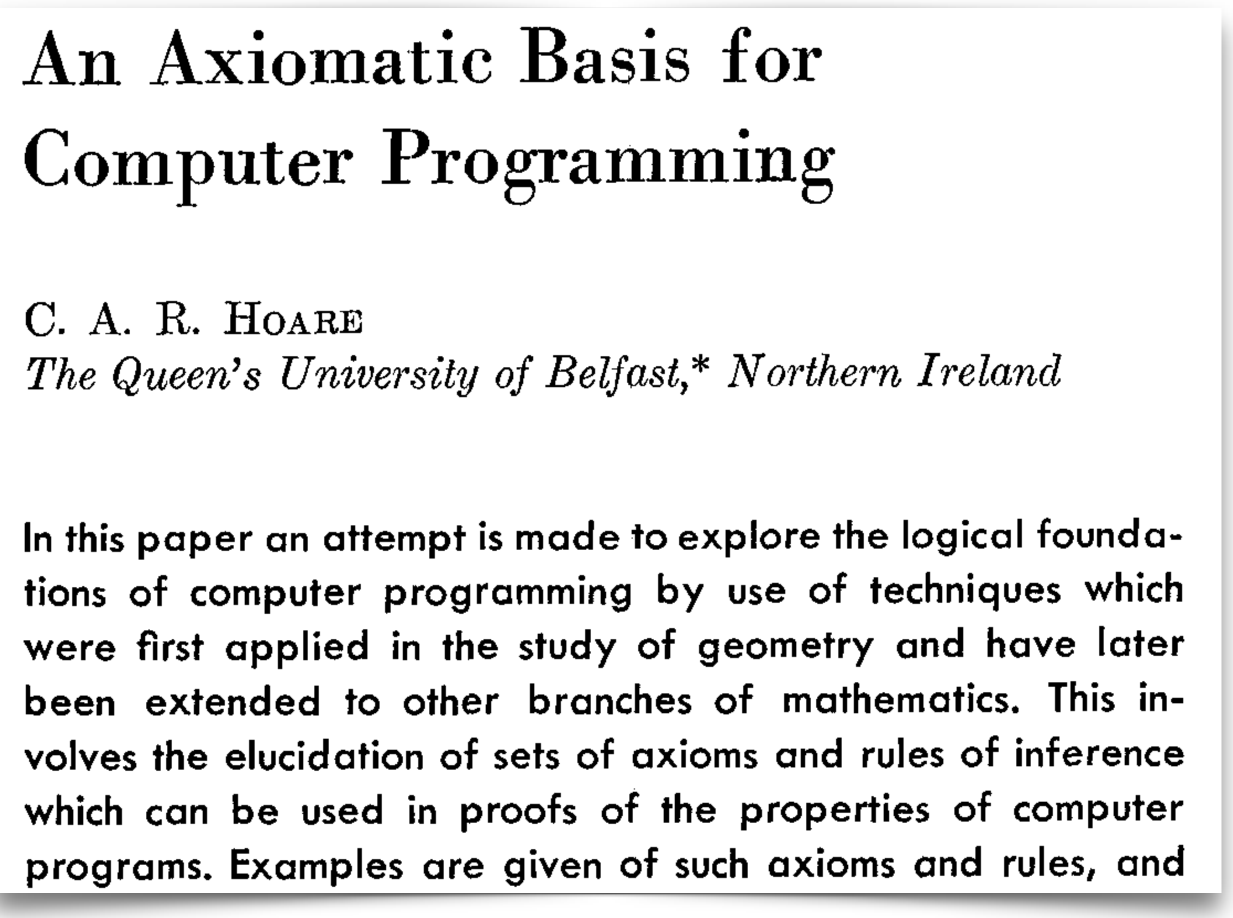
An Axiomatic Basis for Computer Programming

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

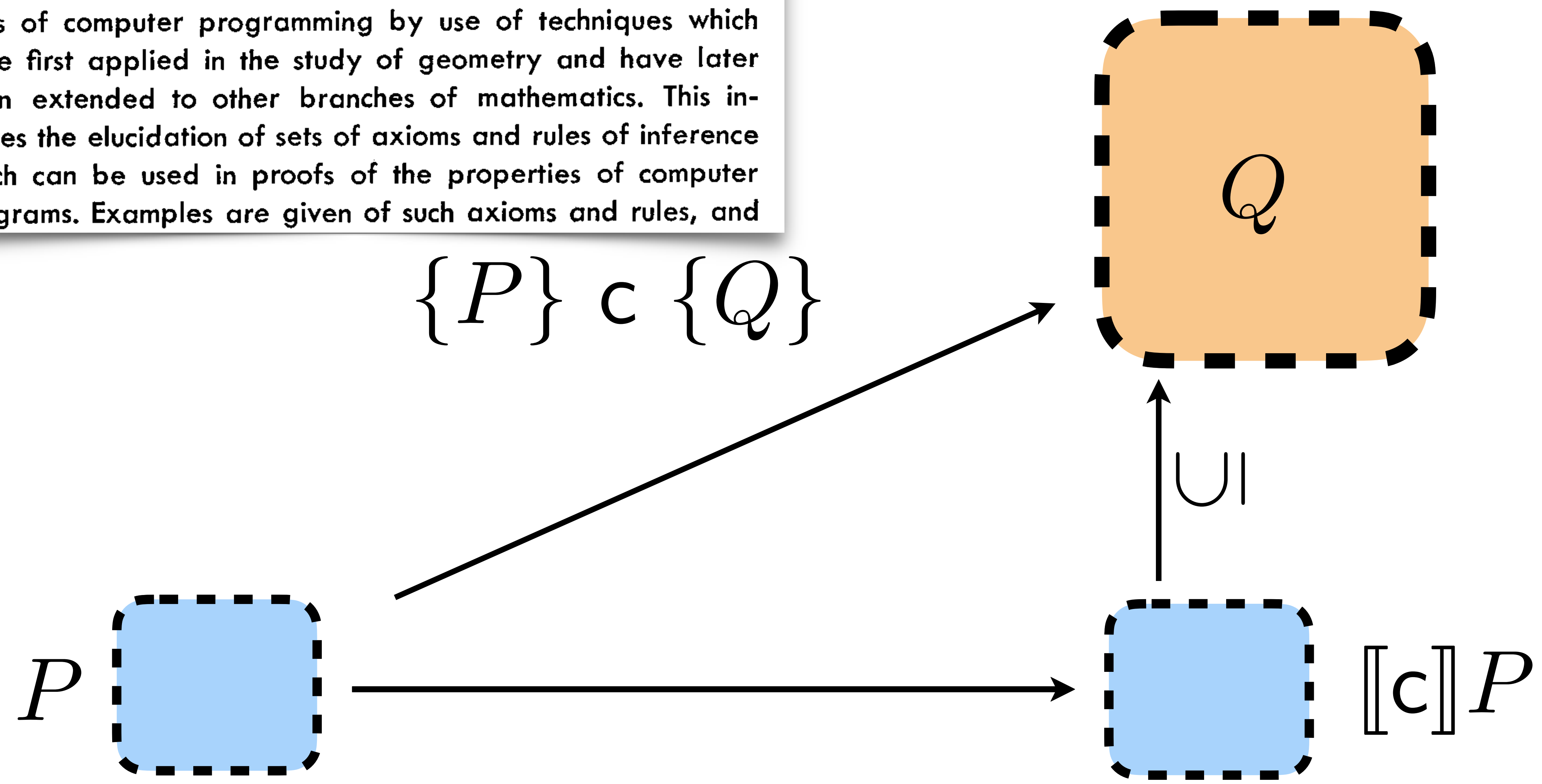



An Axiomatic Basis for Computer Programming

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

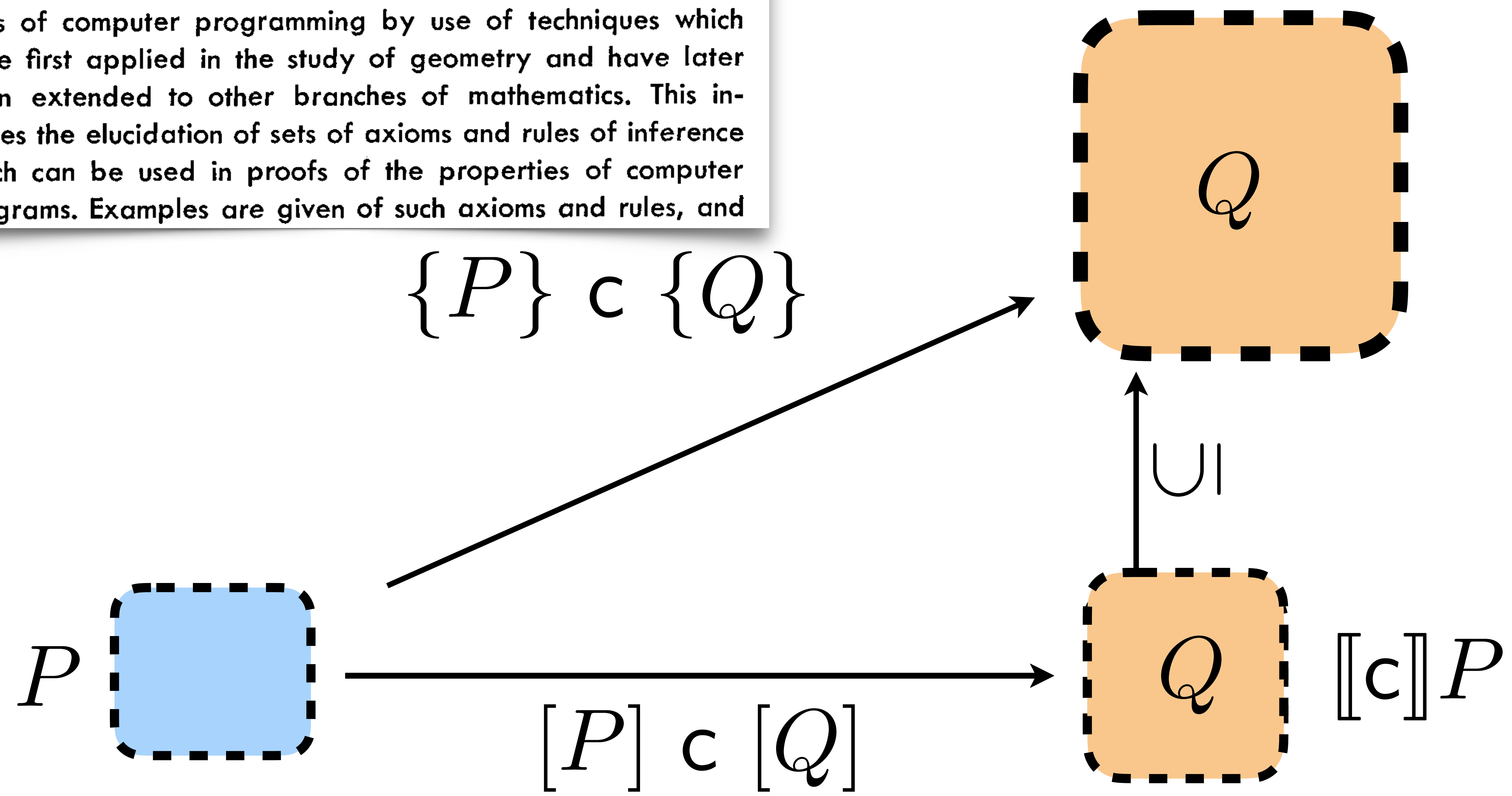


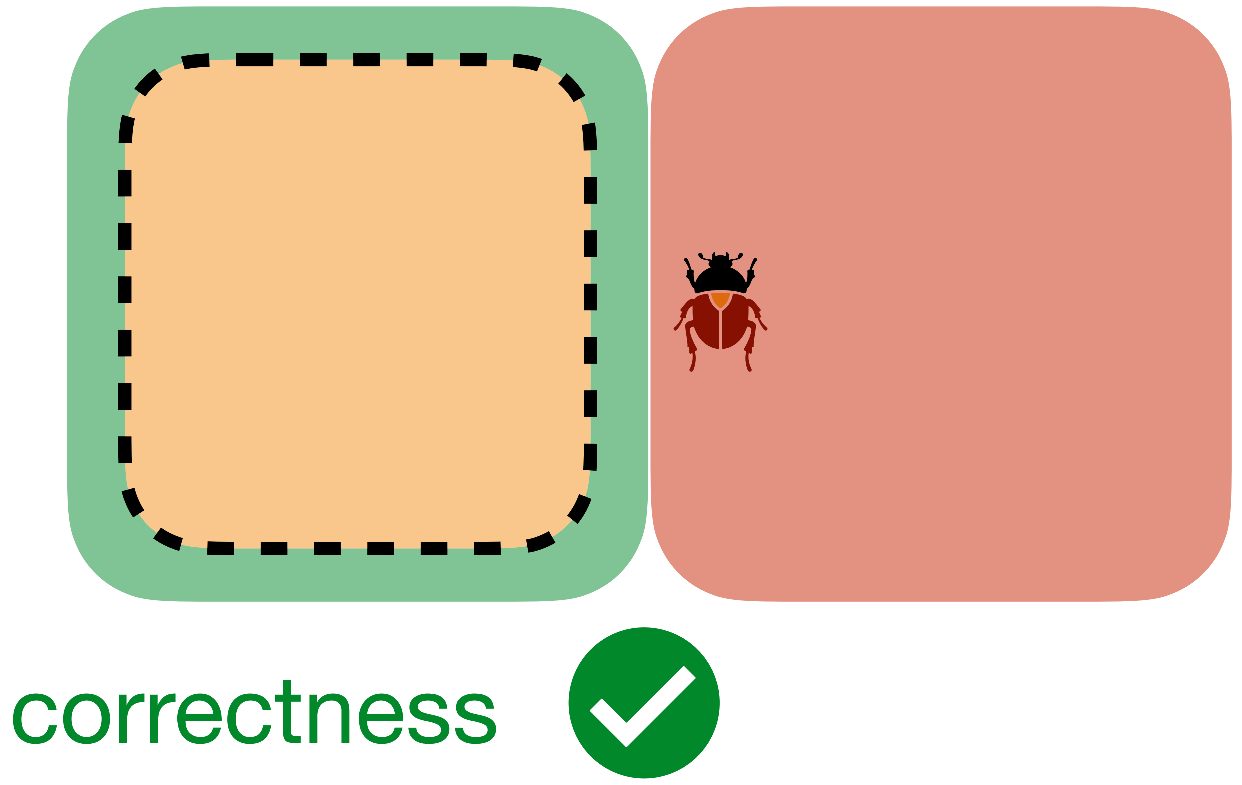
An Axiomatic Basis for Computer Programming

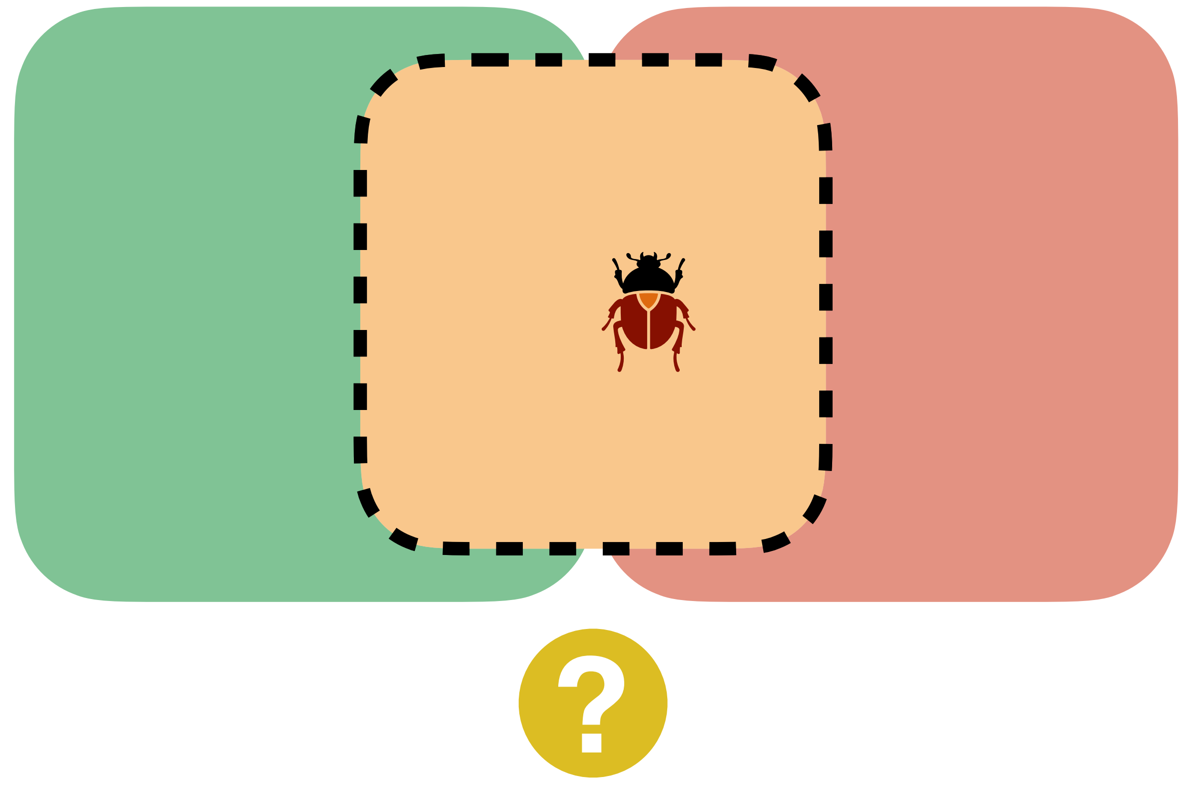
C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

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







Incorrectness Logic

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

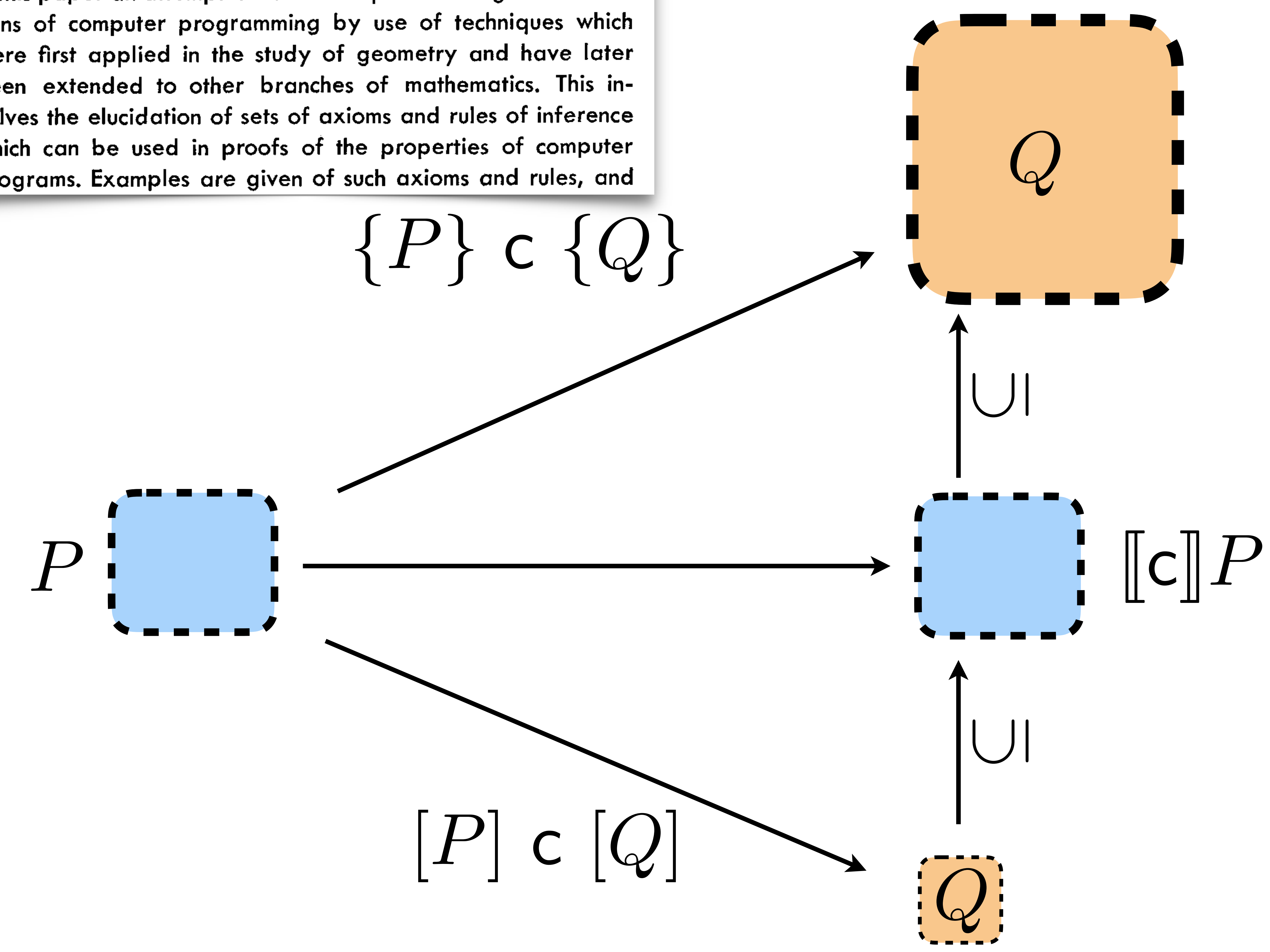
Program correctness and incorrectness are two sides of the same coin. As a programmer, even if you would like to have correctness, you might find yourself spending most of your time reasoning about incorrectness. This includes informal reasoning that people do while looking at or thinking about their 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

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

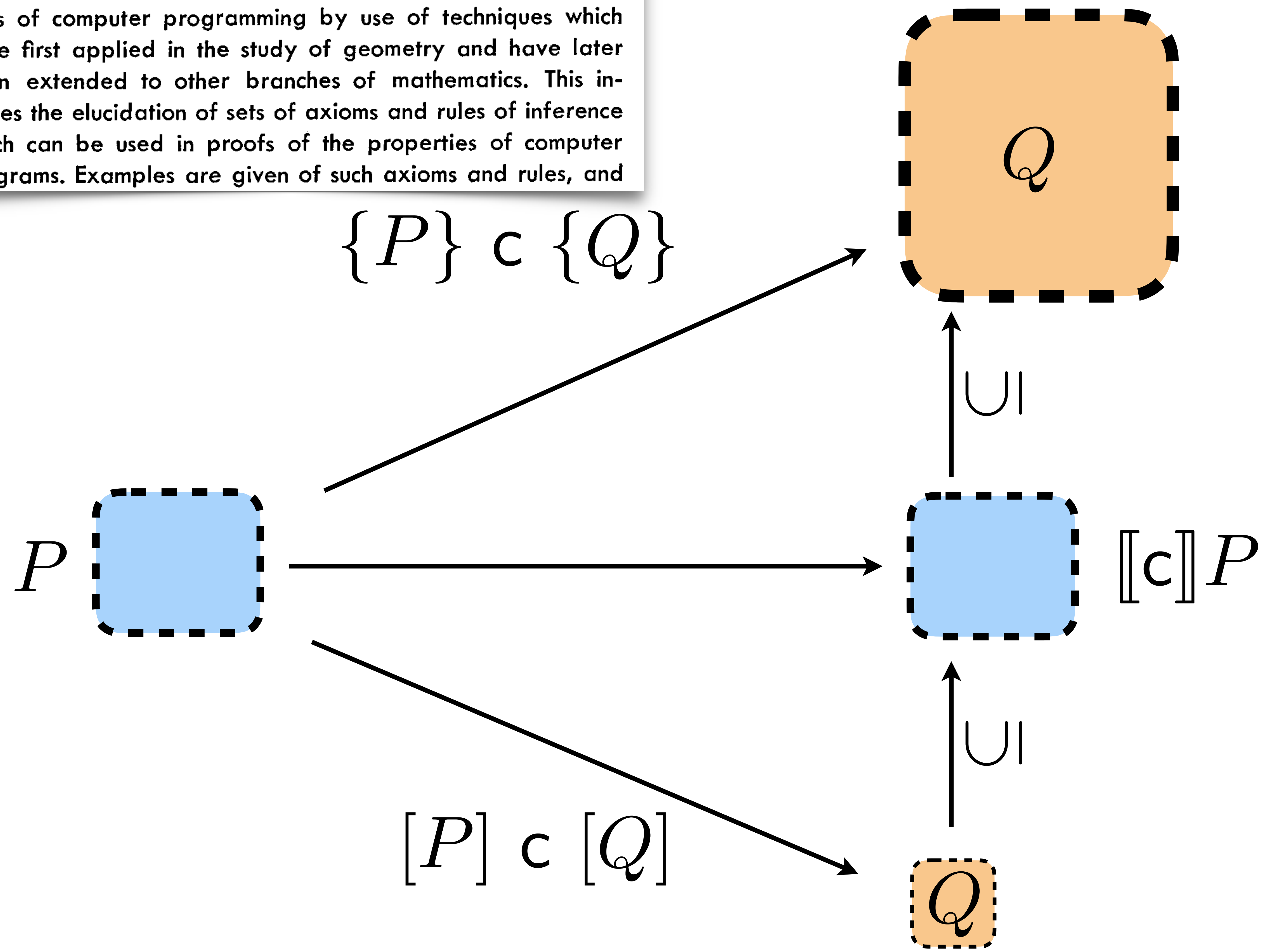
Program correctness and incorrectness are two sides of the same coin. As a programmer, even if you would like to have correctness, you might find yourself spending most of your time reasoning about incorrectness. This includes informal reasoning that people do while looking at or thinking about their 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.

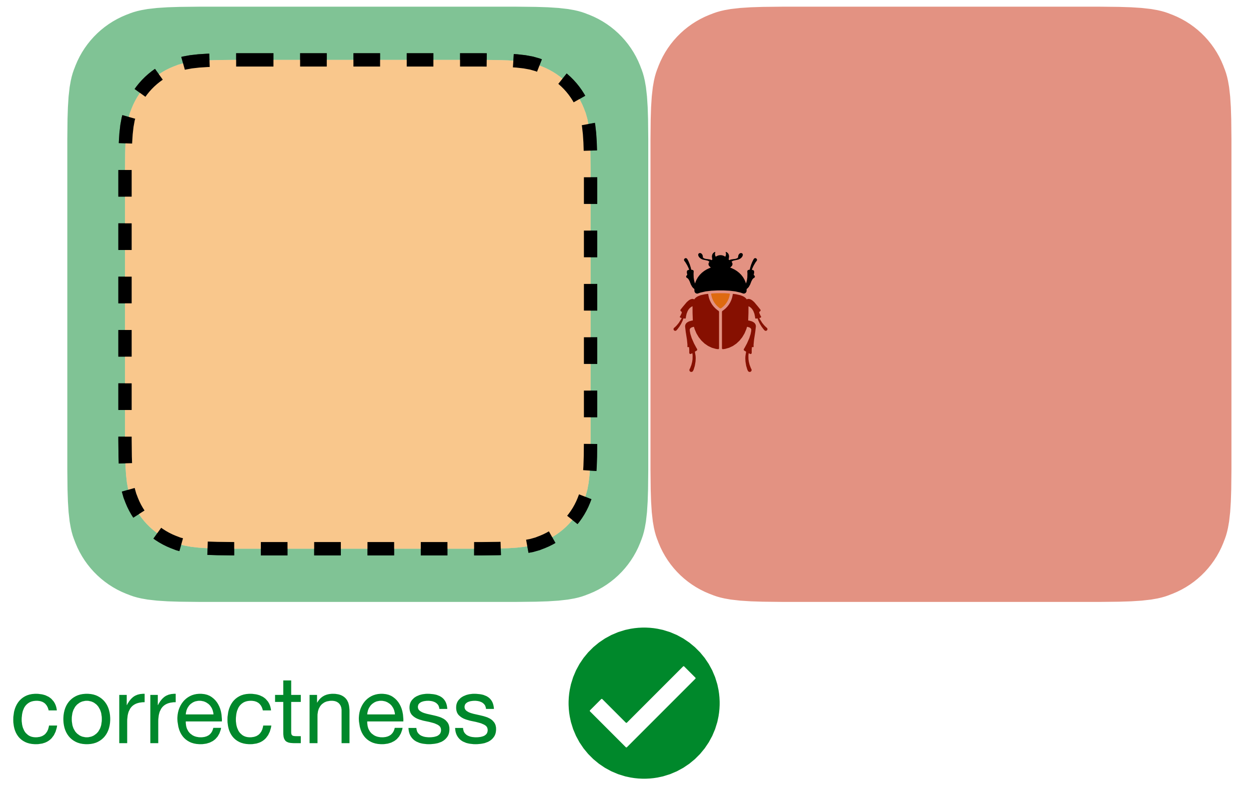
An Axiomatic Basis for
Computer Programming

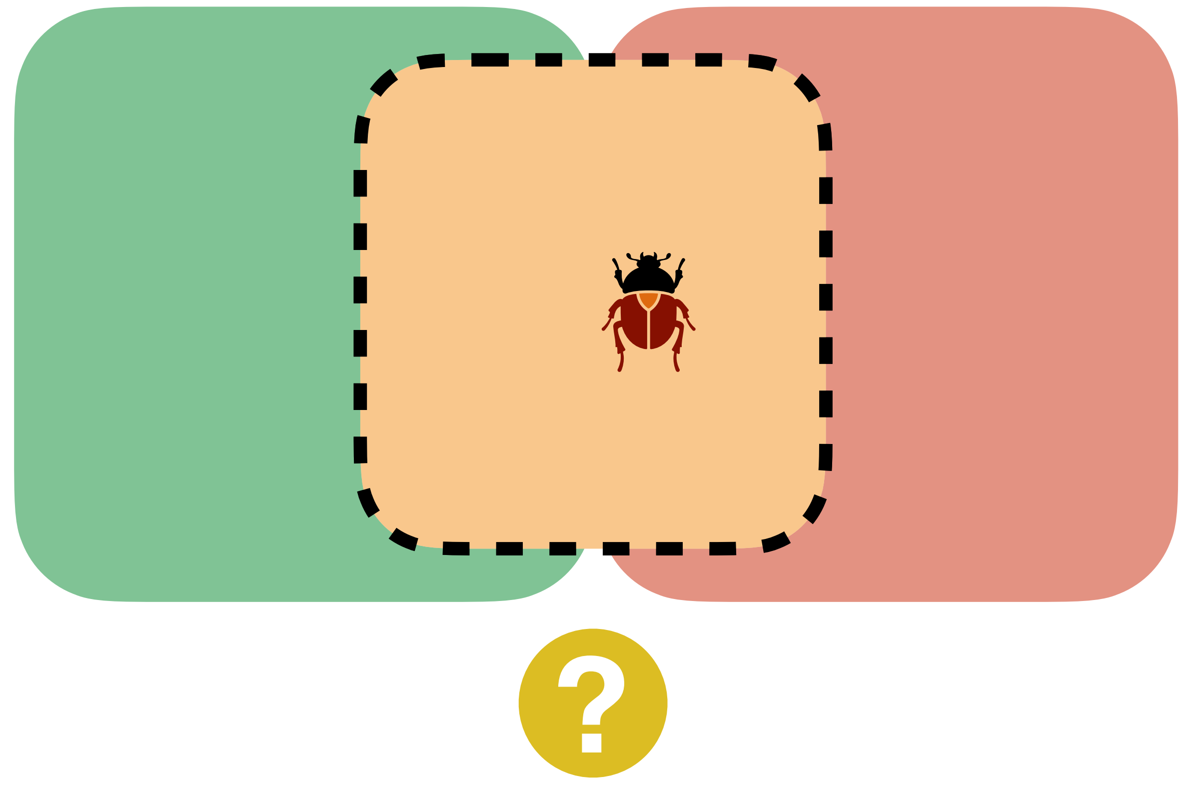
C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

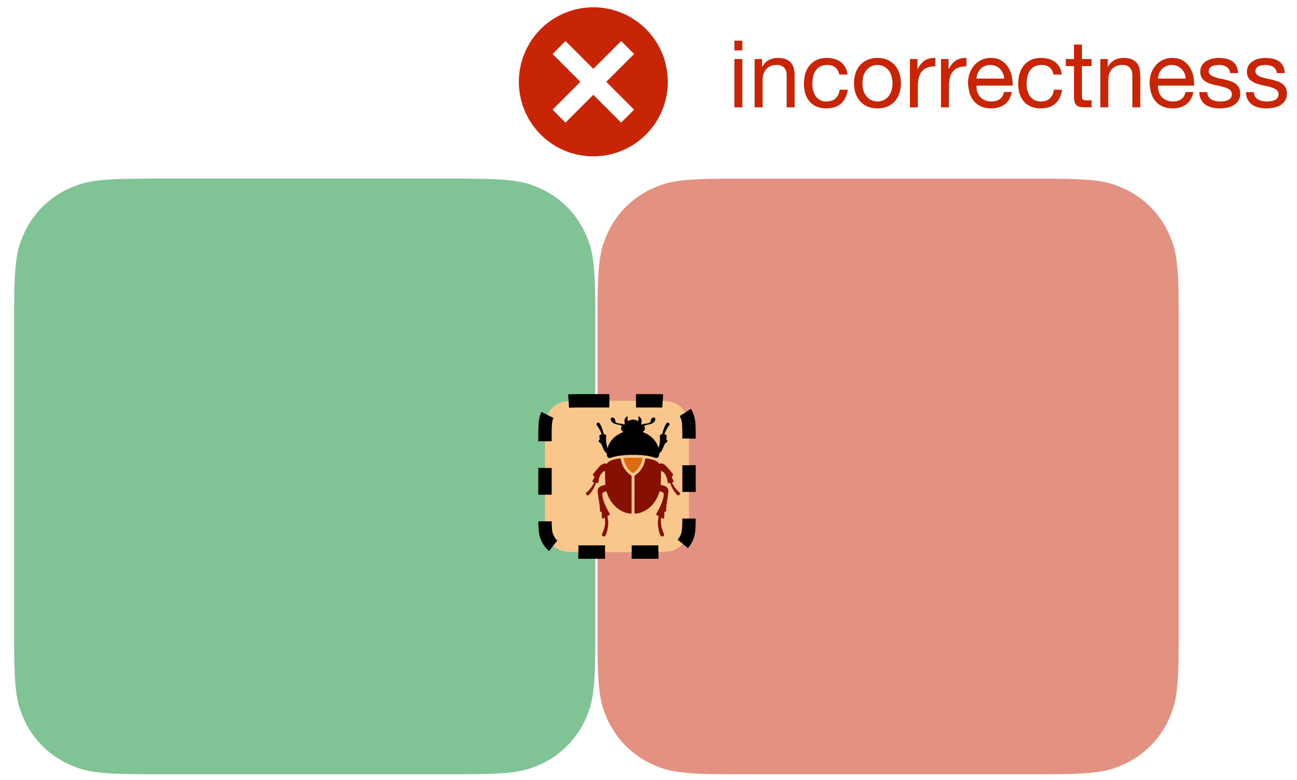
In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which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

Over vs Under









Incorrectness Logic

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

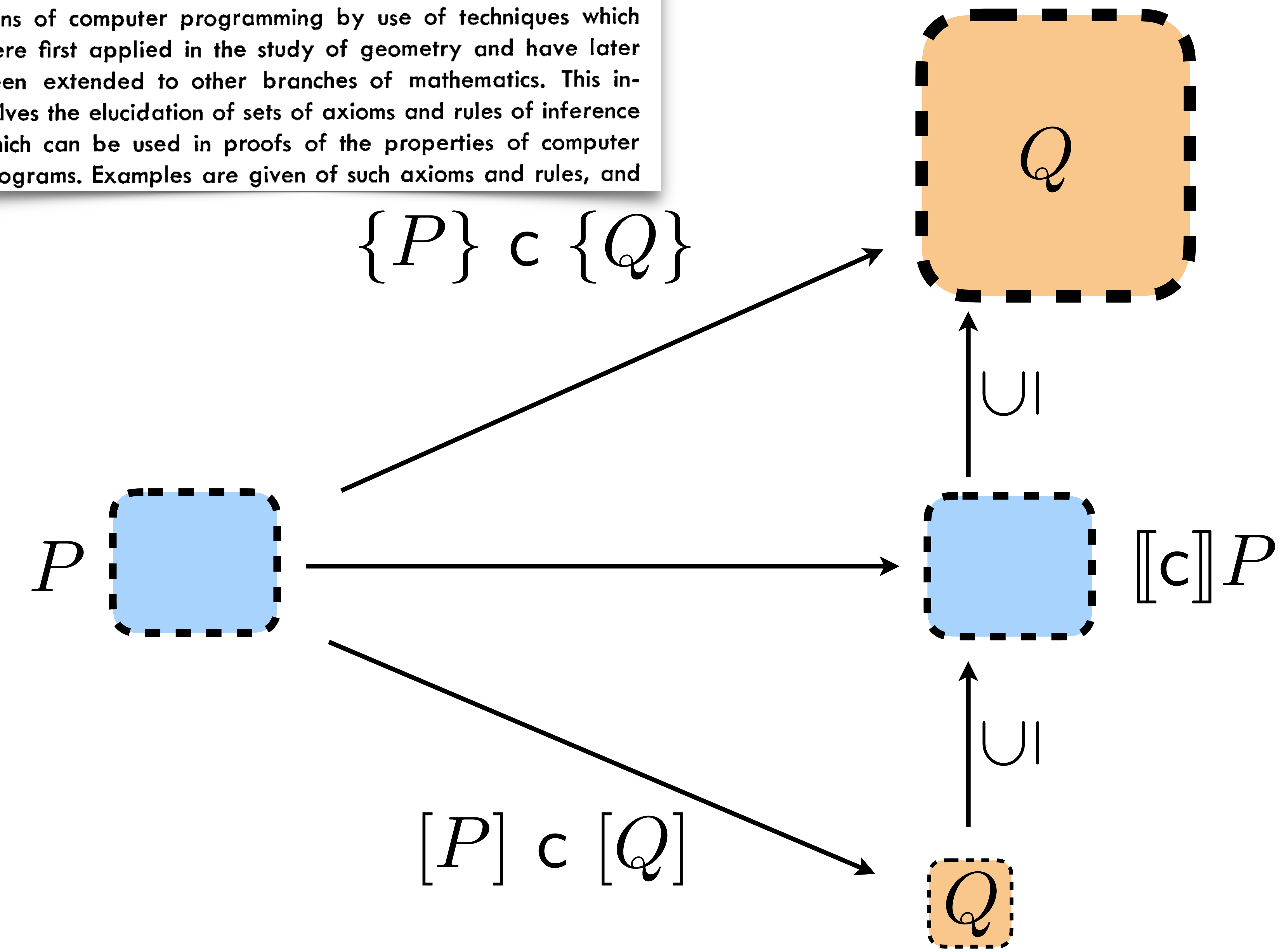
Program correctness and incorrectness are two sides of the same coin. As a programmer, even if you would like to have correctness, you might find yourself spending most of your time reasoning about incorrectness. This includes informal reasoning that people do while looking at or thinking about their 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.

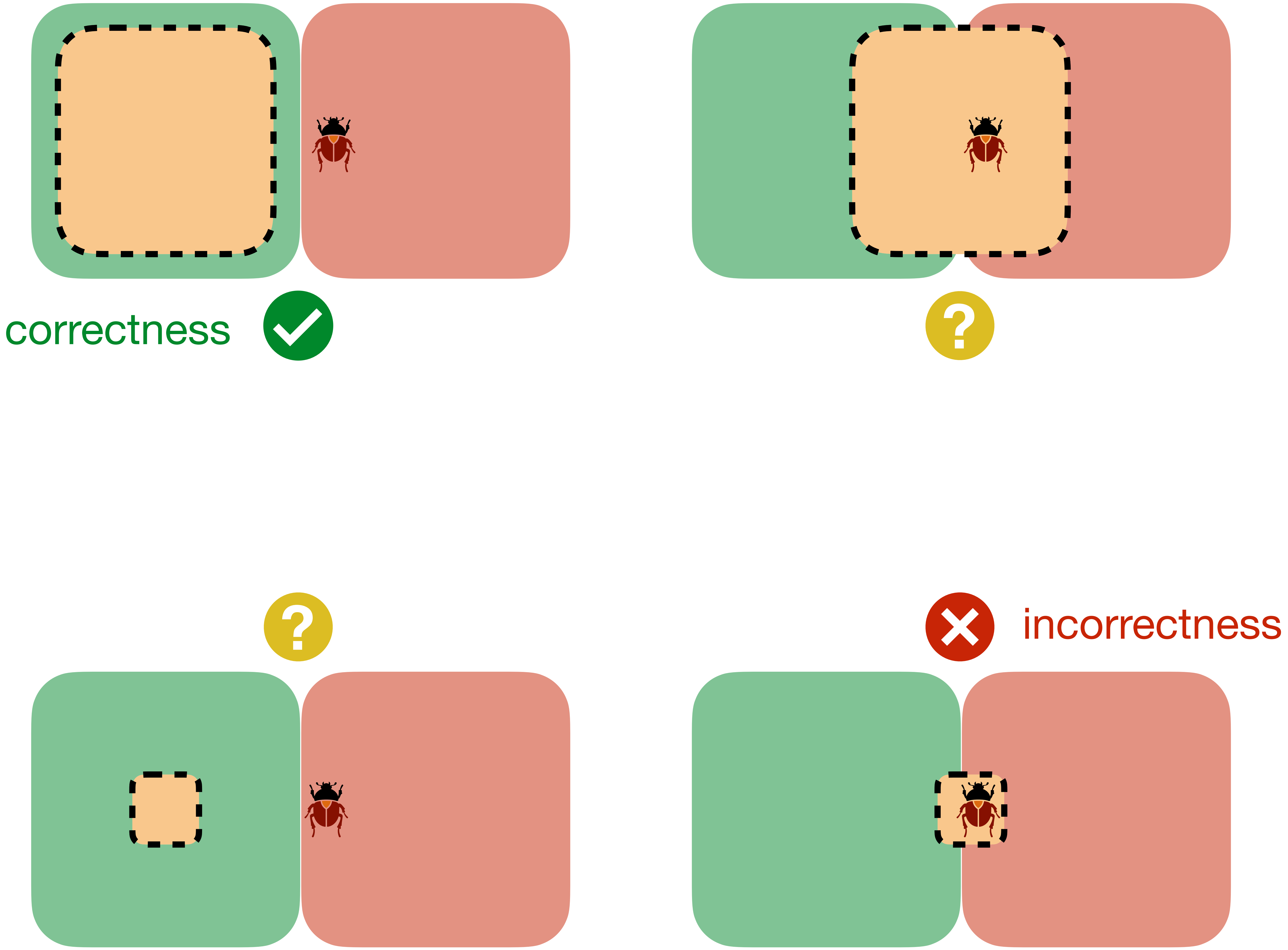
Over vs Under

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





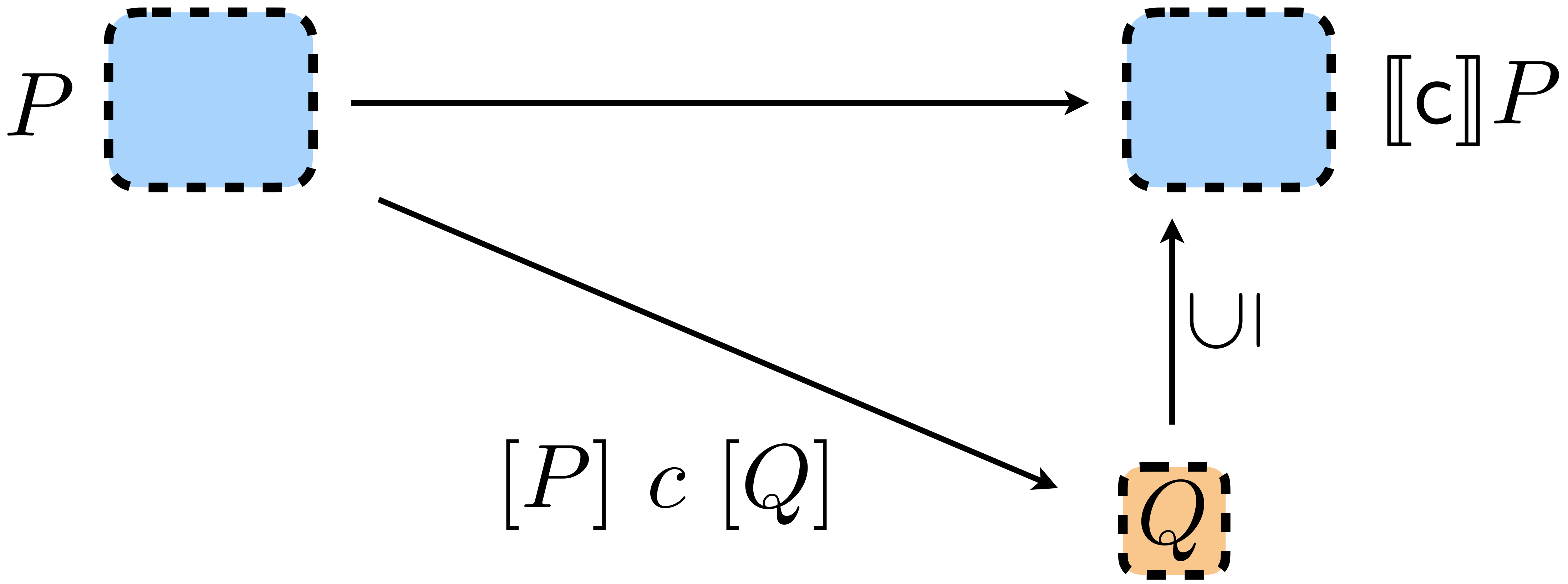
Incorrectness Logic

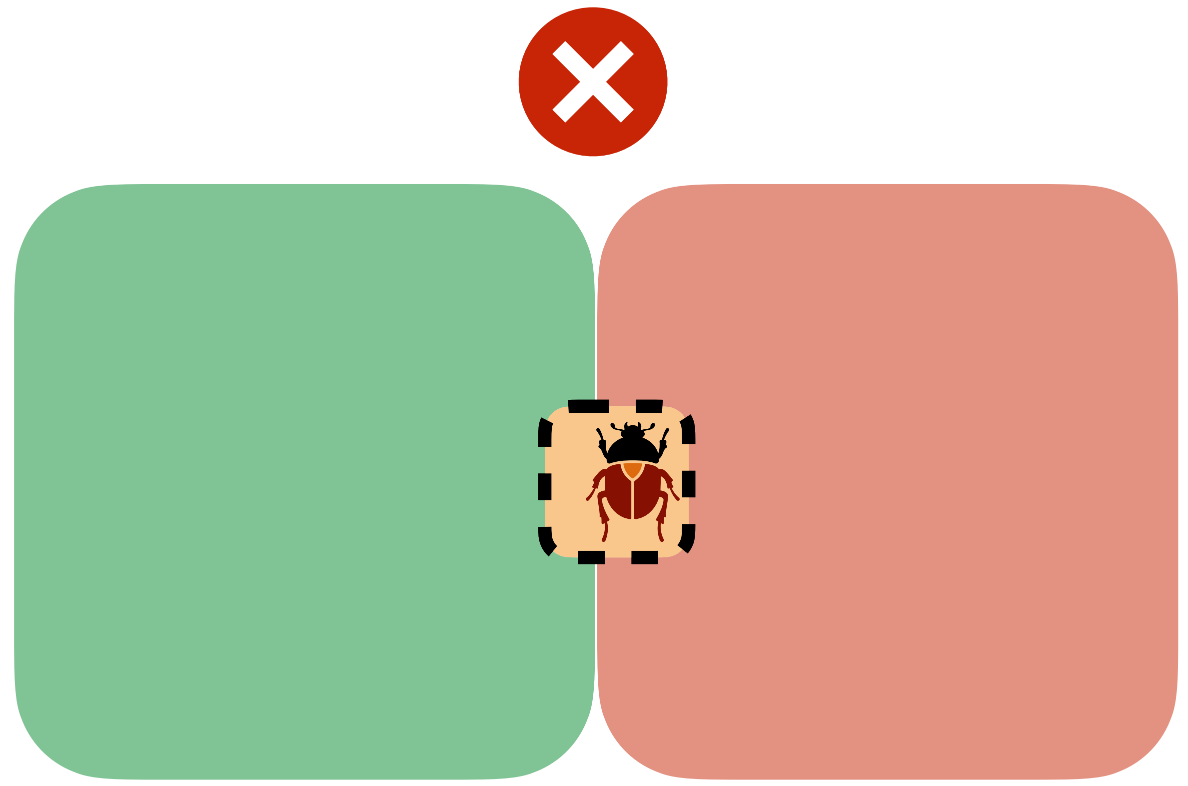
PETER W. O'HEARN, Facebook and University College London, UK

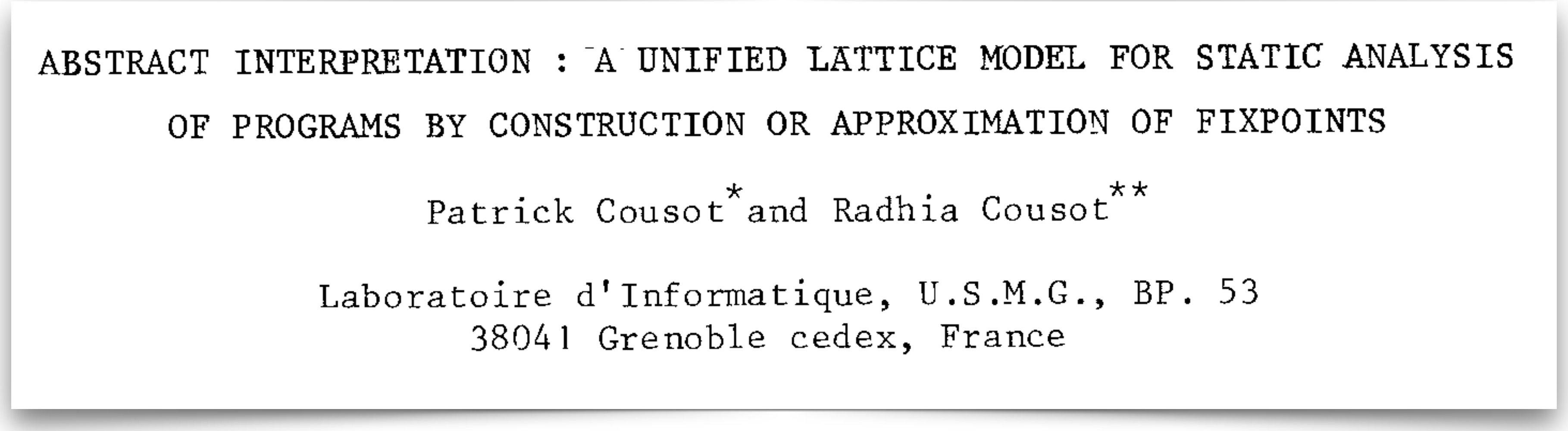
Program correctness and incorrectness are two sides of the same coin. As a programmer, even if you would like to have correctness, you might find yourself spending most of your time reasoning about incorrectness. This includes informal reasoning that people do while looking at or thinking about their 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.

The idea

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





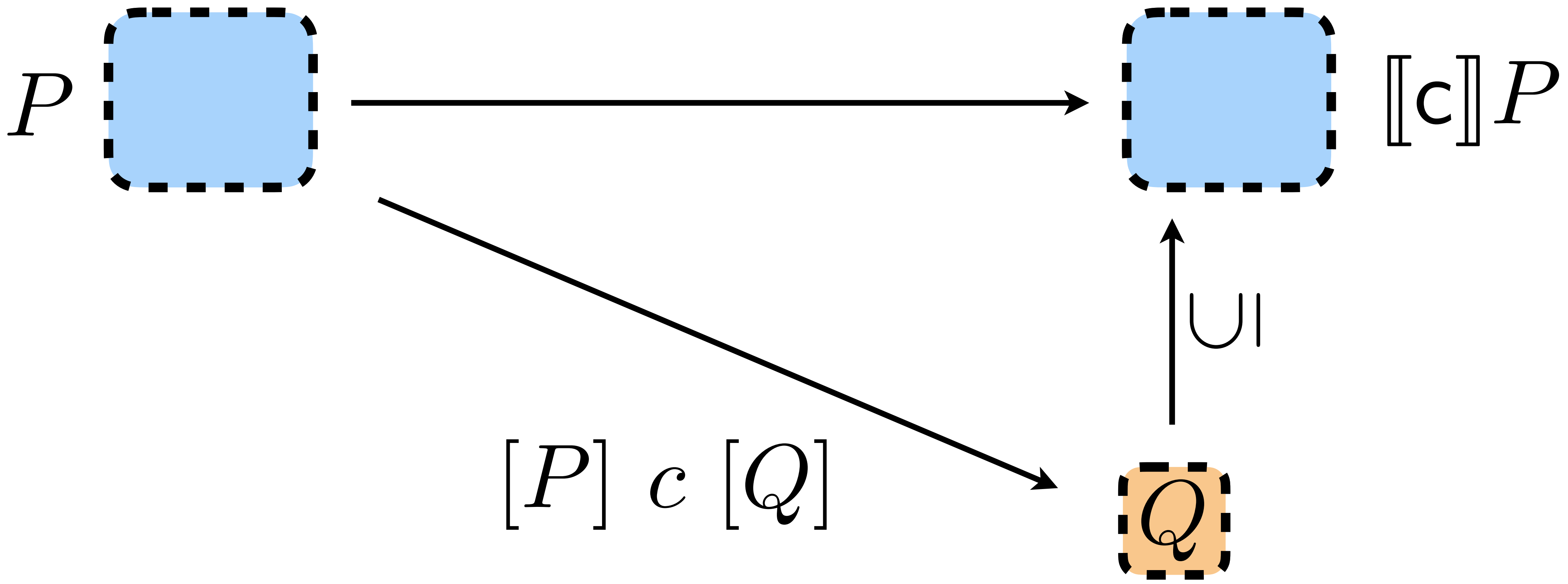

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

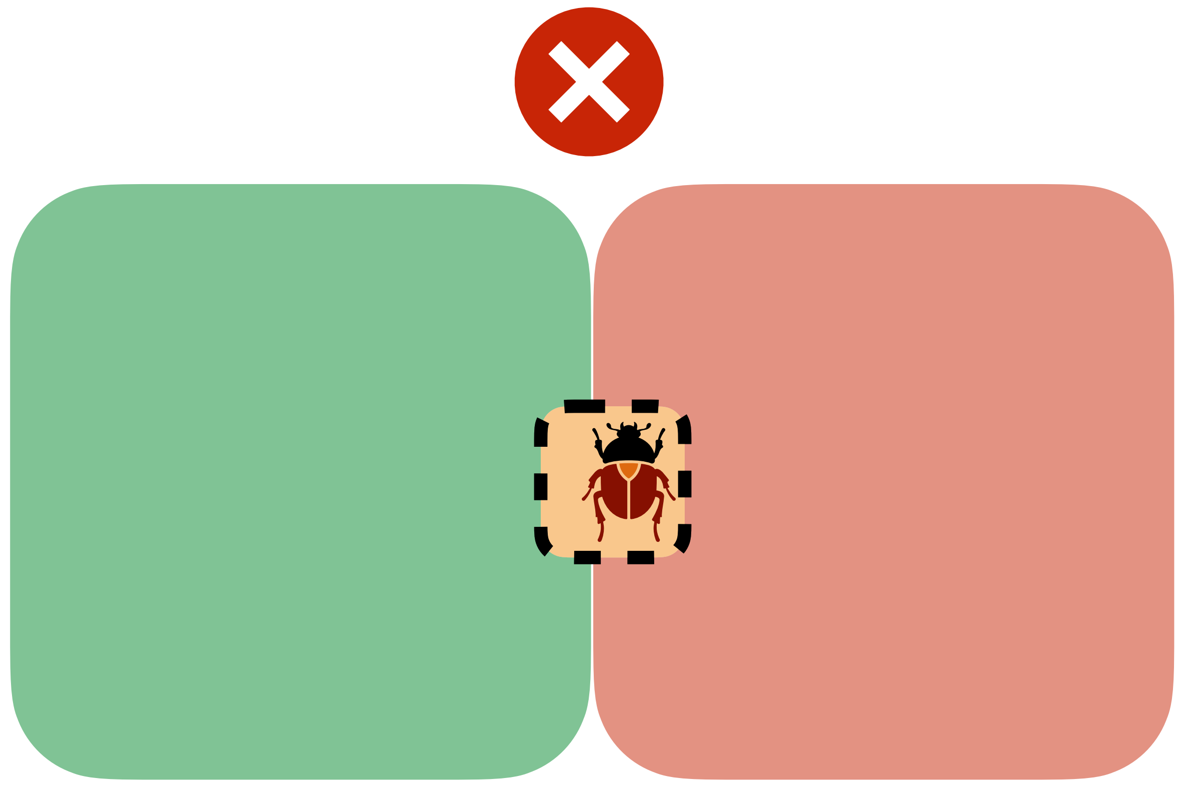
Patrick Cousot* and Radhia Cousot**

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

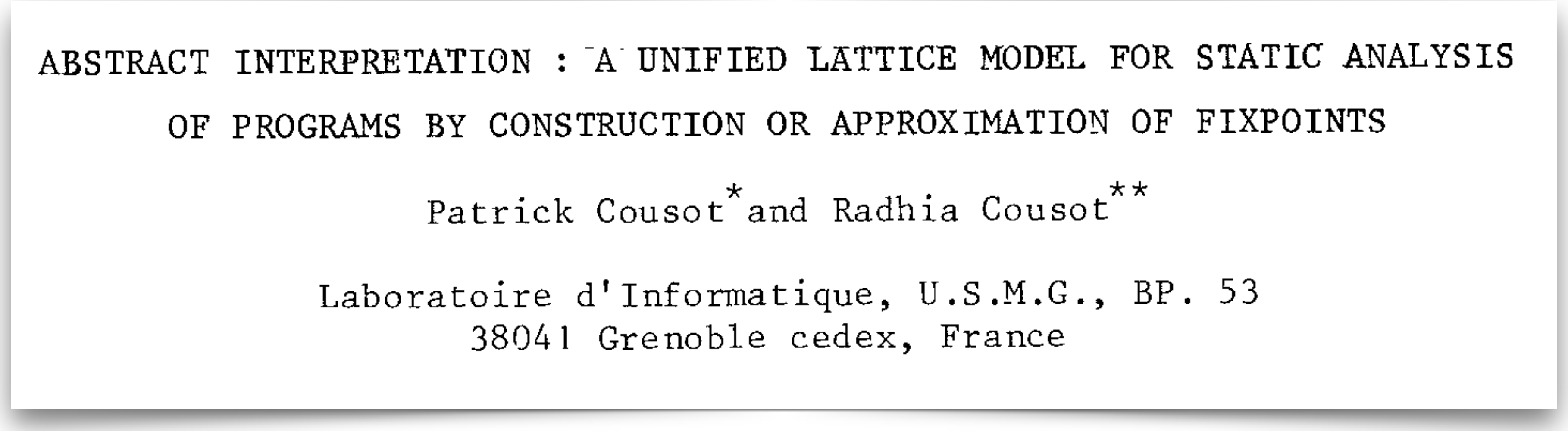
The idea

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





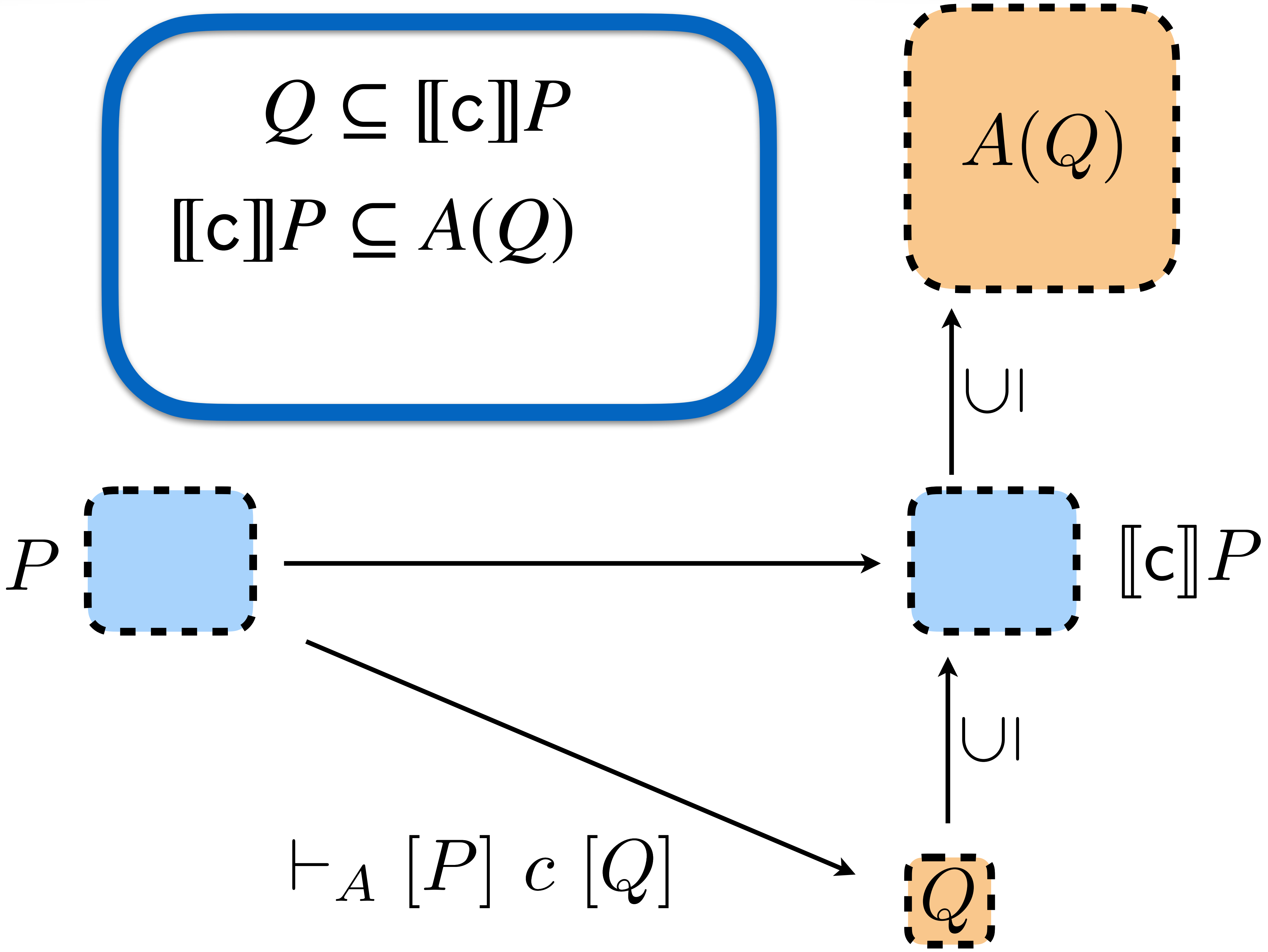
The idea

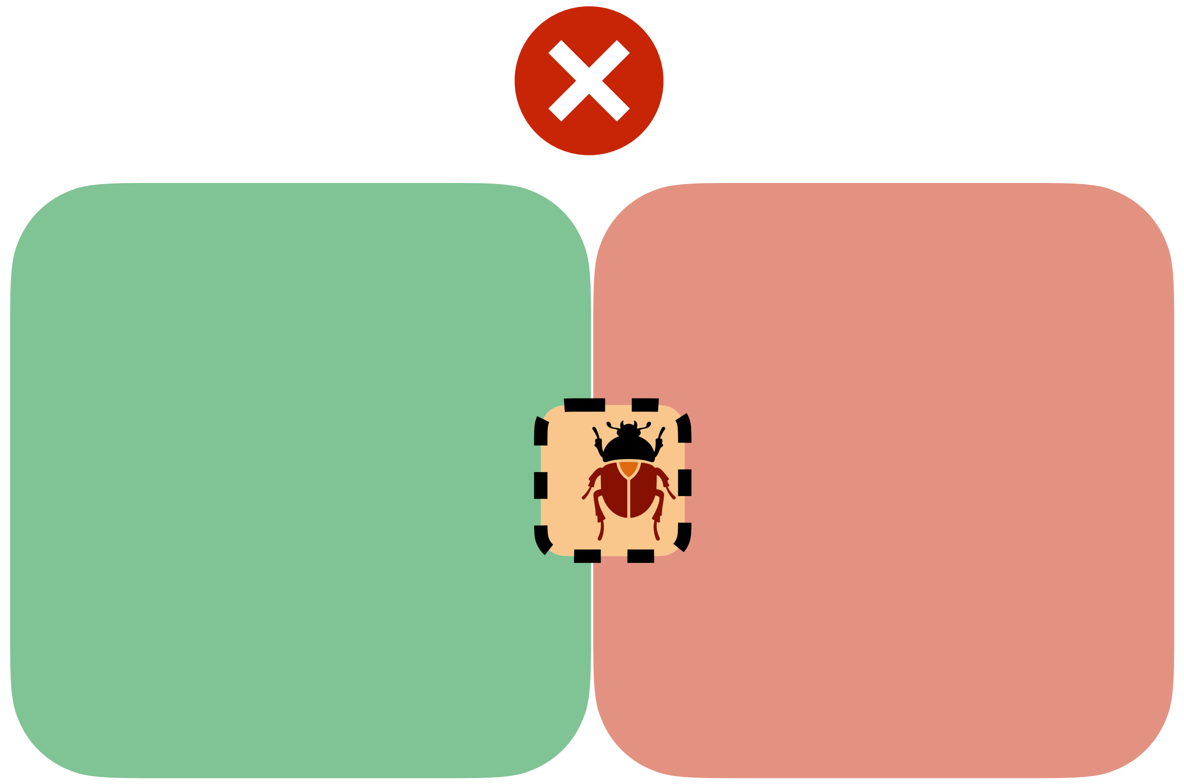


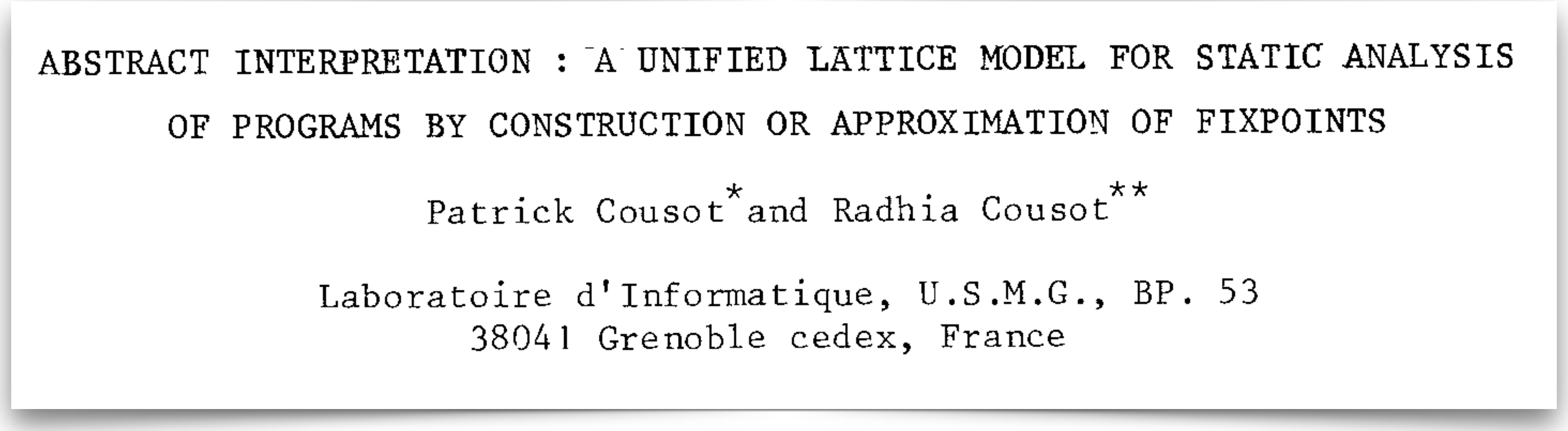
ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS
OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

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






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

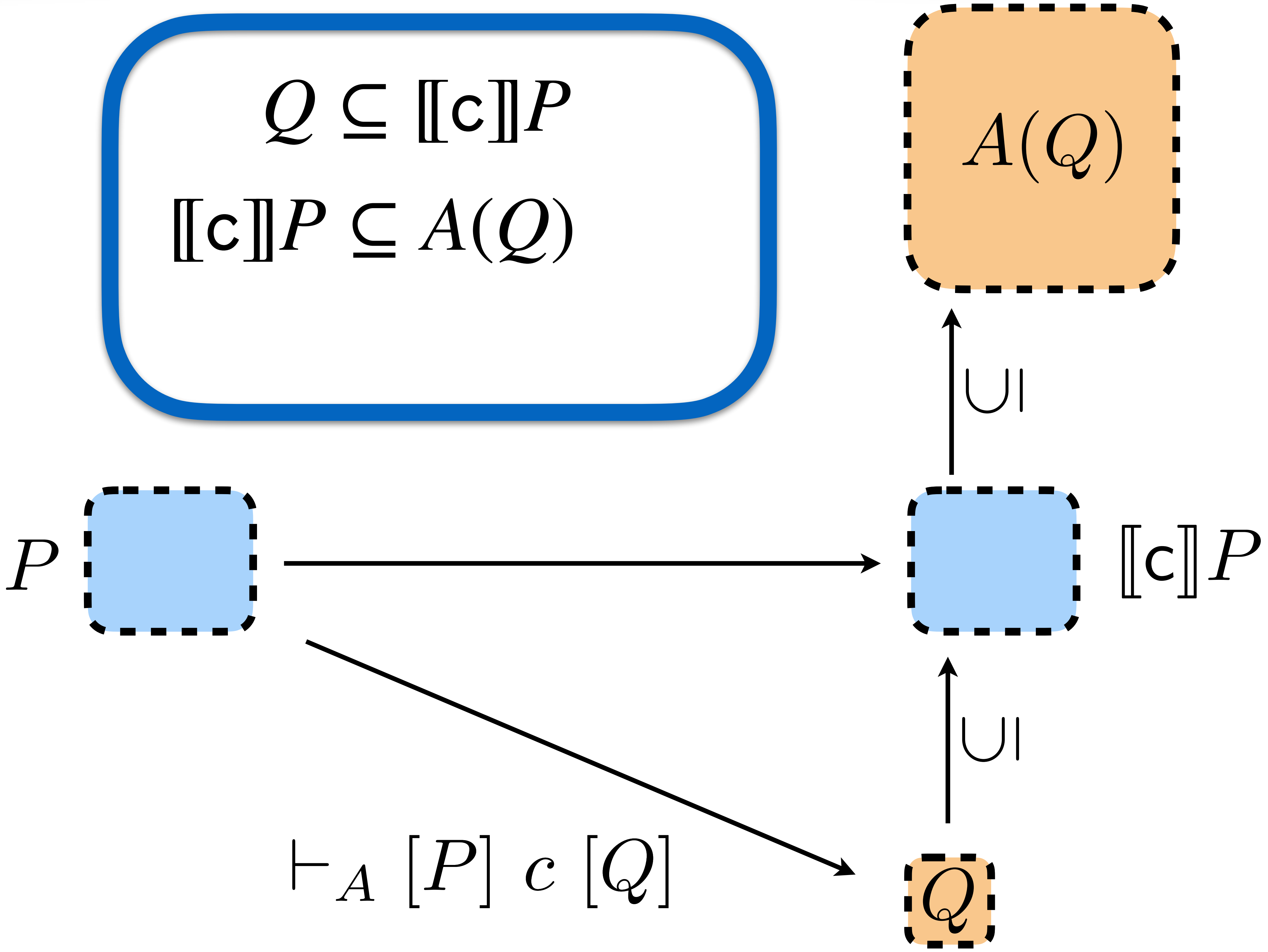
Patrick Cousot* and Radhia Cousot**

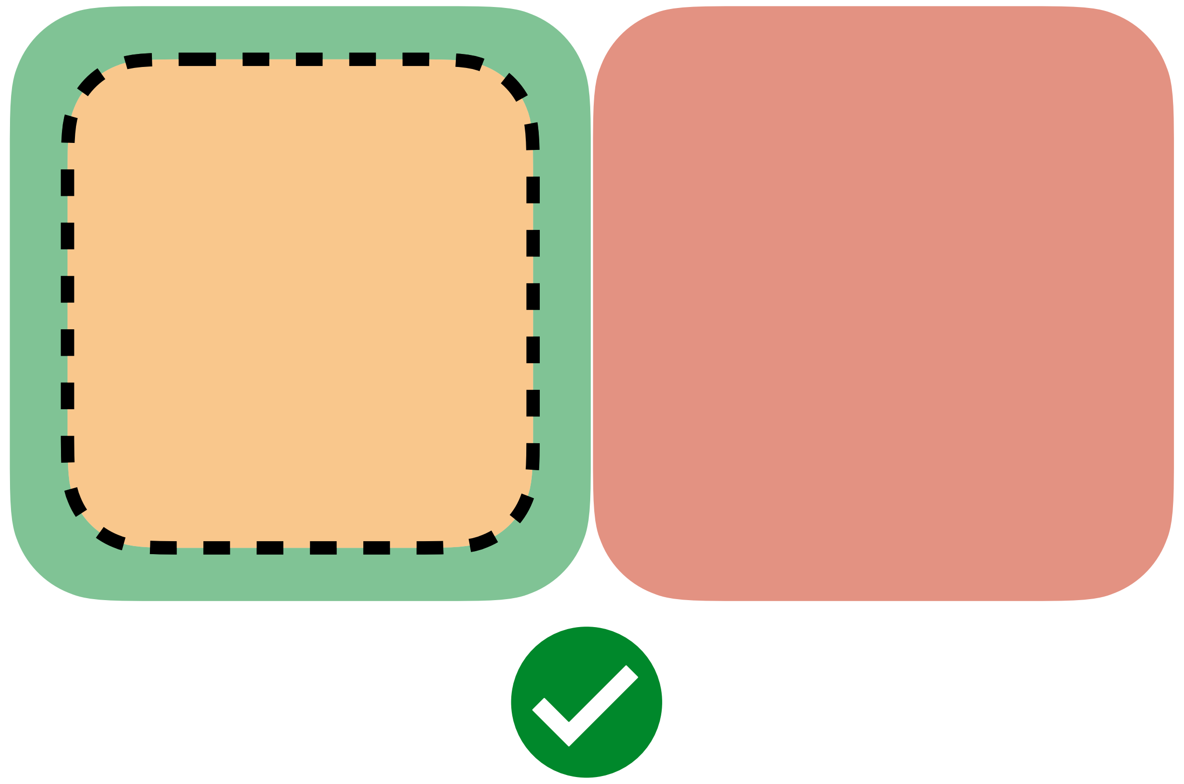
Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

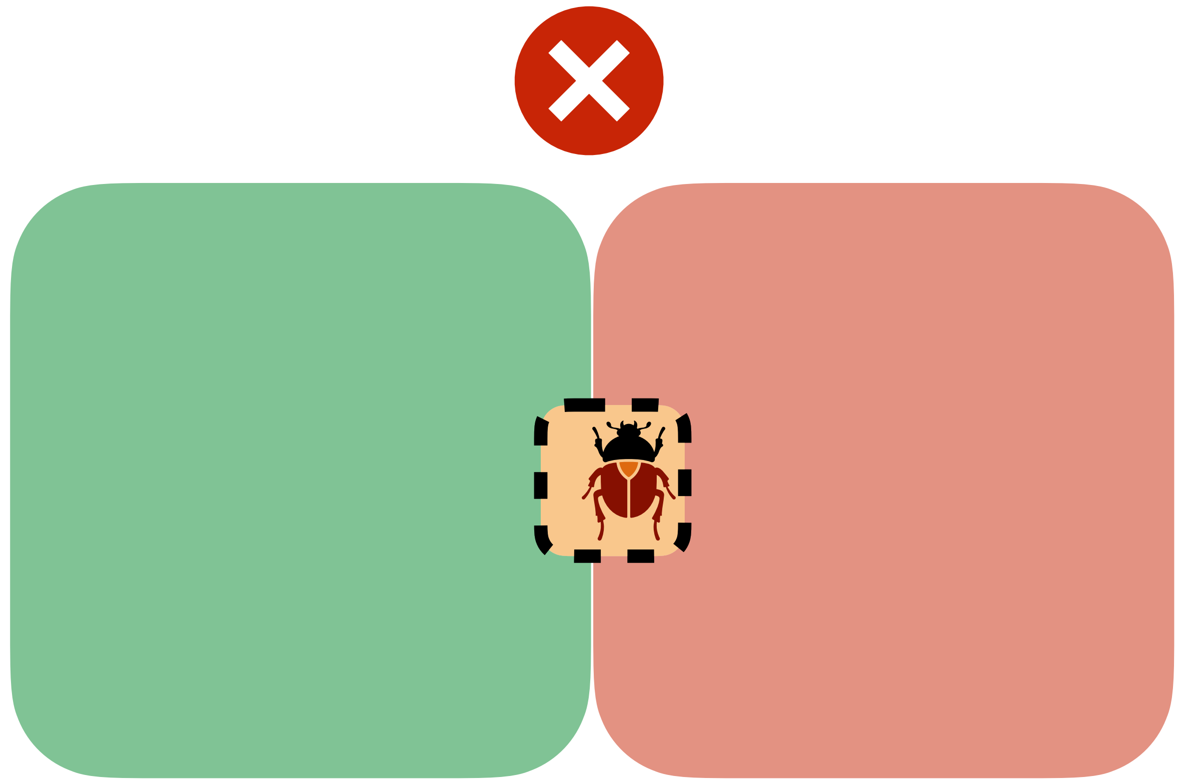
The idea

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

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







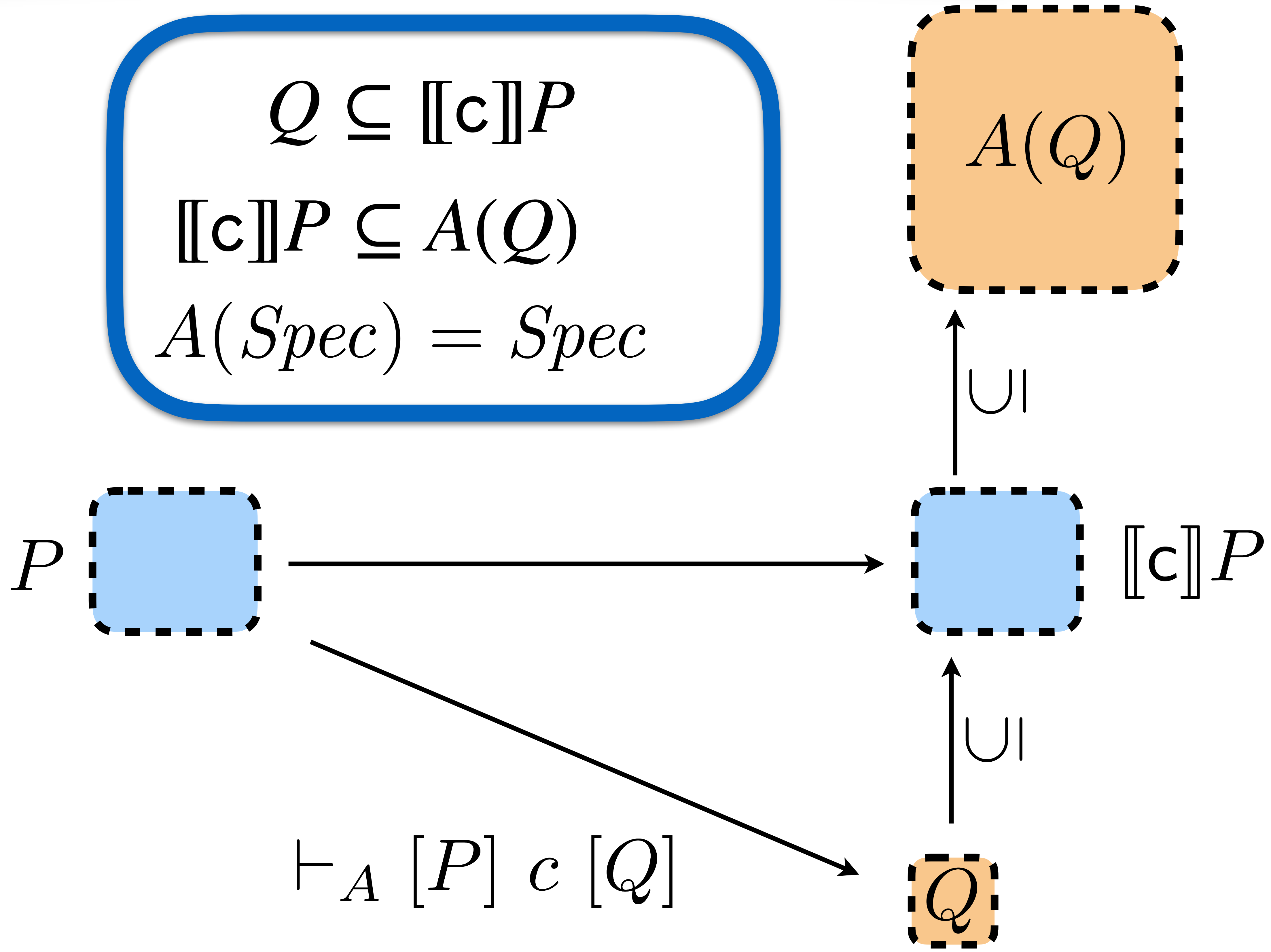
ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

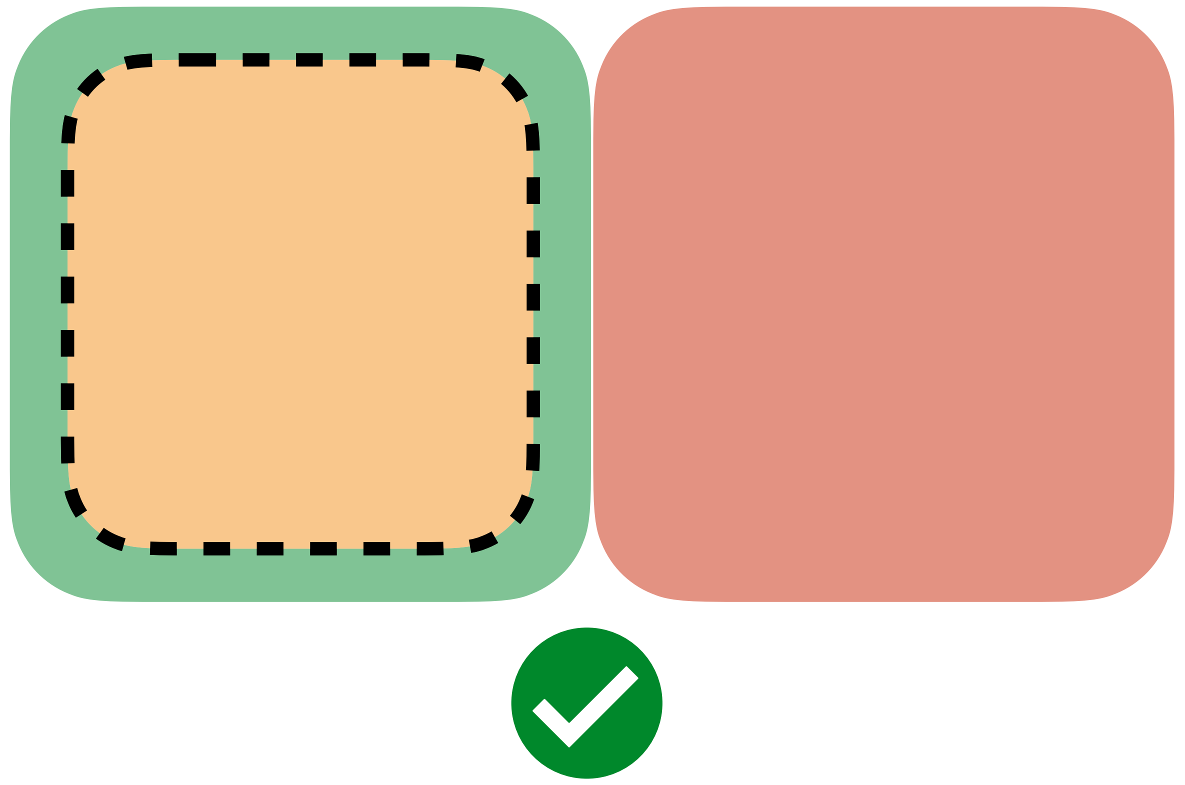
Patrick Cousot* and Radhia Cousot**

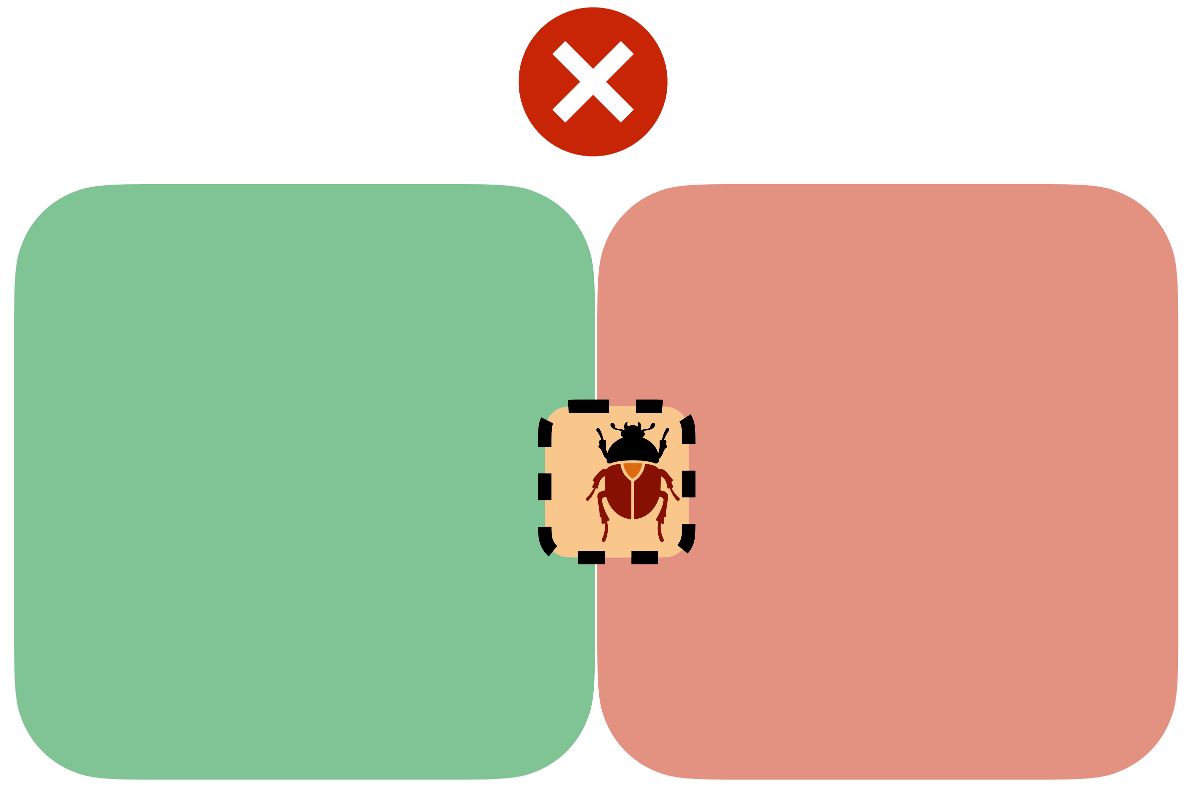
Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

The idea

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







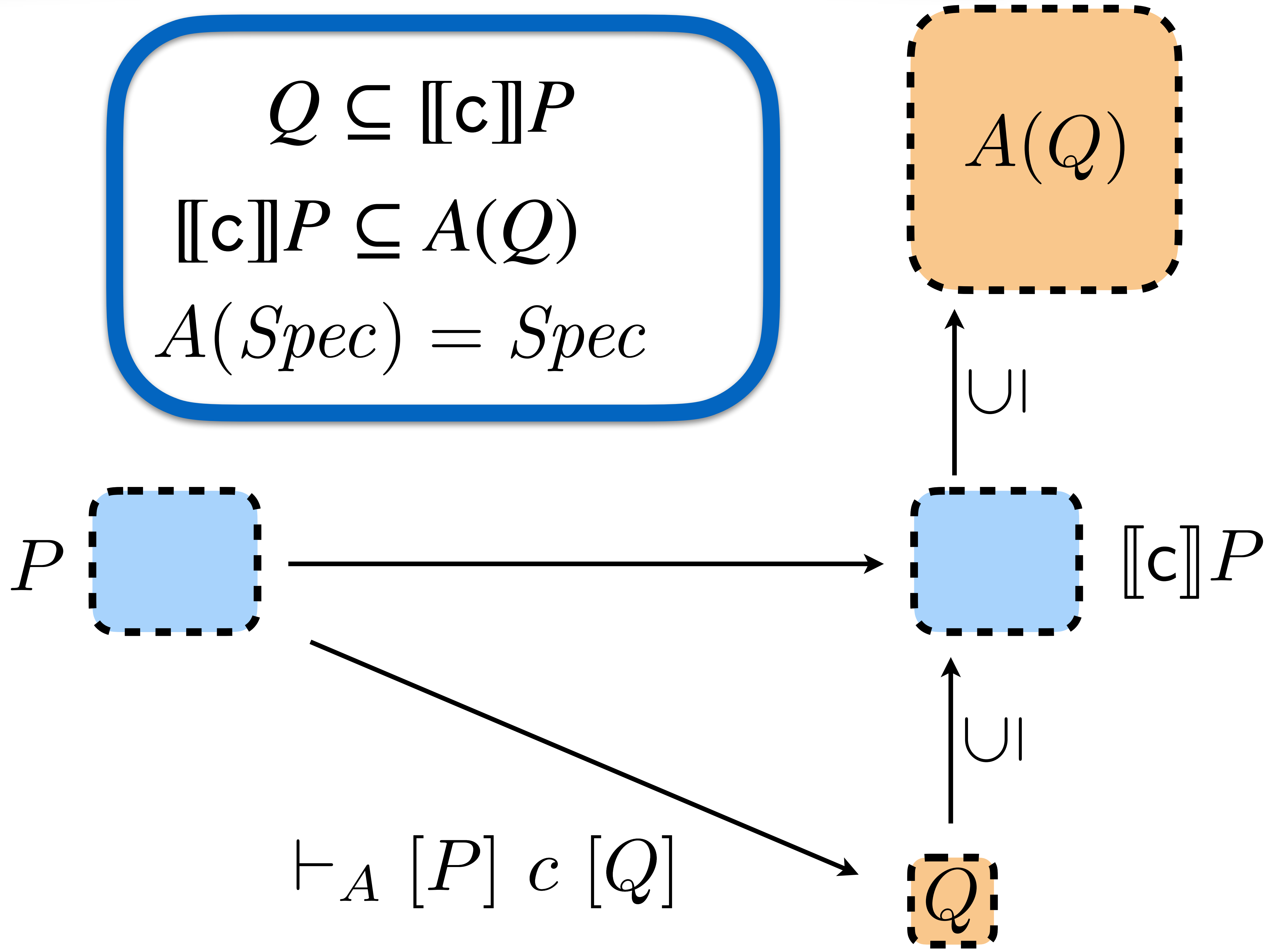
ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS
OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

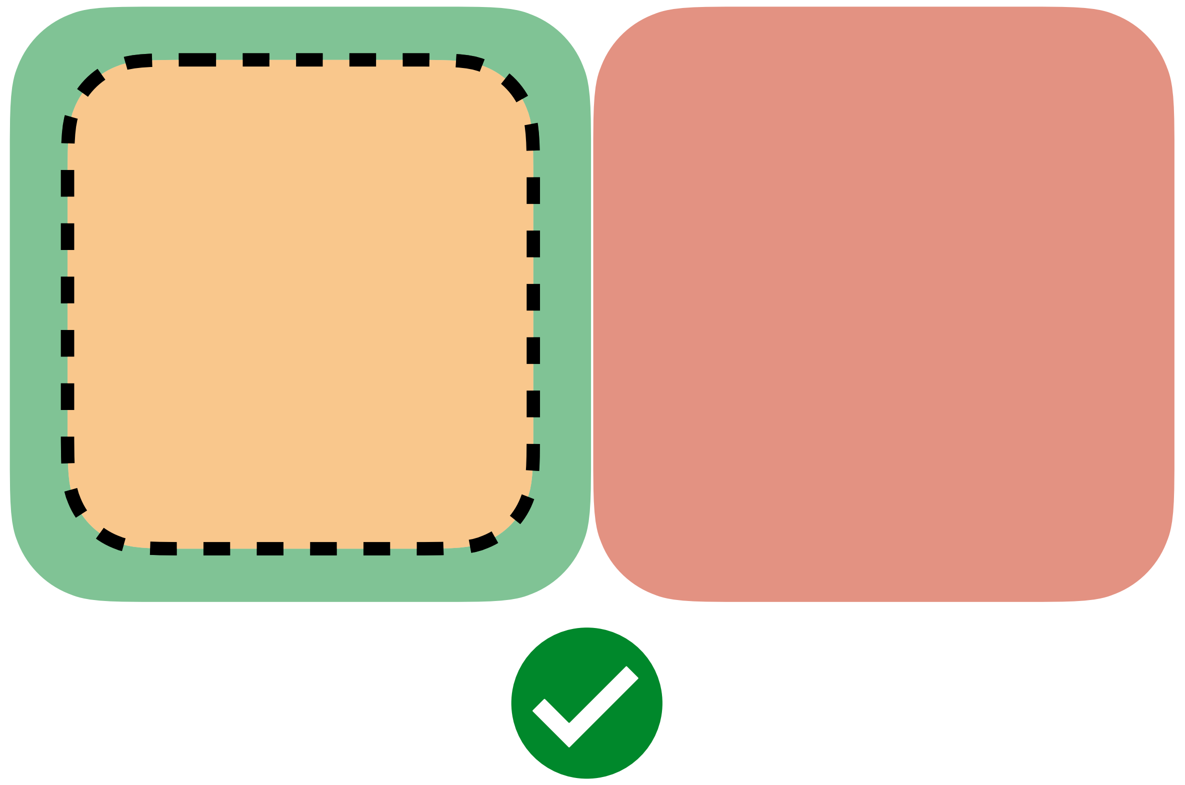
Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

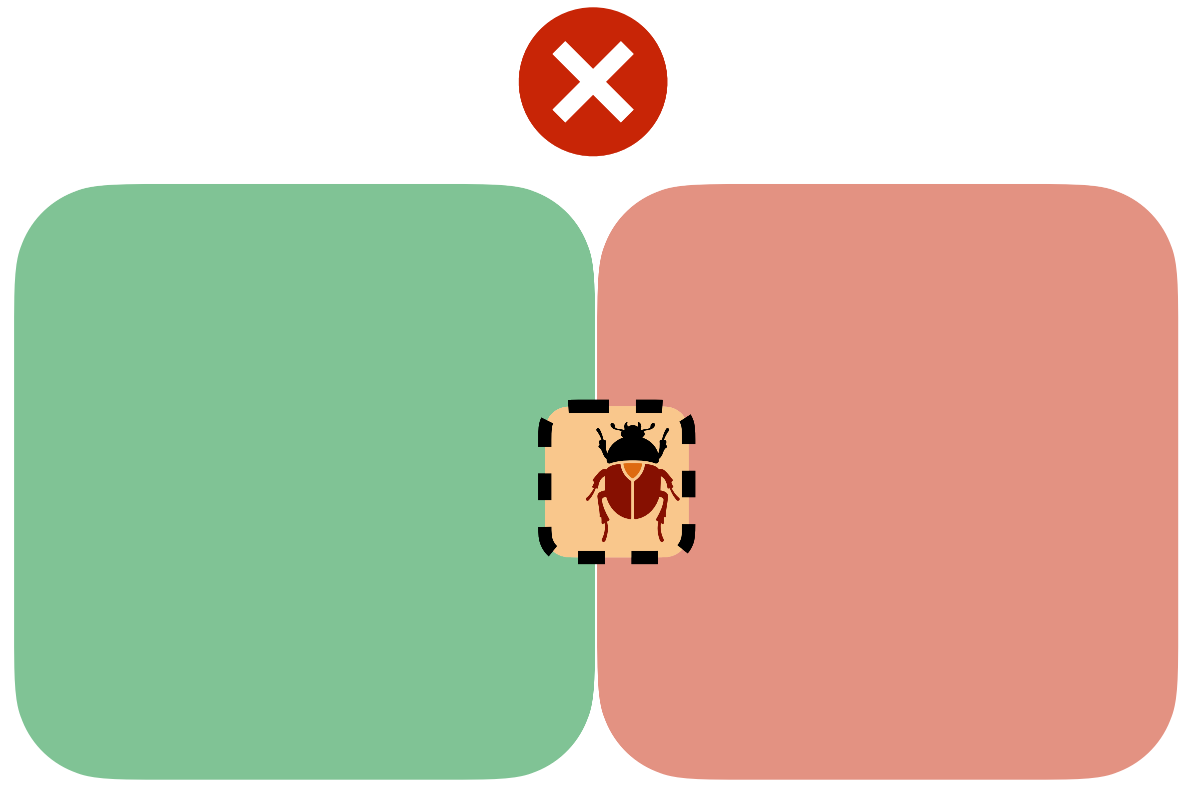
The idea

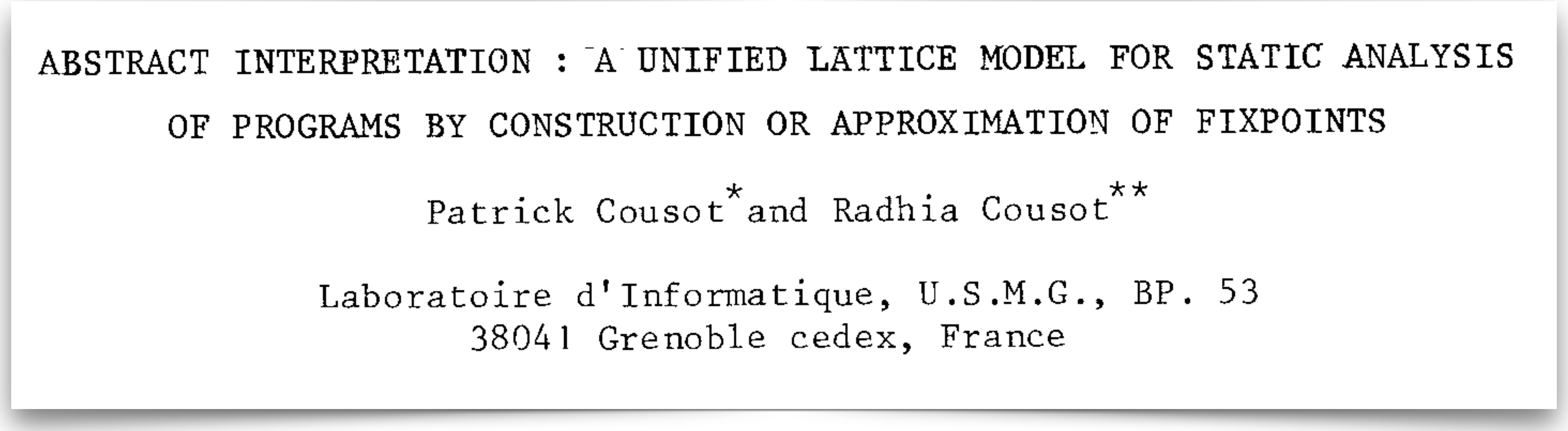
$$Q \subseteq [\![\mathsf{c}]\!]P$$
$$[\![\mathsf{c}]\!]P \subseteq A(Q)$$
$$A(Spec) = Spec$$



Local completeness





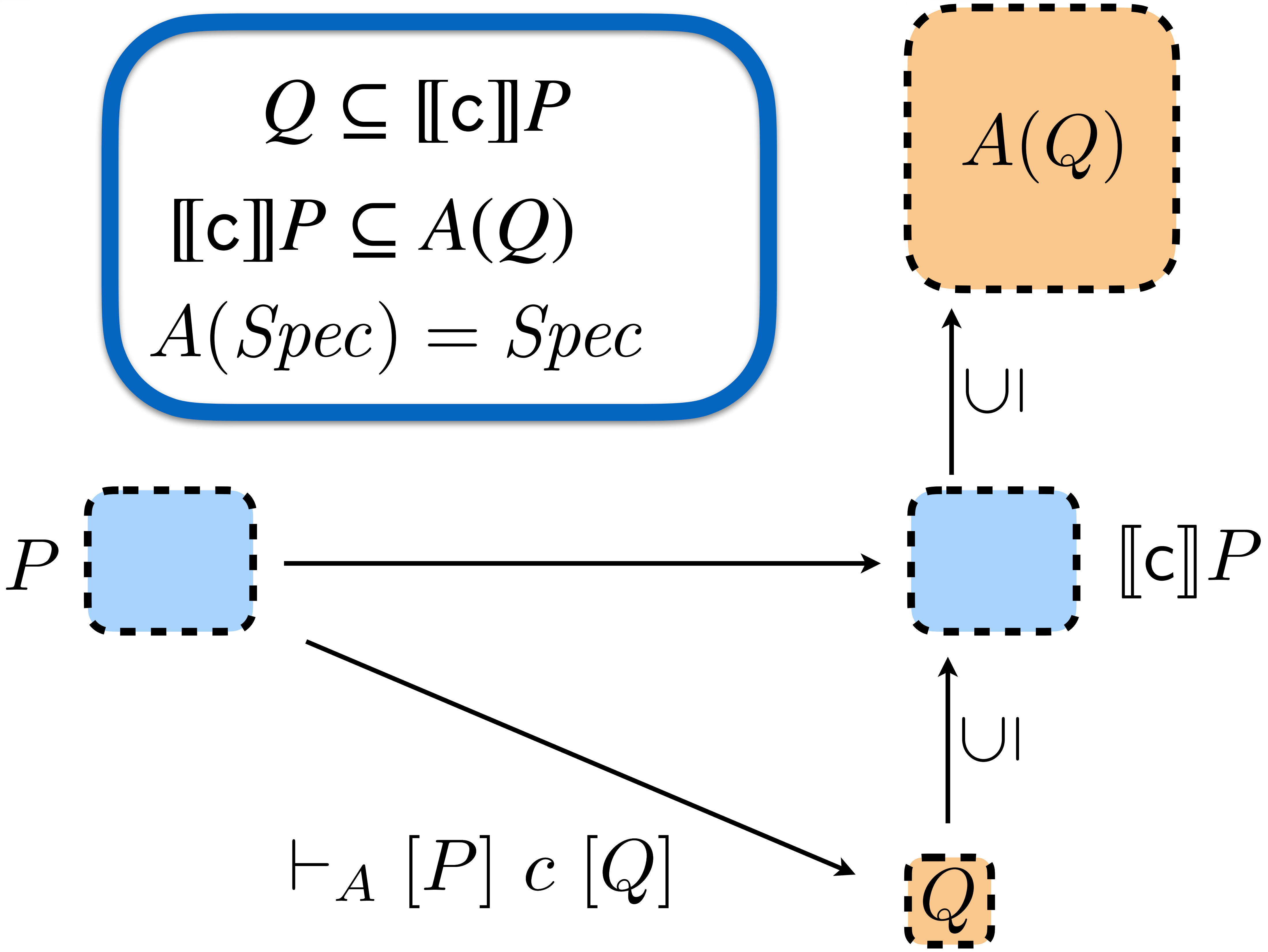

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS
OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

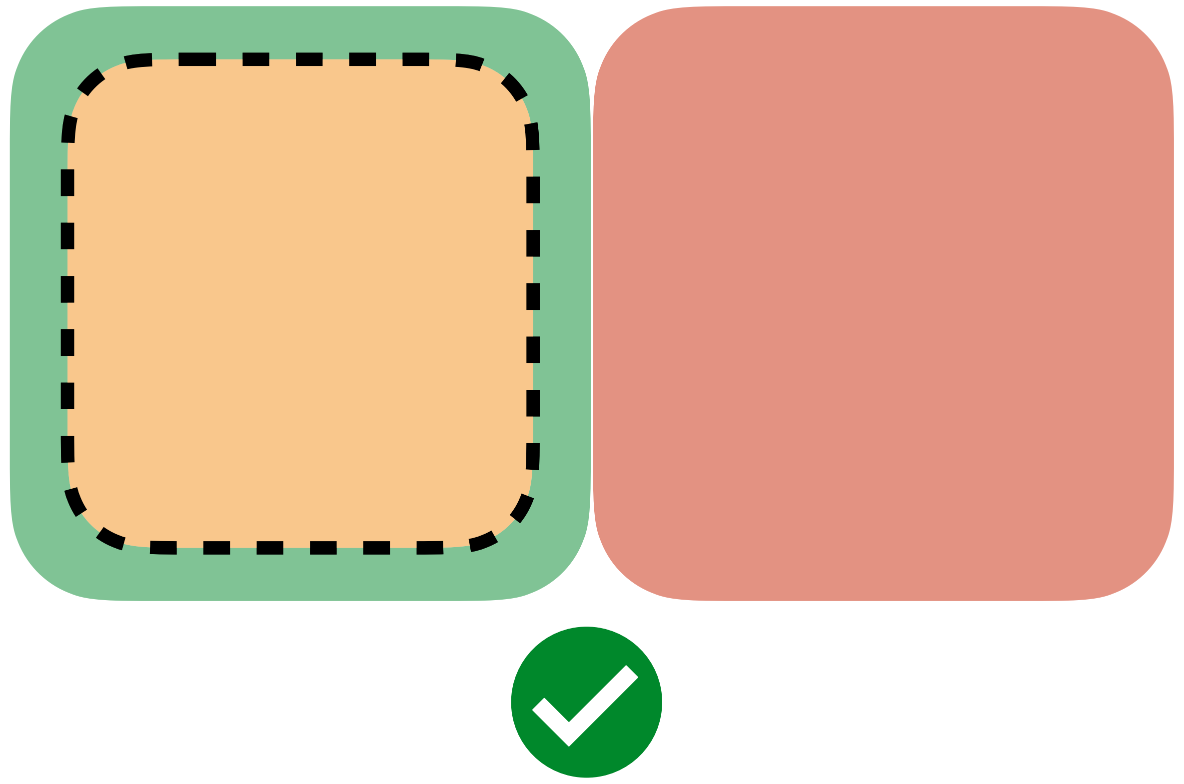
Patrick Cousot* and Radhia Cousot**

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

The idea

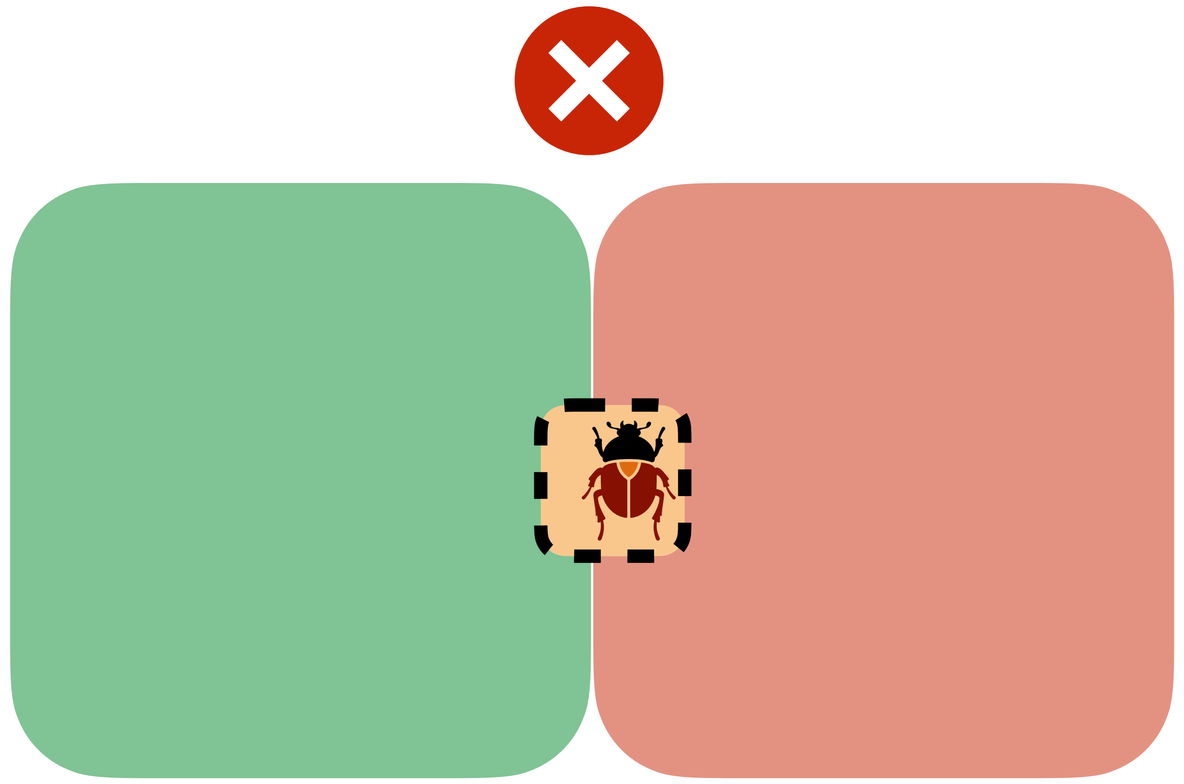
$$Q \subseteq [\![\mathsf{c}]\!] P$$
$$[\![\mathsf{c}]\!] P \subseteq A(Q)$$
$$A(Spec) = Spec$$

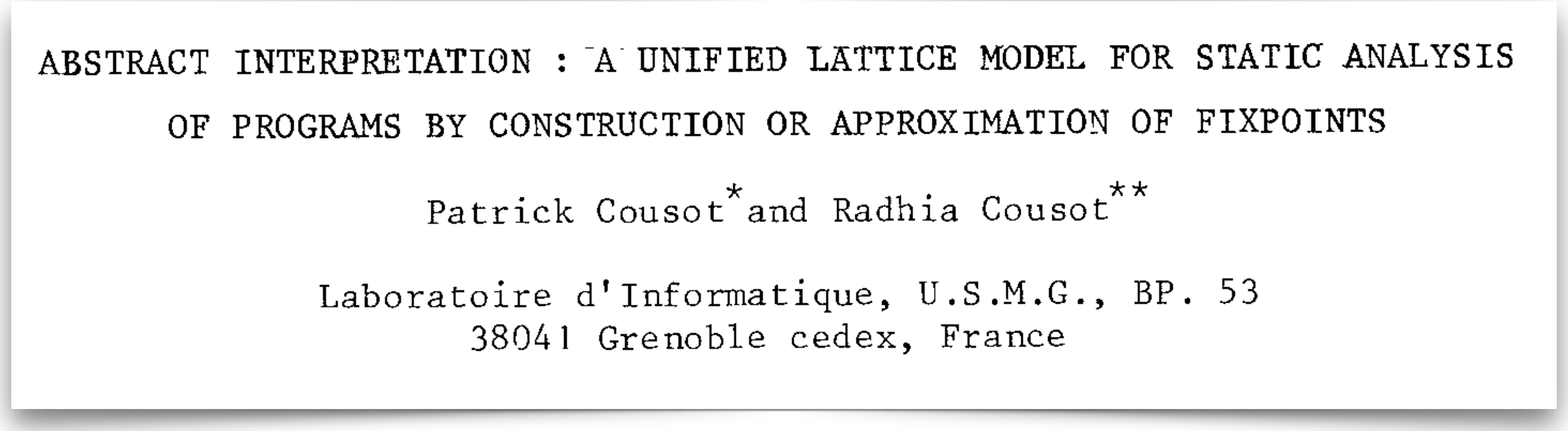




Local completeness

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





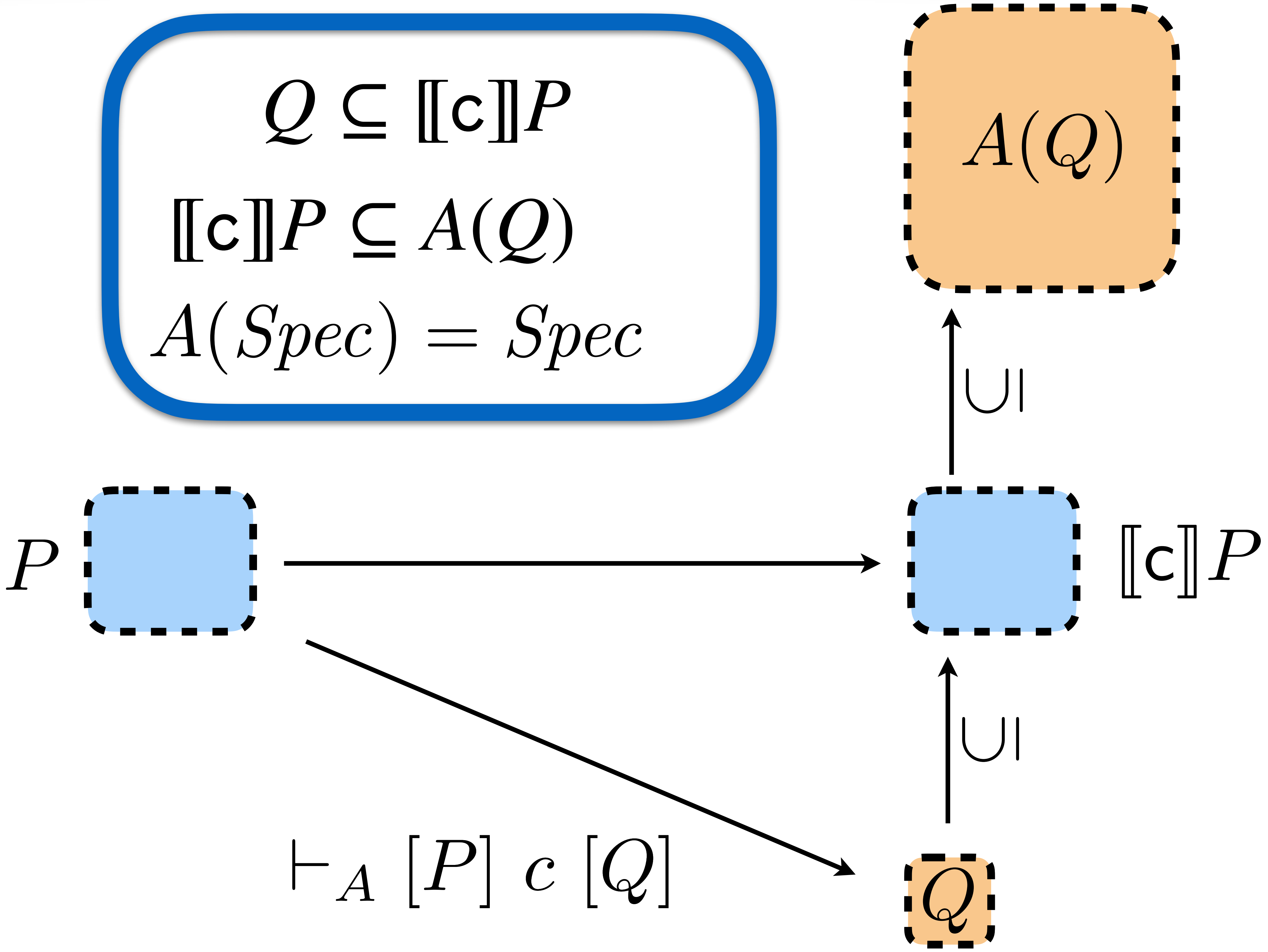
ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

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

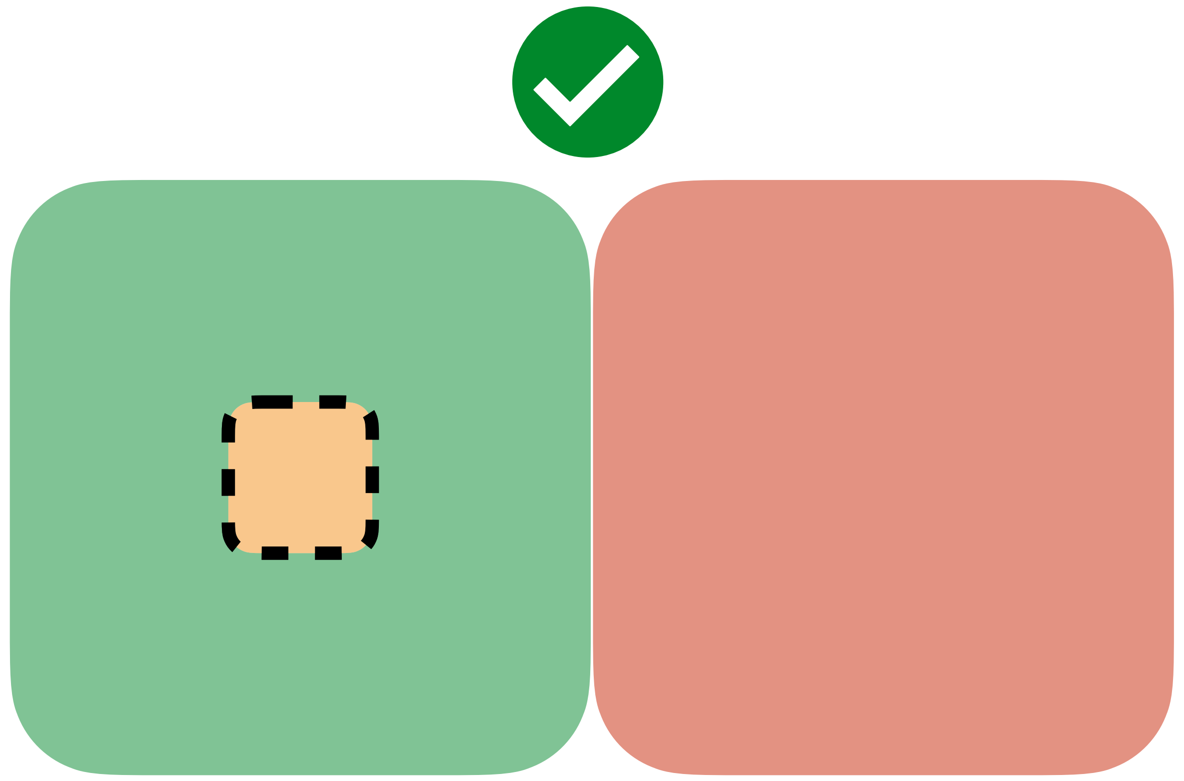
The idea

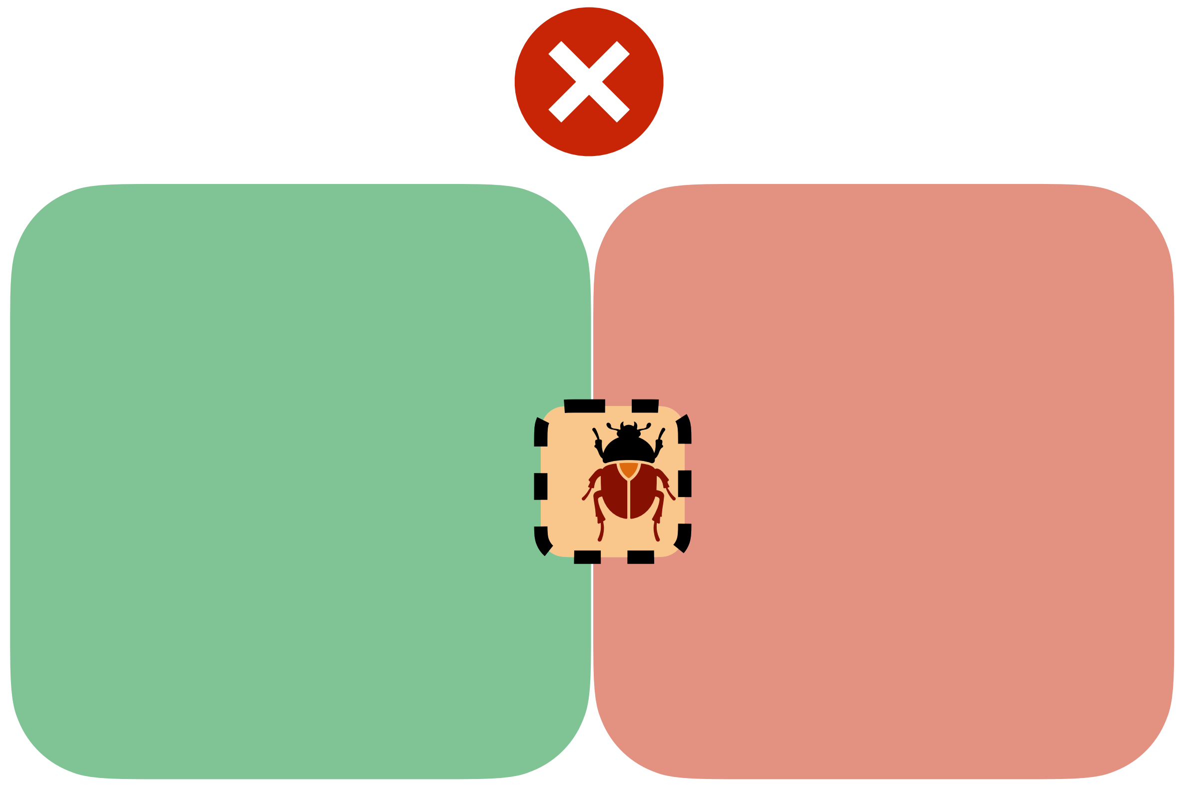
$$Q \subseteq [\![\mathsf{c}]\!]P$$
$$[\![\mathsf{c}]\!]P \subseteq A(Q)$$
$$A(\mathit{Spec}) = \mathit{Spec}$$



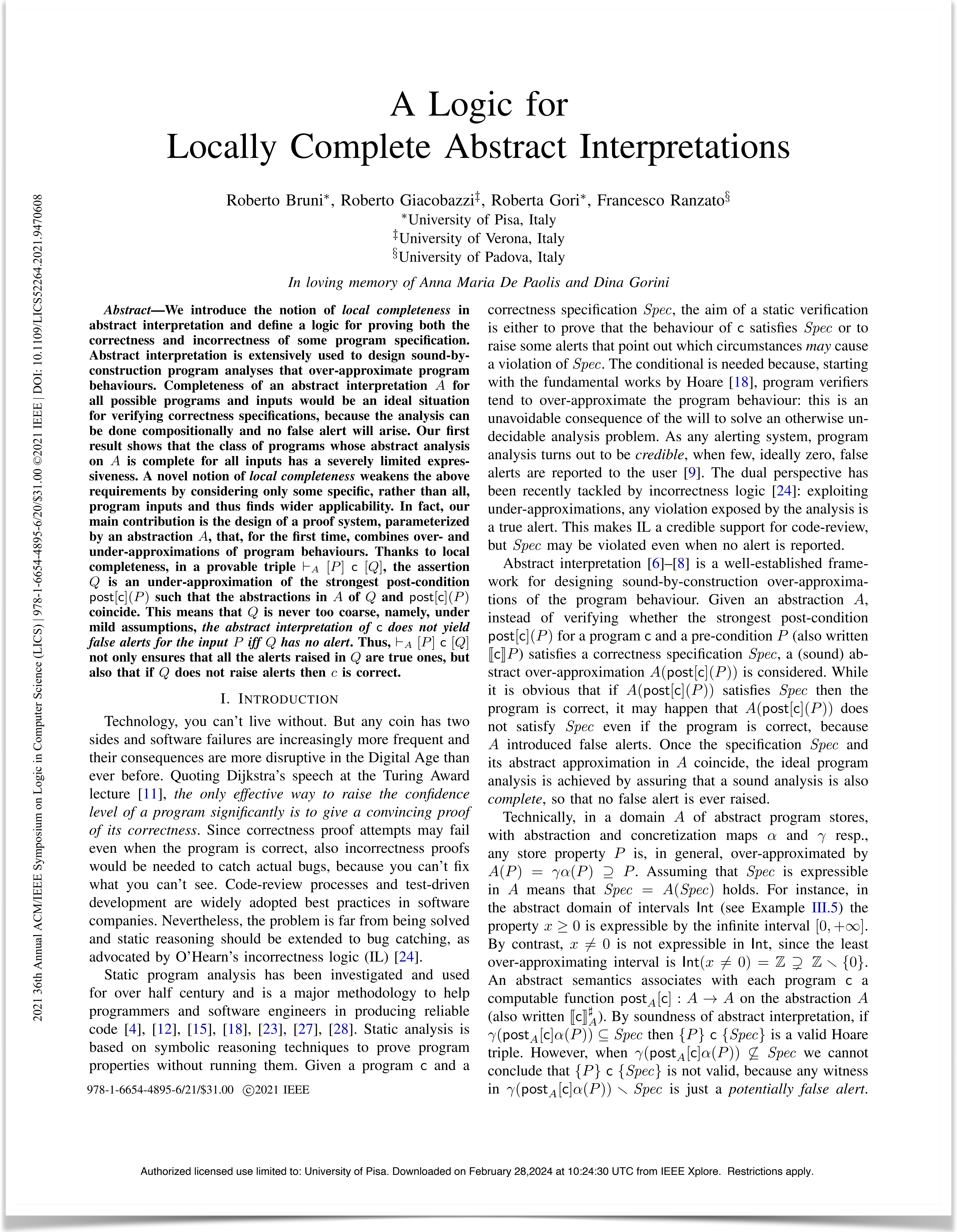
Local completeness

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





LICS 2021



2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) | 978-1-6654-4895-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/LICS52264.2021.9470608

A Logic for Locally Complete Abstract Interpretations

Roberto Bruni*, Roberto Giacobazzi‡, Roberta Gori*, Francesco Ranzato§
*University of Pisa, Italy
‡University of Verona, Italy
§University of Padova, Italy

In loving memory of Anna Maria De Paolis and Dina Gorini

***Abstract*—We introduce the notion of *local completeness* in abstract interpretation and define a logic for proving both the correctness and incorrectness of some program specification. Abstract interpretation is extensively used to design sound-by-construction program analyses that over-approximate program behaviours. Completeness of an abstract interpretation A for all possible programs and inputs would be an ideal situation for verifying correctness specifications, because the analysis can be done compositionally and no false alert will arise. Our first result shows that the class of programs whose abstract analysis on A is complete for all inputs has a severely limited expressiveness. A novel notion of *local completeness* weakens the above requirements by considering only some specific, rather than all, program inputs and thus finds wider applicability. In fact, our main contribution is the design of a proof system, parameterized by an abstraction A, that, for the first time, combines over- and under-approximations of program behaviours. Thanks to local completeness, in a provable triple $\vdash_A [P]\ \mathsf{c}\ [Q]$, the assertion Q is an under-approximation of the strongest post-condition $\mathsf{post}[\mathsf{c}](P)$ such that the abstractions in A of Q and $\mathsf{post}[\mathsf{c}](P)$ coincide. This means that Q is never too coarse, namely, under mild assumptions, *the abstract interpretation of* c *does not yield false alerts for the input* P *iff* Q *has no alert*. Thus, $\vdash_A [P]\ \mathsf{c}\ [Q]$ not only ensures that all the alerts raised in Q are true ones, but also that if Q does not raise alerts then c is correct.**

I. Introduction

Technology, you can't live without. But any coin has two sides and software failures are increasingly more frequent and their consequences are more disruptive in the Digital Age than ever before. Quoting Dijkstra's speech at the Turing Award lecture [11], *the only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness*. Since correctness proof attempts may fail even when the program is correct, also incorrectness proofs would be needed to catch actual bugs, because you can't fix what you can't see. Code-review processes and test-driven development are widely adopted best practices in software companies. Nevertheless, the problem is far from being solved and static reasoning should be extended to bug catching, as advocated by O'Hearn's incorrectness logic (IL) [24].

Static program analysis has been investigated and used for over half century and is a major methodology to help programmers and software engineers in producing reliable code [4], [12], [15], [18], [23], [27], [28]. Static analysis is based on symbolic reasoning techniques to prove program properties without running them. Given a program c and a correctness specification $Spec$, the aim of a static verification is either to prove that the behaviour of c satisfies $Spec$ or to raise some alerts that point out which circumstances *may* cause a violation of $Spec$. The conditional is needed because, starting with the fundamental works by Hoare [18], program verifiers tend to over-approximate the program behaviour: this is an unavoidable consequence of the will to solve an otherwise undecidable analysis problem. As any alerting system, program analysis turns out to be *credible*, when few, ideally zero, false alerts are reported to the user [9]. The dual perspective has been recently tackled by incorrectness logic [24]: exploiting under-approximations, any violation exposed by the analysis is a true alert. This makes IL a credible support for code-review, but $Spec$ may be violated even when no alert is reported.

Abstract interpretation [6]–[8] is a well-established framework for designing sound-by-construction over-approximations of the program behaviour. Given an abstraction A, instead of verifying whether the strongest post-condition $\mathsf{post}[\mathsf{c}](P)$ for a program c and a pre-condition P (also written $[\![\mathsf{c}]\!]P$) satisfies a correctness specification $Spec$, a (sound) abstract over-approximation $A(\mathsf{post}[\mathsf{c}](P))$ is considered. While it is obvious that if $A(\mathsf{post}[\mathsf{c}](P))$ satisfies $Spec$ then the program is correct, it may happen that $A(\mathsf{post}[\mathsf{c}](P))$ does not satisfy $Spec$ even if the program is correct, because A introduced false alerts. Once the specification $Spec$ and its abstract approximation in A coincide, the ideal program analysis is achieved by assuring that a sound analysis is also *complete*, so that no false alert is ever raised.

Technically, in a domain A of abstract program stores, with abstraction and concretization maps α and γ resp., any store property P is, in general, over-approximated by $A(P) = \gamma\alpha(P) \supseteq P$. Assuming that $Spec$ is expressible in A means that $Spec = A(Spec)$ holds. For instance, in the abstract domain of intervals Int (see Example III.5) the property $x \geq 0$ is expressible by the infinite interval $[0, +\infty]$. By contrast, $x \neq 0$ is not expressible in Int, since the least over-approximating interval is $\mathsf{Int}(x \neq 0) = \mathbb{Z} \supsetneq \mathbb{Z} \smallsetminus \{0\}$. An abstract semantics associates with each program c a computable function $\mathsf{post}_A[\mathsf{c}] : A \to A$ on the abstraction A (also written $[\![\mathsf{c}]\!]^\sharp_A$). By soundness of abstract interpretation, if $\gamma(\mathsf{post}_A[\mathsf{c}]\alpha(P)) \subseteq Spec$ then $\{P\}\ \mathsf{c}\ \{Spec\}$ is a valid Hoare triple. However, when $\gamma(\mathsf{post}_A[\mathsf{c}]\alpha(P)) \not\subseteq Spec$ we cannot conclude that $\{P\}\ \mathsf{c}\ \{Spec\}$ is not valid, because any witness in $\gamma(\mathsf{post}_A[\mathsf{c}]\alpha(P)) \smallsetminus Spec$ is just a *potentially false alert*.

any locally complete under approximation either proves the program correct or incorrect (without false positives)

Local completeness

Expressible specifications

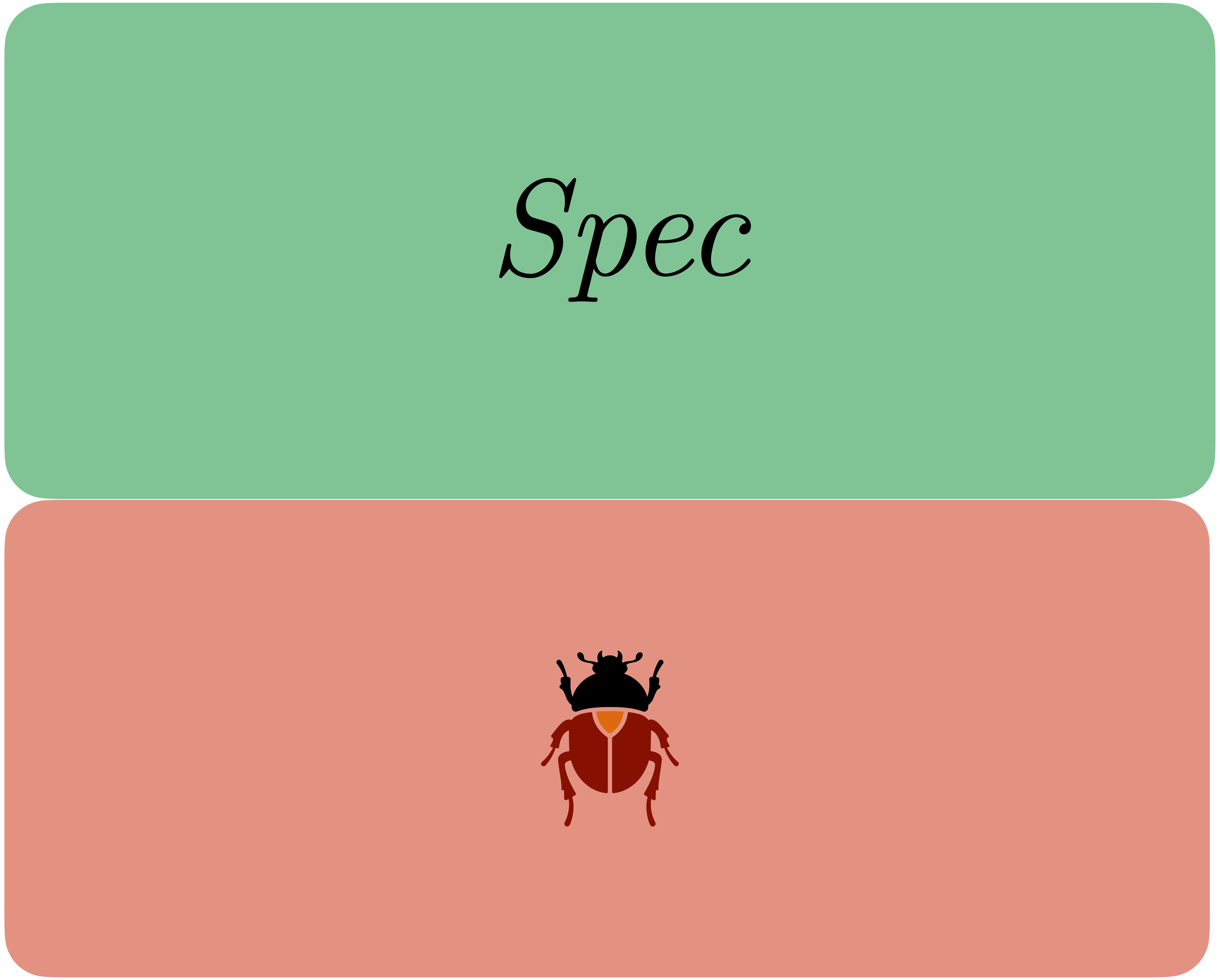
Assume $A(Spec) = Spec$

Take a post $Q \in C$

If $Q \not\subseteq Spec$
then $Q \subseteq A(Q) \not\subseteq Spec$

If $Q \subseteq Spec$
then $A(Q) \subseteq A(Spec) = Spec$

$$Q \subseteq Spec \Leftrightarrow A(Q) \subseteq Spec$$



Example

$\text{Int}(x \geq 0) = [0,\infty] = (x \geq 0)$

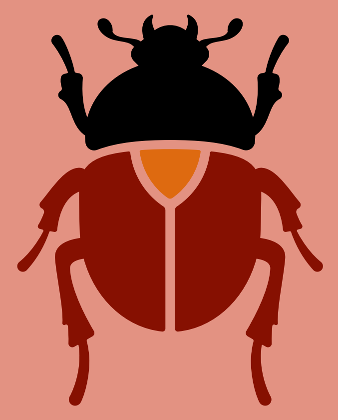
If $Q_1 \triangleq (|x| = 1)$
then $\text{Int}(Q_1) = [-1,1] \not\subseteq (x \geq 0)$

If $Q_2 \triangleq (x > 0 \land x \,\%\, 5 = 0)$
then $\text{Int}(Q_2) = [5,\infty] \subseteq (x \geq 0)$

$Q \subseteq Spec \Leftrightarrow A(Q) \subseteq Spec$

expressible specification

$x \geq 0$



The role of completeness

correctness of AI

if $[\![c]\!]^\#_A A(P) \subseteq Spec$

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

$[\![c]\!]P \subseteq A([\![c]\!]P) \subseteq [\![c]\!]^\#_A A(P) \subseteq Spec$

expressible specification

$Spec = A(Spec)$

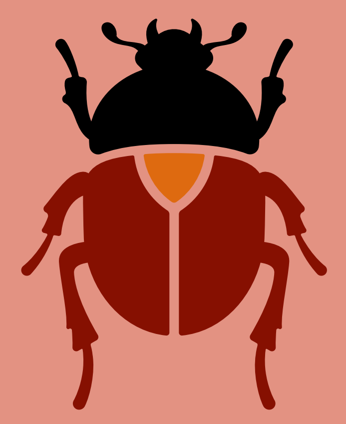
if completeness holds

if $[\![c]\!]^\#_A A(P) \not\subseteq Spec$

then $[\![c]\!]P \not\subseteq Spec$

$A([\![c]\!]P) = [\![c]\!]^\#_A A(P) \not\subseteq Spec$

$\Leftrightarrow [\![c]\!]P \not\subseteq Spec$



$[\![c]\!]P \subseteq Spec \Leftrightarrow [\![c]\!]^\#_A A(P) \subseteq Spec$

Example

$+, \times$ complete in Int

$$c \triangleq x := 3x\,;x := x + 2$$

$$P \triangleq (x \in \{1,3,6\})$$

$$[\![c]\!]^{\#}_{\mathsf{Int}}\mathsf{Int}(P) = [\![c]\!]^{\#}_{\mathsf{Int}}[1,6] = [5,20] \nsubseteq (x \leq 15)$$

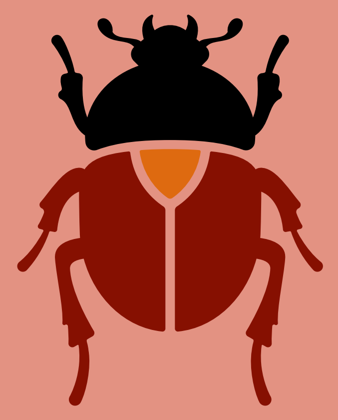
$$\Leftrightarrow$$

$$[\![c]\!]P \nsubseteq (x \leq 15)$$

However, not all elements in $[16,20]$ are true positives!

expressible specification

$$x \leq 15$$



Sources of incompleteness

Completeness is preserved by ; , ∪ and fix

Incompleteness can only be introduced by atomic commands e

?

assignments: settled for many domains

guards: troublesome

if the bca $[\![e]\!]^A$ is incomplete, then any (sound) $[\![e]\!]^\#_A$ is incomplete

i.e., incompleteness is an intrinsic property of a domain

Completeness for guards

Completeness equation: $\forall P\,.\; A([\![e]\!]P) = A([\![e]\!]A(P))$

must be a strict domain

Lemma. [*a necessary condition for complete guards*]
If a test b is complete in A, then b and $\neg b$ are expressible in A

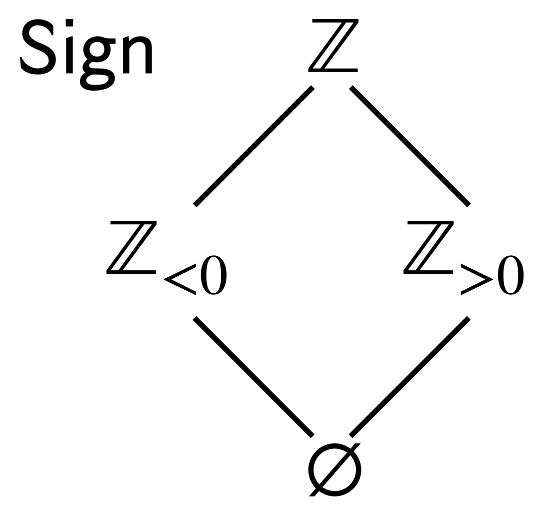
Proof. Assume b not expressible, take $P = b$ and show $\neg b$ is not complete.

Examples

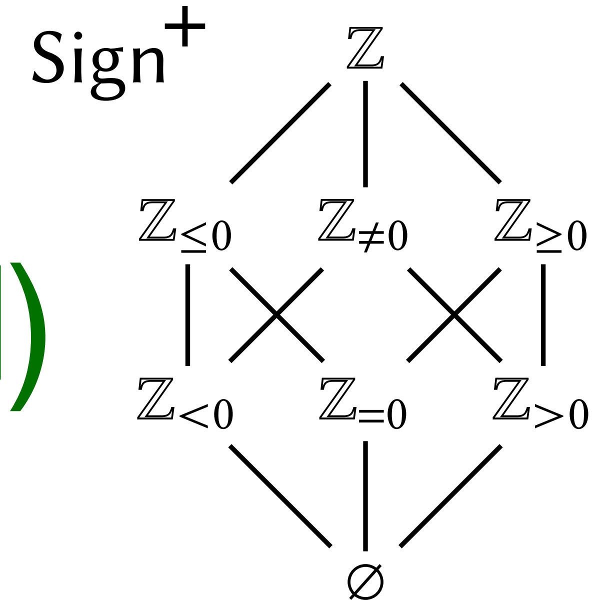
Int: the test $(x = 0)$ is not complete ($x \neq 0$ not expressible)

Int: the test $(x > 5)$ might be complete (but it is not)

Sign: the test $(x > 5)$ is not complete



Sign$^+$: the test $(x > 0)$ might be complete (and it is indeed)



Completeness for guards

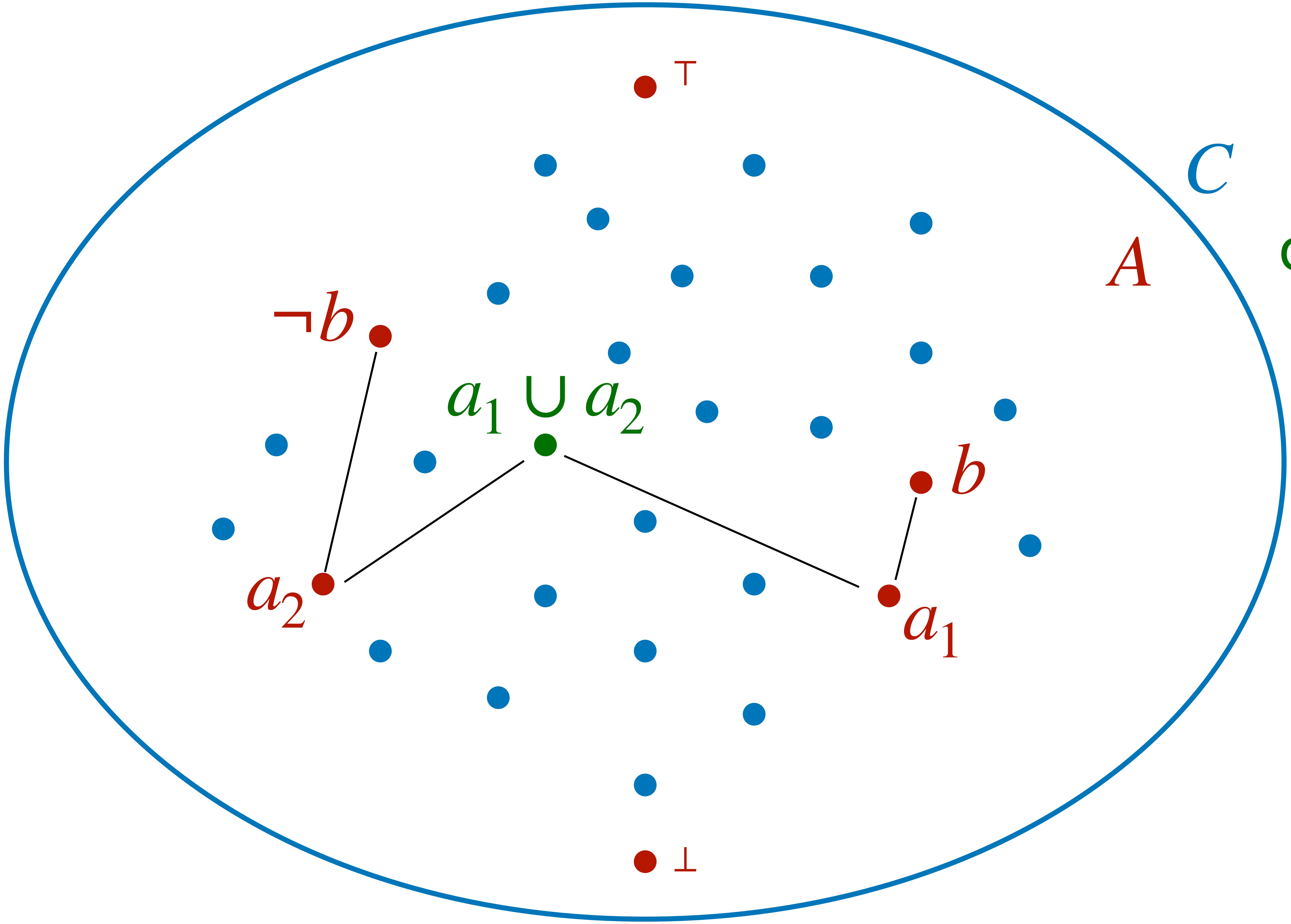
Th. [*a necessary and sufficient condition for complete guards*]
Let b and $\neg b$ be expressible in A.

The test b is complete in A

iff

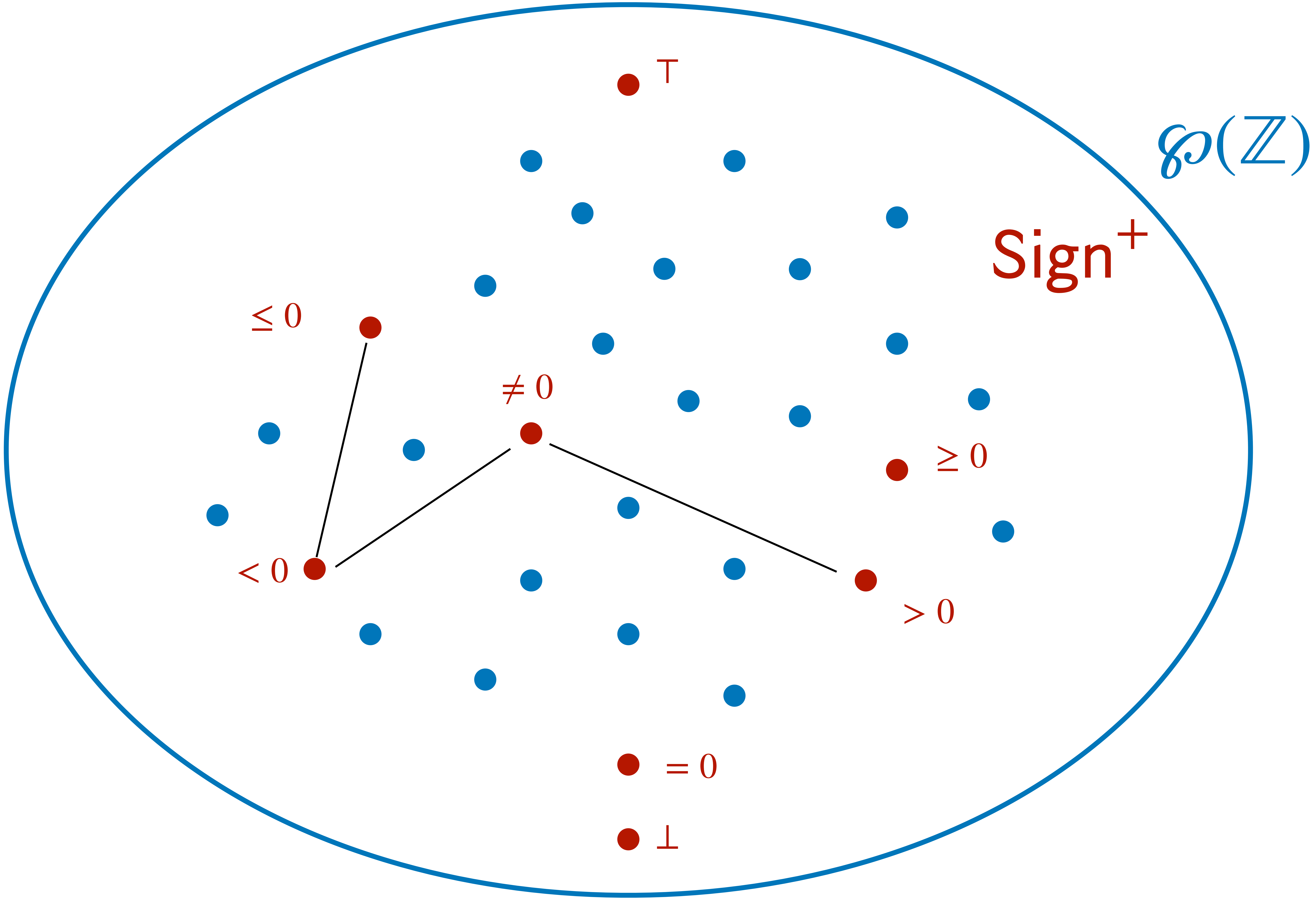
the join of any two abstract points below b and $\neg b$ is expressible.

Completeness illustrated



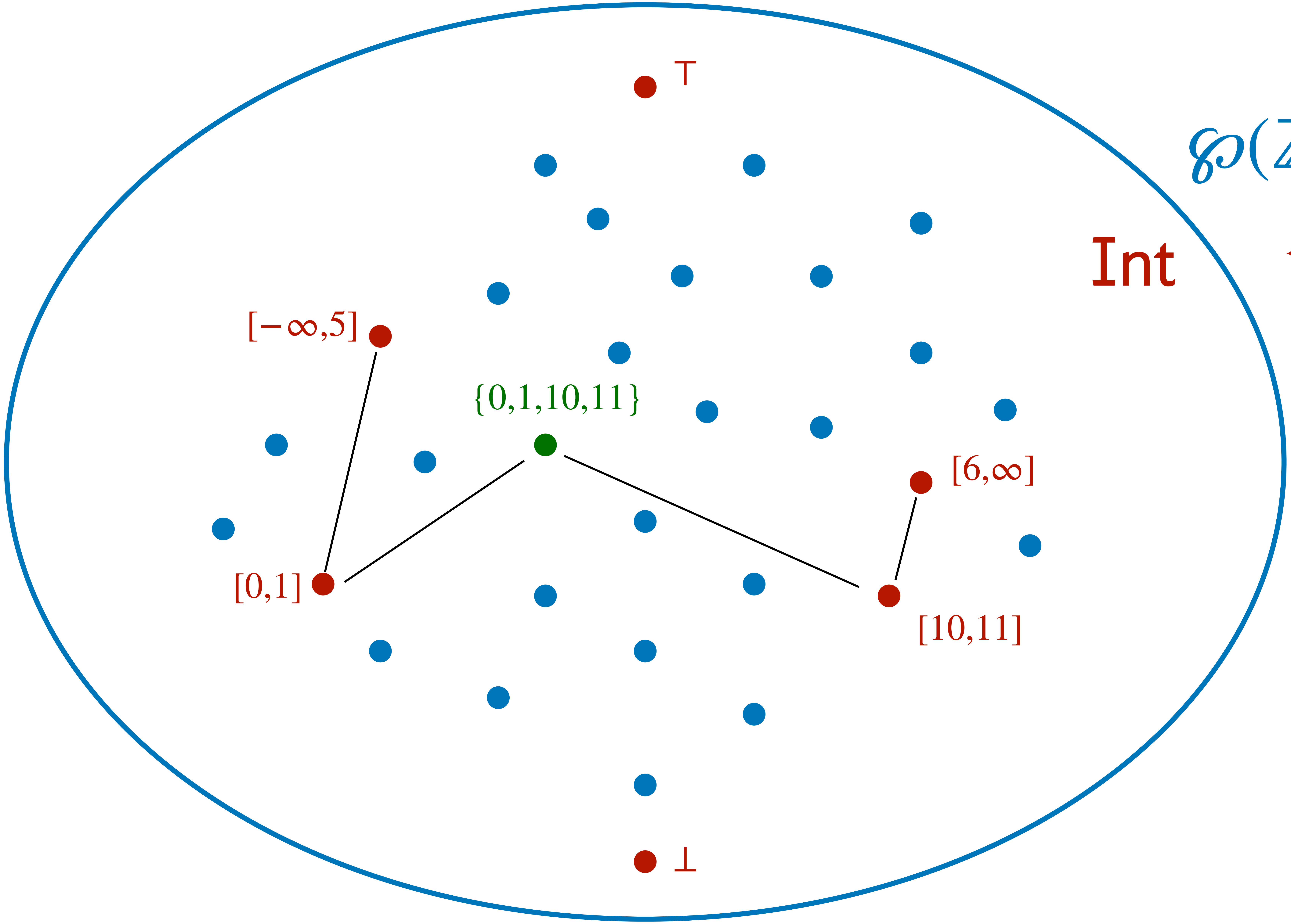
does $a_1 \cup a_2 \in A$?

Example



$\neq 0 \in \text{Sign}^+$

Example



$\{0,1,10,11\} \notin \text{Int}$

Incompleteness everywhere

Unfortunately, common tests are incomplete in most domains

One possibility:

take the most abstract domain A_b (called complete shell) that:

1) refines A, and
2) is complete for the test b

ok, but:

may cause a blow up (abstract domains are closed under meet)

operations that where complete in A may be incomplete in A_b

Local completeness

We don't need completeness for all inputs:

e.g., $b \triangleq (x > 0)$ is complete in Int for $P \triangleq \{-10,0,1,10\}$

Local completeness equation: ~~$\forall P$~~ $. \; A(⟦e⟧P) = A(⟦e⟧A(P))$

We say that e is locally complete in A for input P and write

$$\mathbb{C}_P^A(\mathsf{e})$$

Idea: we focus on completeness along traversed states
(which can be collected as underapproximation)

Local Completeness Logic (LCL)

O'Hearn's triples

pre condition

$$[P]\ c\ [Q]$$

post condition

any output matching the postcondition
can be reached by executing the command
on some input matching the precondition

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

under approximation!

includes just reachable states

LCL triples

pre condition

$$\vdash_A [P]\ c\ [Q]$$

abstract domain

post condition

any output matching the postcondition
can be reached by executing the command
on some input matching the precondition
+
for any input matching the precondition executing the command establishes the abstraction of the postcondition

can include non reachable states

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

over approximation!

under approximation!

includes just reachable states

Atomic commands

local completeness requirement

$A([\![e]\!]P) = A([\![e]\!]A(P))$

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

Floyd's rule for assignments

$$\frac{\mathbb{C}^A_P(a)}{\vdash_A [P]\ x := a\ [\exists x'.\, P[x'/x] \wedge x = a[x'/x]]}$$

IL rule for tests

$$\frac{\mathbb{C}^A_P(b)}{\vdash_A [P]\ b?\ [P \wedge b]}$$

Atomic commands

Floyd's rule for assignments

$$\frac{\mathbb{C}_P^A(a)}{\vdash_A [P]\ x := a\ [\exists x' . P[x'/x] \wedge x = a[x'/x]]}$$

$$\vdash_{\mathsf{Int}} [x \in \{-7,7\}]\ x := 3x + 1\ [x \in \{-20,22\}]$$

sum and product are complete in Int

Atomic commands

IL rule for tests

$$\frac{\mathbb{C}^A_P(b)}{\vdash_A [P]\ b?\ [P \wedge b]}$$

locally incomplete in Int !

$$\vdash_{\mathsf{Int}} [x \in \{-7,7\}]\ x > 0?\ [x \in \{7\}]\ ?$$ ✗

$\mathsf{Int}([\![x > 0?]\!]\{-7,7\}) = \mathsf{Int}(\{7\}) = [7,7] \neq [1,7] = \mathsf{Int}([1,7]) = \mathsf{Int}([\![x > 0?]\!][-7,7]) = \mathsf{Int}([\![x > 0?]\!]\mathsf{Int}(\{-7,7\}))$

$\mathsf{Int}([\![x > 0?]\!]\{-7,1,7\}) = \mathsf{Int}(\{1,7\}) = [1,7] = \mathsf{Int}([1,7]) = \mathsf{Int}([\![x > 0?]\!][-7,7]) = \mathsf{Int}([\![x > 0?]\!]\mathsf{Int}(\{-7,1,7\}))$

$$\vdash_{\mathsf{Int}} [x \in \{-7,1,7\}]\ x > 0?\ [x \in \{1,7\}]\ ?$$ ✓

Consequence rule

preserve abstraction

preserve abstraction

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

IL style (reversed Hoare)

we can weaken the pre and
shrink the post, but not too much!
scalable bug detection

Convexity

Lemma. [convexity]
If $\mathbb{C}_P^A$(e) and $P \Rightarrow R \Rightarrow A(P)$ then $\mathbb{C}_R^A$(e)

Proof.
Assume $A([\![e]\!]P) = A([\![e]\!]A(P))$
we want to prove $A([\![e]\!]R) = A([\![e]\!]A(R))$

$$A([\![e]\!]P) \leq A([\![e]\!]R) \leq A([\![e]\!]A(R)) = A([\![e]\!]A(P)) = A([\![e]\!]P)$$

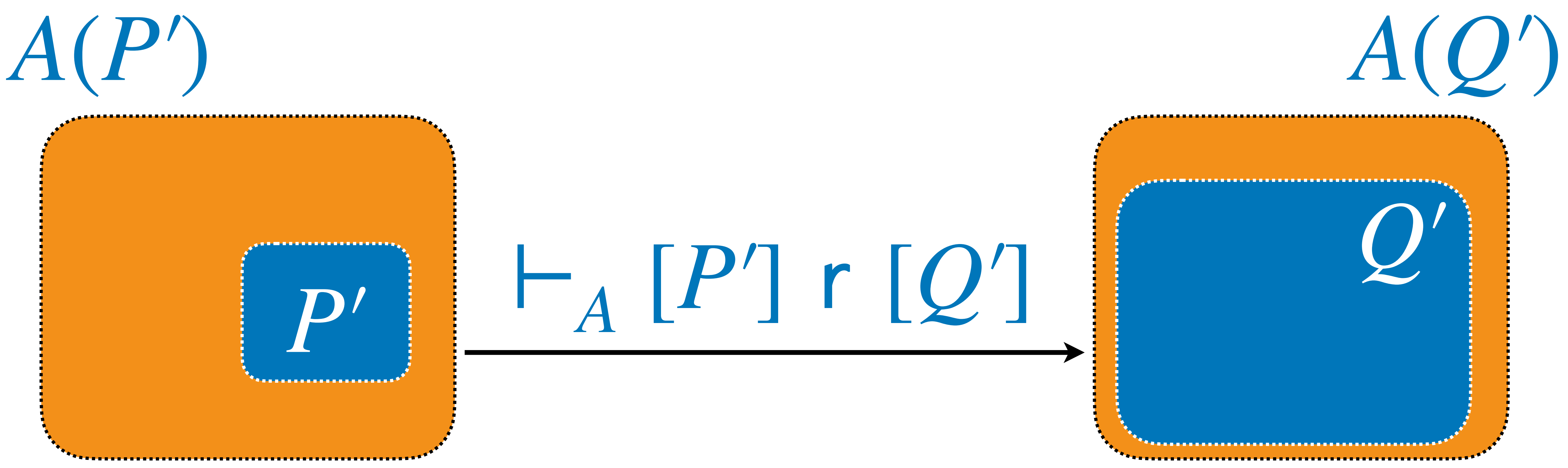
Consequence rule

preserve abstraction

preserve abstraction

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

IL style
(reversed Hoare)



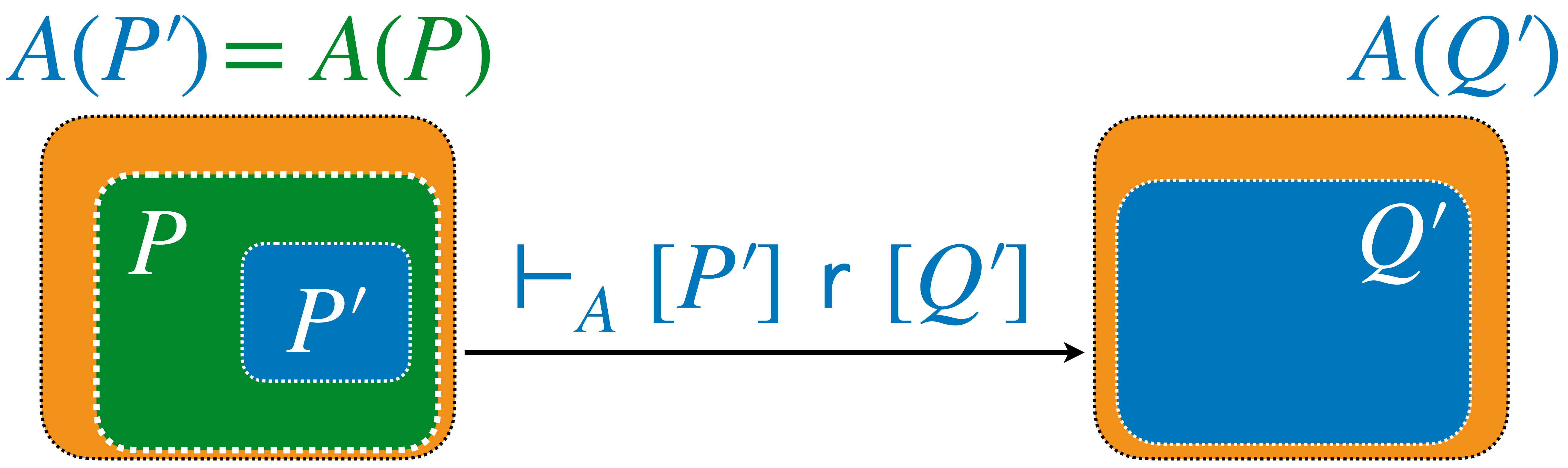
Consequence rule

preserve abstraction

preserve abstraction

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

IL style
(reversed Hoare)



Consequence rule

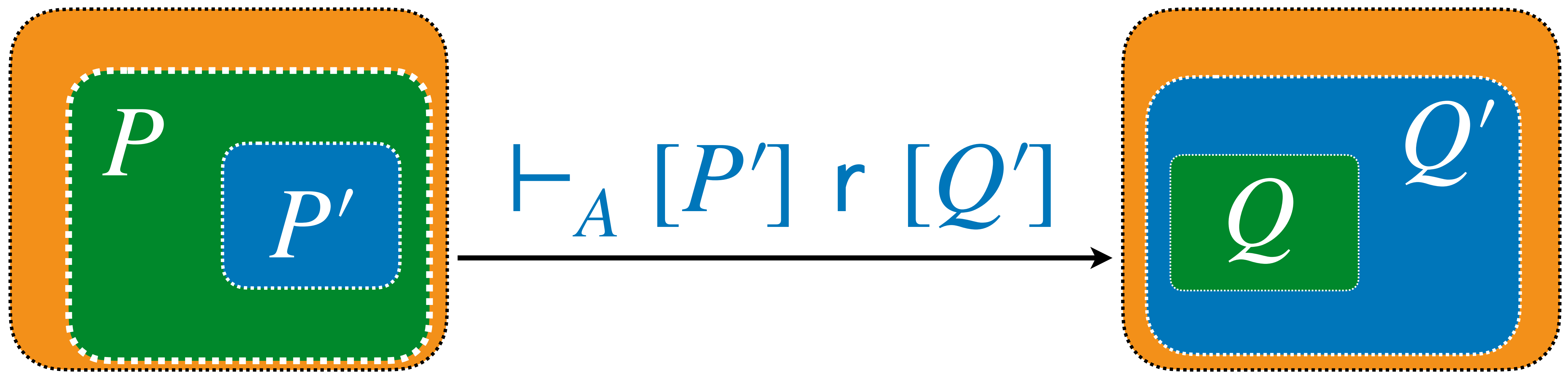
preserve abstraction

preserve abstraction

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

IL style (reversed Hoare)

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



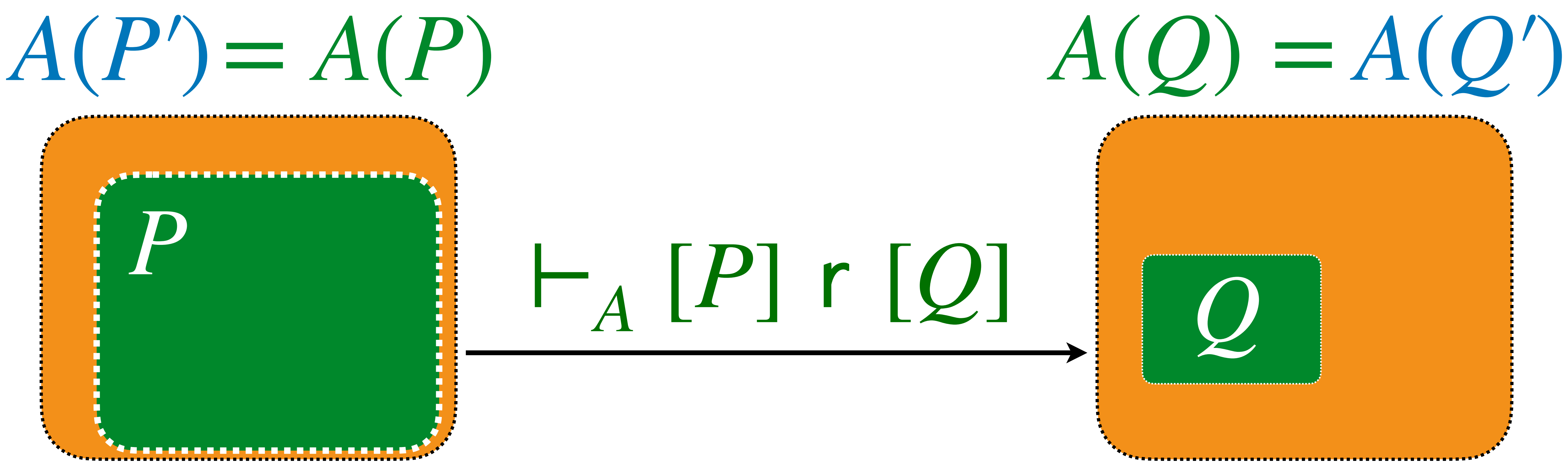
Consequence rule

preserve abstraction

preserve abstraction

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

IL style (reversed Hoare)



Consequence rule

preserve abstraction

preserve abstraction

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

IL style (reversed Hoare)

$$\frac{\vdash_{\text{Int}} [x \in \{-7,0,7\}]\, r\, [x \in \{-5,-2,8\}]}{\vdash_{\text{Int}} [x \in \{-7,0,3,7\}]\, r\, [x \in \{-5,8\}]}$$

Consequence rule

preserve abstraction

preserve abstraction

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

IL style
(reversed Hoare)

$$\frac{\vdash_{\text{Int}} [x \in \{-7,0,7\}]\ r\ [x \in \{-5,-2,8\}]}{\vdash_{\text{Int}} [x \in \{-7,0,7,9\}]\ r\ [x \in \{-2,8\}]}$$

✗

Fixpoint acceleration

one-step unroll

abstract fixpoint!

$$\frac{\vdash_A [P]\ r\ [R] \qquad \vdash_A [P \vee R]\ r^\star\ [Q]}{\vdash_A [P]\ r^\star\ [Q]}\ \text{[rec]}$$

$$\frac{\vdash_A [P]\ r\ [Q] \qquad Q \Rightarrow A(P)}{\vdash_A [P]\ r^\star\ [P \vee Q]}\ \text{[iterate]}$$

locally complete under-approximation!

scalable bug detection

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

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

Sequential composition

$$\frac{\vdash_A [P]\ r_1\ [R] \qquad \vdash_A [R]\ r_2\ [Q]}{\vdash_A [P]\ r_1 ; r_2\ [Q]}\ \text{[seq]}$$

$$\vdash_{\text{Int}} [\text{true}]\ x < 0? ;\ x := -x\ [x \in [1, \infty]]$$

Choice

$$\frac{\vdash_A [P]\ r_1\ [Q_1] \qquad \vdash_A [P]\ r_2\ [Q_2]}{\vdash_A [P]\ r_1 + r_2\ [Q_1 \vee Q_2]}\ \text{[join]}$$

$$\vdash_{\text{Int}} [\text{true}]\ \text{if } x < 0 \text{ then } x := -x \text{ else skip } [x \in [0,\infty]]$$

Validity, soundness, completeness

The rules of LCL

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

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

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

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

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

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

Auxiliary rules

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

$$\frac{\vdash_A [P]\ \mathsf{r}\ [Q] \quad A(P) = A(Q)}{\vdash_A [P]\ \mathsf{r}^*\ [Q]}\ \text{(abs-fix)}$$

$$\frac{\forall n \in \mathbb{N}.\ \vdash_A [P_n]\ \mathsf{r}\ [P_{n+1}]}{\vdash_A [P_0]\ \mathsf{r}^*\ [\bigvee_{i \in \mathbb{N}} P_i]}\ \text{(limit)}$$

Validity

A LCL triple $\vdash_A [P]\ r\ [Q]$ is **valid** if $Q \subseteq [\![r]\!]P \subseteq A(Q)$

Is $\vdash_{\mathsf{Int}} [x > 0]\ x := 10x\ [x \geq 10]$ valid? ❌

Is $\vdash_{\mathsf{Int}} [x > 0, y \geq 0]\ x := yx\ [x \geq 0]$ valid? ❌

Is $\vdash_{\mathsf{Sign}} [x > 0, y > 0]\ x := yx\ [x = 42, y = 7]$ valid? ✅

Is $\vdash_{\mathsf{Sign}} [x > 0]\ (x := x + 1)^\star\ [x > 0]$ valid? ✅

Logical correctness

Th.

If $\vdash_A [P]\ r\ [Q]$ then $Q \subseteq [\![r]\!]P \subseteq A(Q) = [\![r]\!]^{\#}_{A}A(P)$

Proof.

By induction on the derivation.

Verification

Th.

If $A(Spec) = Spec$, then any provable triple $\vdash_A [P]\ r\ [Q]$
either shows the program correct ($Q \subseteq Spec$)
or exposes some true positives ($Q \setminus Spec \neq \emptyset$)

Proof.

$$
\begin{aligned}
[\![r]\!]P \subseteq Spec &\Leftrightarrow A[\![r]\!]P \subseteq Spec \\
&\Leftrightarrow A(Q) \subseteq Spec \\
&\Leftrightarrow Q \subseteq Spec
\end{aligned}
$$

Verification

Th.
If $A(Spec) = Spec$, then any provable triple $\vdash_A [P]\ r\ [Q]$
either shows the program correct ($Q \subseteq Spec$)
or exposes some true positives ($Q \setminus Spec \neq \emptyset$)

correctness + bug finding!

Proof.

$$
\begin{aligned}
[\![r]\!]P \subseteq Spec &\Leftrightarrow A[\![r]\!]P \subseteq Spec \\
&\Leftrightarrow A(Q) \subseteq Spec \\
&\Leftrightarrow Q \subseteq Spec
\end{aligned}
$$

Logical completeness

Th.
If A is complete for any atomic command in r,
then any valid triple $\vdash_A [P]\ r\ [Q]$ can be derived

Proof.
We first derive $\vdash_A [P]\ r\ [[\![r]\!]P]$,
then use [relax] with $Q \subseteq [\![r]\!]P$

Intrinsic logical incompleteness

Th.
For any Turing complete language and any non-trivial abstraction A, there are valid triples that cannot be proved

Proof.
See full version of LICS 2001 paper

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

IL as LCL

Putting pieces together

A IL triple $[P]\ r\ [Q]$ is **valid** if $Q \subseteq [\![r]\!]P$

Th. Any valid IL triple can be derived

A LCL triple $\vdash_A [P]\ r\ [Q]$ is **valid** if $Q \subseteq [\![r]\!]P \subseteq A(Q)$

⇒ $[\![r]\!]P \subseteq A(Q)$ must hold for any $Q \subseteq [\![r]\!]P$ $\quad A = \{ \top \}$

Th. If A is complete for any atomic command in r, then any valid triple $\vdash_A [P]\ r\ [Q]$ can be derived

⇒ A must be complete

Th. For any non-trivial abstraction A, there are valid LCL triples that cannot be proved

⇒ A must be trivial $\quad$ ~~$A = C$~~ $\quad A = \{ \top \}$

Consequences

$A = \{ \top \}$

trivial

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

trivial　　trivial

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

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

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

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

trivial

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

Consequences

$A = \{ \top \}$

$$\frac{}{\vdash_A [P]\ \mathsf{e}\ [\llbracket \mathsf{e} \rrbracket P]}\ (\text{transfer}) \qquad \frac{P' \leq P \qquad \vdash_A [P']\ \mathsf{r}\ [Q'] \qquad Q \leq Q'}{\vdash_A [P]\ \mathsf{r}\ [Q]}\ (\text{relax})$$

$$\frac{\vdash_A [P]\ \mathsf{r}_1\ [R] \qquad \vdash_A [R]\ \mathsf{r}_2\ [Q]}{\vdash_A [P]\ \mathsf{r}_1; \mathsf{r}_2\ [Q]}\ (\text{seq}) \qquad \frac{\vdash_A [P]\ \mathsf{r}_1\ [Q_1] \qquad \vdash_A [P]\ \mathsf{r}_2\ [Q_2]}{\vdash_A [P]\ \mathsf{r}_1 \oplus \mathsf{r}_2\ [Q_1 \vee Q_2]}\ (\text{join})$$

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

How to handle ok and er

$A = \{\ \top\ \}$ $\qquad C \triangleq \wp(\{\mathsf{ok},\mathsf{er}\} \times \Sigma)$

$\epsilon : Q$ shorthand for $\{\epsilon : \sigma \mid \sigma \in Q\} = \{\epsilon\} \times Q$

$$[\![\mathsf{skip}]\!](\mathsf{ok} : Q \cup \mathsf{er} : R) \triangleq \mathsf{ok} : Q \cup \mathsf{er} : R$$

$$[\![x := a]\!](\mathsf{ok} : Q \cup \mathsf{er} : R) \triangleq \mathsf{ok} : \{\sigma[x \mapsto [\![a]\!]\sigma] \mid \sigma \in Q\} \cup \mathsf{er} : R$$

$$[\![\mathsf{error}(\,)]\!](\mathsf{ok} : Q \cup \mathsf{er} : R) \triangleq \mathsf{er} : Q \cup R$$

$$[\![b?]\!](\mathsf{ok} : Q \cup \mathsf{er} : R) \triangleq \mathsf{ok} : (Q \wedge b) \cup \mathsf{er} : R$$

$$[\![x := \mathsf{nondet}(\,)]\!](\mathsf{ok} : Q \cup \mathsf{er} : R) \triangleq \mathsf{ok} : \{\sigma[x \mapsto v] \mid \sigma \in Q, v \in \mathbb{Z}\} \cup \mathsf{er} : R$$

Lemma

error preserving semantics

$$[\![r]\!](\mathsf{ok} : Q \cup \mathsf{er} : R) = \mathsf{er} : R \cup \bigcup_{\sigma \in Q} [\![r]\!](\mathsf{ok} : \sigma)$$

IL as LCL

IL's relational semantics

IL's relational semantics

Lemma.

$$[\![r]\!](\mathsf{ok} : P) = \mathsf{ok} : [\![r]\!]\mathsf{ok}(P) \cup \mathsf{er} : [\![r]\!]\mathsf{er}(P)$$

Corollary. [*IL as an instance of LCL*]

$$[P]\; r\; [\mathsf{ok} : Q][\mathsf{er} : R] \text{ in IL iff } \vdash_{\{\top\}} [\mathsf{ok} : P]\; r\; [\mathsf{ok} : Q \cup \mathsf{er} : R]$$

Questions

Question 1

Which LCL triples are valid for any r and P ?

$\vdash_{\{\top\}} [P]\ r\ [\mathsf{false}]$ ✓

$\vdash_{\{\top\}} [P]\ r\ [\mathsf{true}]$ ✗

$\vdash_{\mathsf{Sign}} [x > 10]\ r\ [\mathsf{false}]$ ✗

$\vdash_{\{\top\}} [wlp(r, P)]\ r\ [P]$ ✗

Question 2

Find a derivation for the IL triple

$\vdash_{\text{Oct}}$ $[x < 10, y > 20]$ if $x \geq y$ then $z := x$ else $z := y$ $[x < 10, y > 20, z = \max(x, y)]$

$[x < 10, y > 20]$

if $x \geq y$ then

$[\text{false}]$

$z := x$

$[\text{false}]$

else

$[x < 10, y > 20]$

$z := y$

$[x < 10, z = y > 20] \equiv [x < 10, y > 20, z = \max(x, y)]$

$[x < 10, y > 20, z = \max(x, y)]$

Question 3

Are these “mixed” HL+LCL inference rules valid ?

$$\frac{\vdash_A [P]\ r\ [Q]}{\{P\}\ r\ \{A(Q)\}} \qquad \frac{\vdash_A [P]\ r\ [Q]}{\{A(P)\}\ r\ \{A(Q)\}}$$

If $\vdash_A [P]\ r\ [Q]$ then $[\![r]\!]P \subseteq A(Q)$, hence $\{P\}\ r\ \{A(Q)\}$ is valid

If $\vdash_A [P]\ r\ [Q]$ then $[\![r]\!]A(P) \subseteq [\![r]\!]^{\#}A(P) = A(Q)$, hence $\{A(P)\}\ r\ \{A(Q)\}$ is valid

* Exam 8

Prove that [conj] is unsound for LCL

$$\frac{\vdash_A [P_1]\ r\ [Q_1] \quad \vdash_A [P_2]\ r\ [Q_2]}{\vdash_A [P_1 \wedge P_2]\ r\ [Q_1 \wedge Q_2]} \text{ [conj]}$$

* Exam 9

Show that the following rule is not sound

$$\frac{}{\vdash_A [P]\ x := \text{nondet}()\ [P[v/x]]}$$

arbitrary value