Program analysis: from proving correctness to proving incorrectness

Roberto Bruni, Roberta Gori
(University of Pisa)

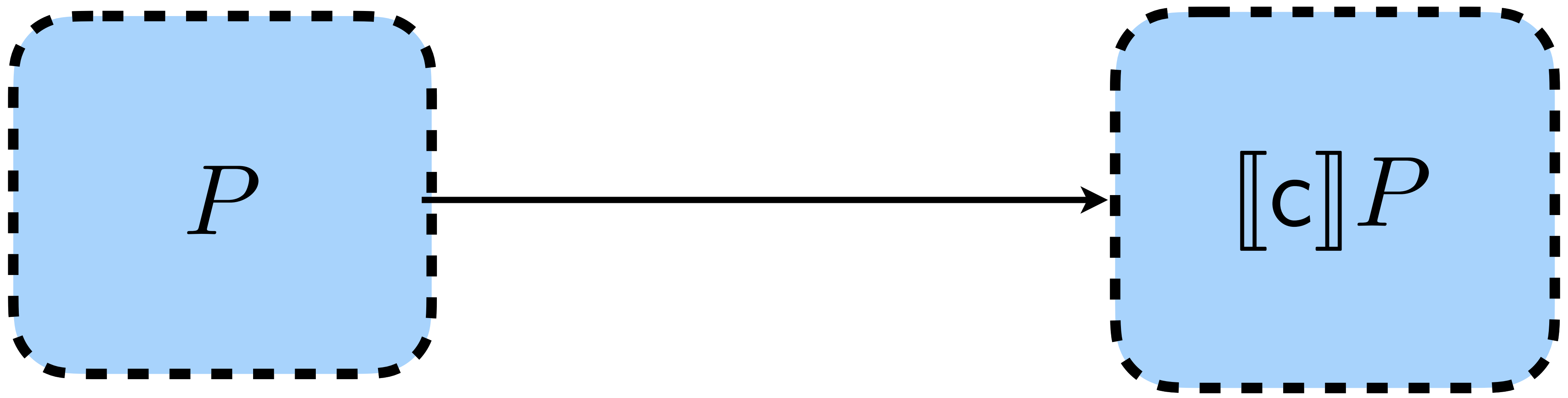
Lecture #07

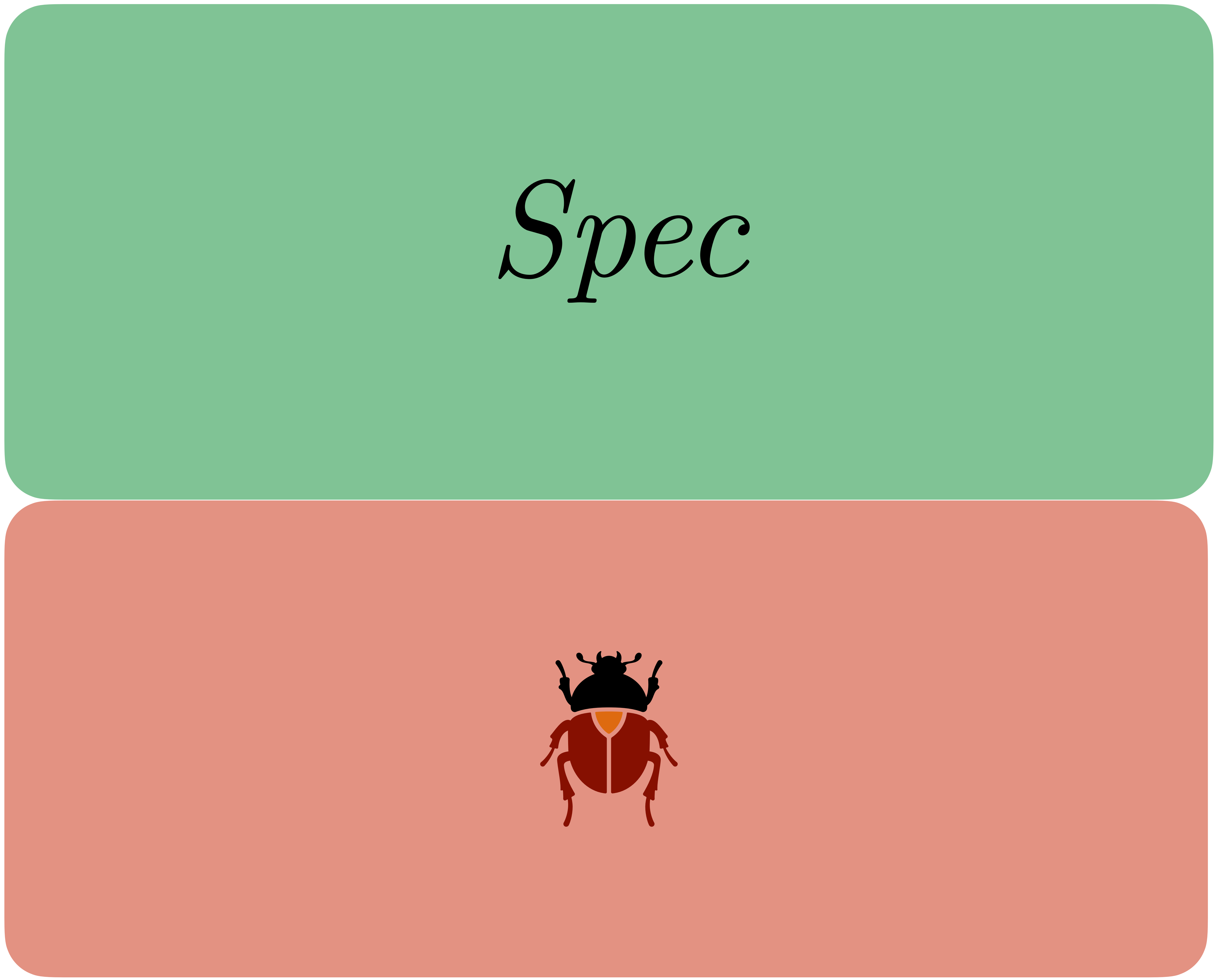
BISS 2024
March 11-15, 2024

Recap:
combining over and under

Verification problem

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





Over vs Under

HL

$$P \xrightarrow[\text{logically complete}]{\{P\}\ \mathsf{c}\ \{Q\}} Q$$

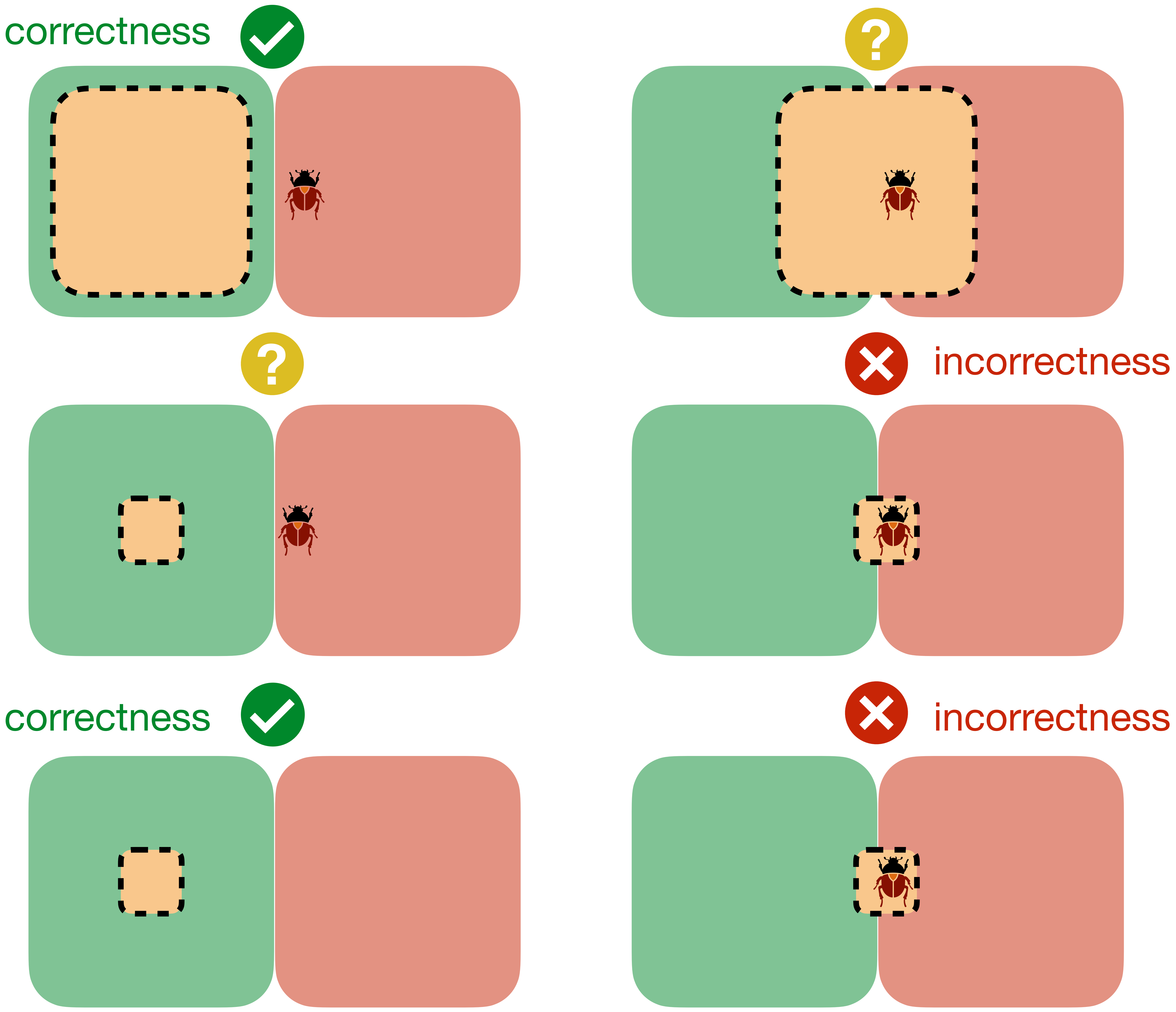
IL

$$P \xrightarrow[\text{logically complete}]{[P]\ \mathsf{c}\ [Q]} Q$$

LCL

$$P \xrightarrow[A(Spec) = Spec]{\vdash_A [P]\ c\ [Q]} Q$$

logically incomplete



What can go wrong?

local completeness requirement

local completeness requirement

local completeness requirement

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

local-completeness proof obligations can fail!

any non-trivial abstract domain A introduces some imprecision!

J.ACM 70(2)

A Correctness and Incorrectness Program Logic

ROBERTO BRUNI, University of Pisa, Italy
ROBERTO GIACOBAZZI, University of Verona, Italy
ROBERTA GORI, University of Pisa, Italy
FRANCESCO RANZATO, University of Padova, Italy

Abstract interpretation is a well-known and extensively used method to extract over-approximate program invariants by a sound program analysis algorithm. Soundness means that no program errors are lost and it is, in principle, guaranteed by construction. Completeness means that the abstract interpreter reports no false alarms for all possible inputs, but this is extremely rare because it needs a very precise analysis. We introduce a weaker notion of completeness, called *local completeness*, which requires that no false alarms are produced only relatively to some fixed program inputs. Based on this idea, we introduce a program logic, called Local Completeness Logic for an abstract domain A, for proving both the correctness and incorrectness of program specifications. Our proof system, which is parameterized by an abstract domain A, combines over- and under-approximating reasoning. In a provable triple $\vdash_A [p]\ \mathsf{c}\ [q]$, c is a program, q is an under-approximation of the strongest post-condition of c on input p such that their abstractions in A coincide. This means that q is never too coarse, namely, under some mild assumptions, *the abstract interpretation of* c *does not yield false alarms for the input* p *iff* q *has no alarm.* Therefore, proving $\vdash_A [p]\ \mathsf{c}\ [q]$ not only ensures that all the alarms raised in q are true ones, but also that if q does not raise alarms, then c is correct. We also prove that if A is the straightforward abstraction making all program properties equivalent, then our program logic coincides with O'Hearn's incorrectness logic, while for any other abstraction, contrary to the case of incorrectness logic, our logic can also establish program correctness.

CCS Concepts: • **Theory of computation → Logic and verification**; **Abstraction**; **Programming logic**; **Semantics and reasoning**; **Program analysis**; *Hoare logic*; Axiomatic semantics; **Abstraction**; **Program reasoning;**

Additional Key Words and Phrases: Abstract interpretation, abstract domain, program analysis, program verification, program logic, local completeness, best correct approximation, incorrectness logic

ACM Reference format:
Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2023. A Correctness and Incorrectness Program Logic. *J. ACM* 70, 2, Article 15 (March 2023), 45 pages.
https://doi.org/10.1145/3582267

The authors have been funded by the *Italian MIUR*, under the PRIN2017 project no. 201784YSZ5 "AnalysiS of PRogram Analyses (ASPRA)" and by a *Meta research* gift. Roberto Giacobazzi and Francesco Ranzato have been partially funded by *Facebook Research*, under a "Probability and Programming Research Award," by an *Amazon Research Award* for "AWS Automated Reasoning," and by a *WhatsApp Research Award* on "Privacy-aware Program Analysis."

Authors' addresses: R. Bruni and R. Gori, University of Pisa, Pisa, Italy; emails: bruni@di.unipi.it, gori@di.unipi.it; R. Giacobazzi, University of Verona, Verona, Italy; email: roberto.giacobazzi@univr.it; F. Ranzato, University of Padova, Padova, Italy; email: ranzato@math.unipd.it.

0004-5411/2023/03-ART15 $15.00
https://doi.org/10.1145/3582267

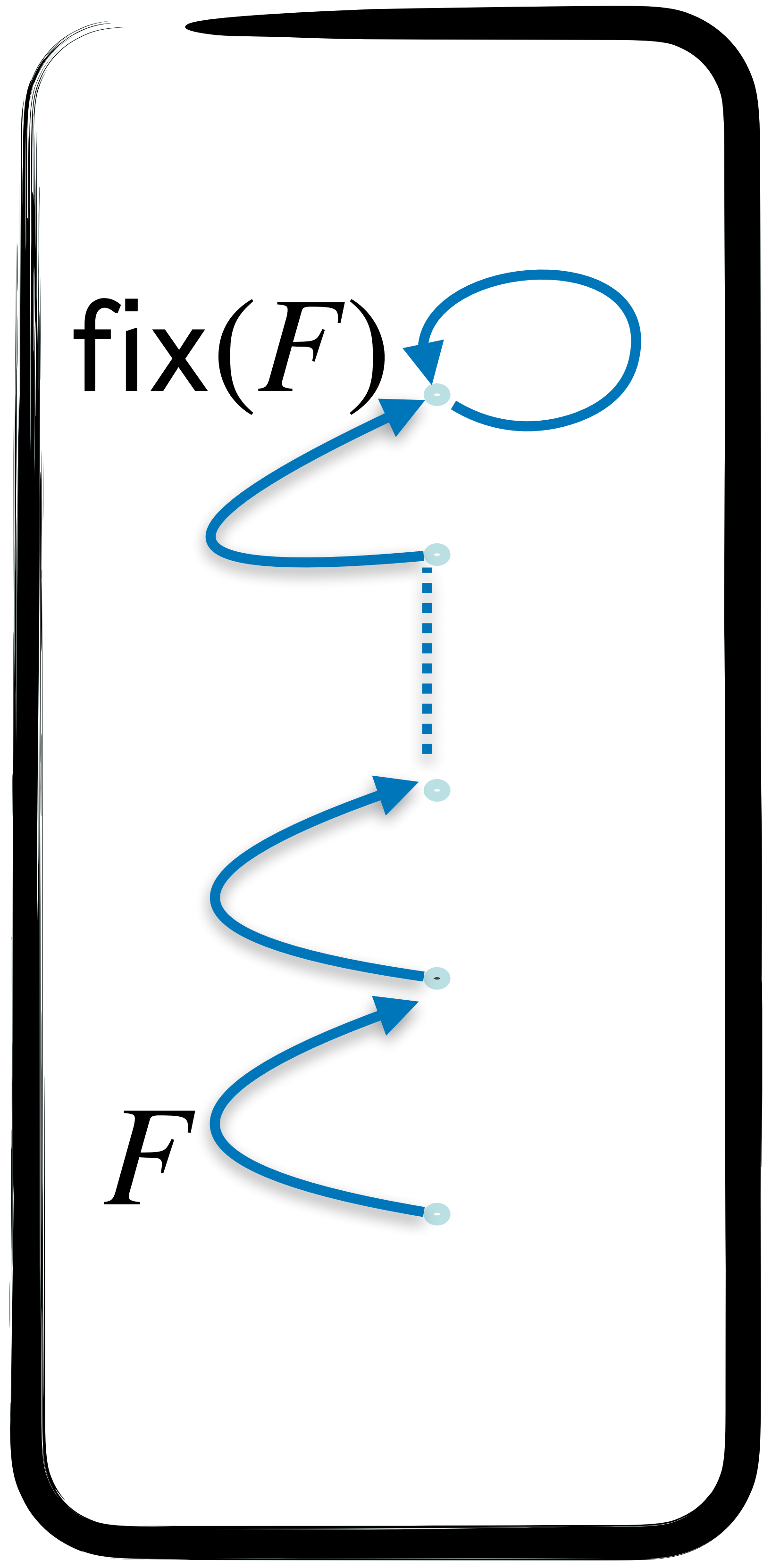
15

we show how to relax local-completeness requirements for while loops and by domain refinement

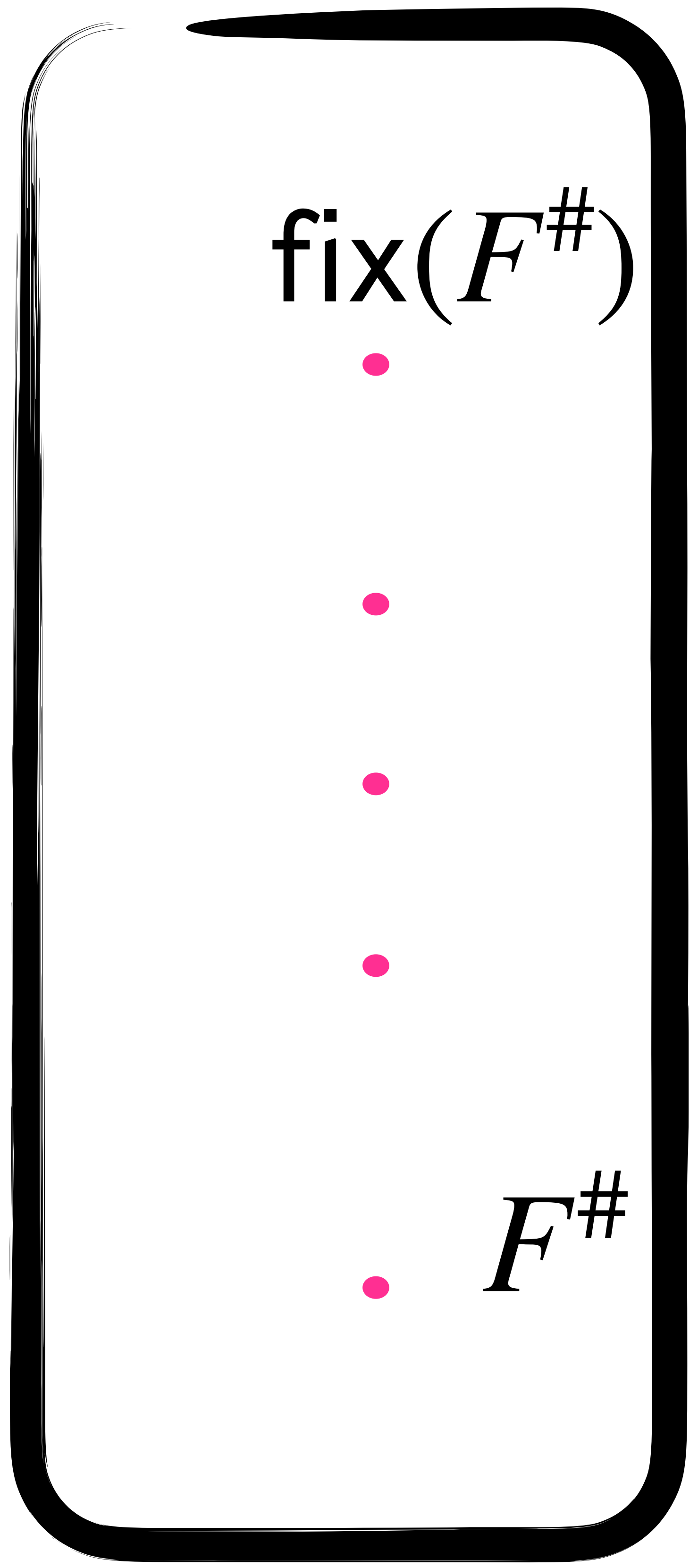
While loops

Fixpoints preserve completeness

$(C, \subseteq)$

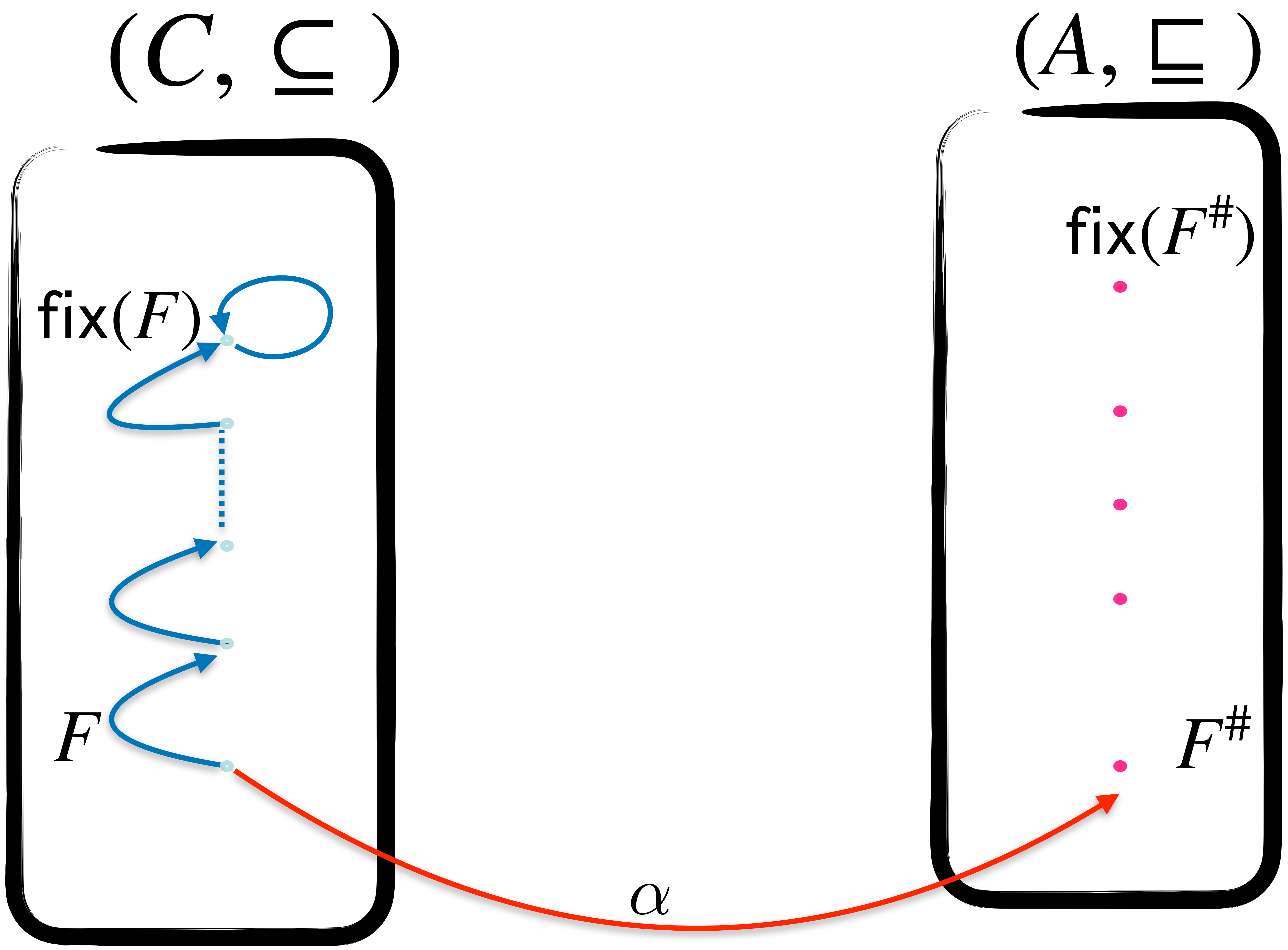


$(A, \sqsubseteq)$



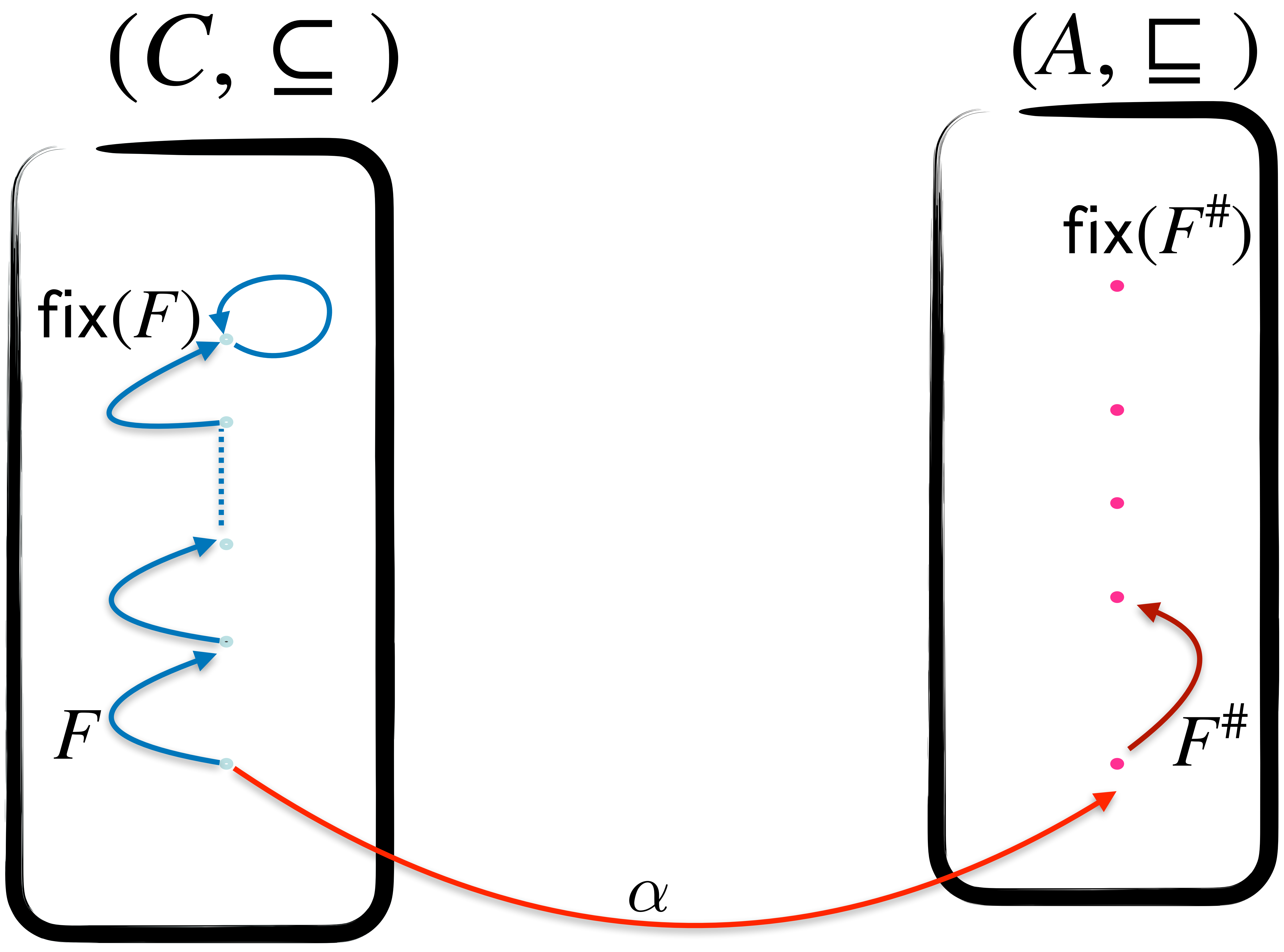
if $F^\#$ is complete, then $\text{fix}(F^\#) = \alpha(\text{fix}(F))$

Fixpoints preserve completeness



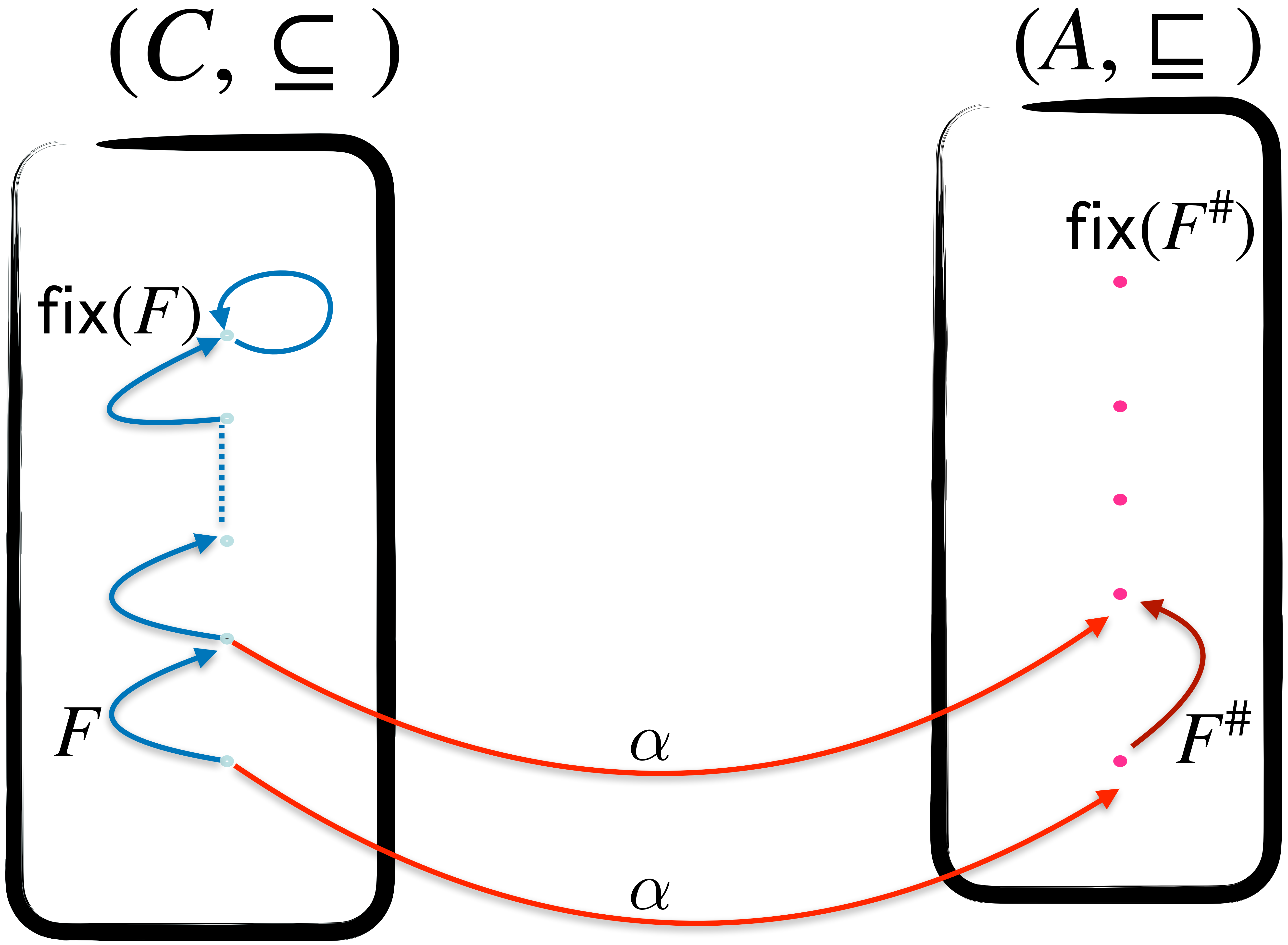
if $F^\#$ is complete, then fix$(F^\#) = \alpha(\text{fix}(F))$

Fixpoints preserve completeness



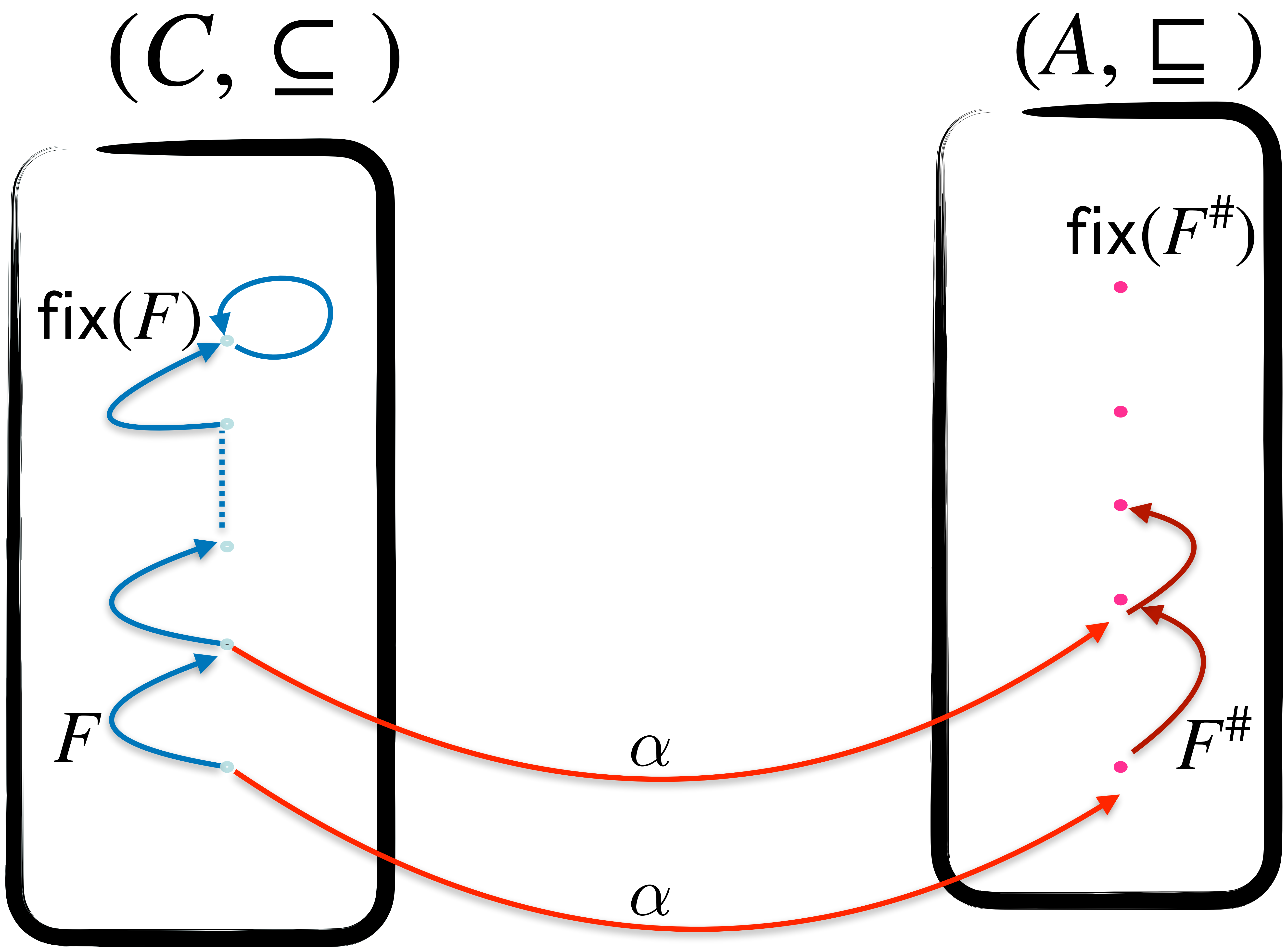
if $F^{\#}$ is complete, then $\mathsf{fix}(F^{\#}) = \alpha(\mathsf{fix}(F))$

Fixpoints preserve completeness



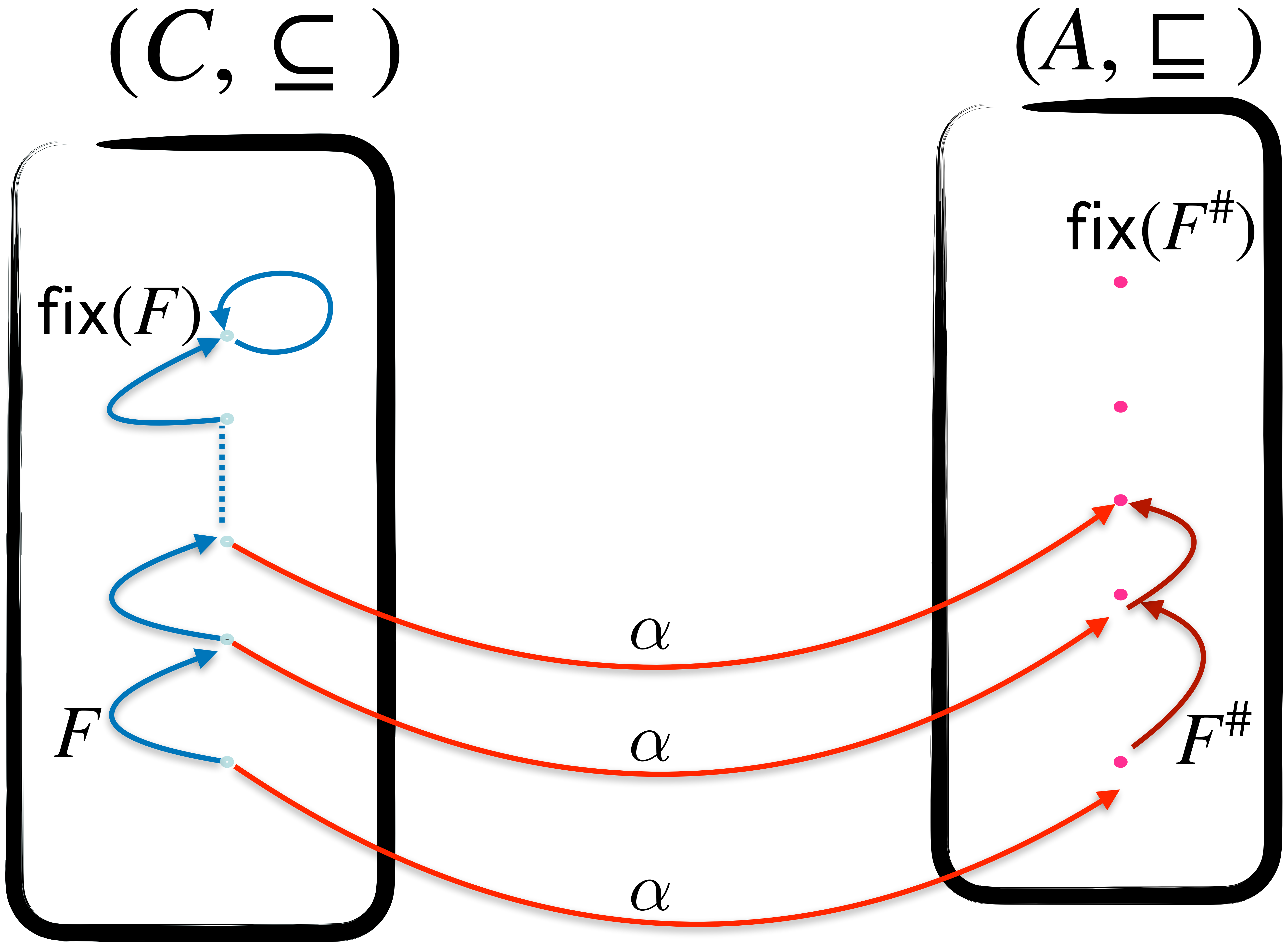
if $F^\#$ is complete, then $\mathsf{fix}(F^\#) = \alpha(\mathsf{fix}(F))$

Fixpoints preserve completeness



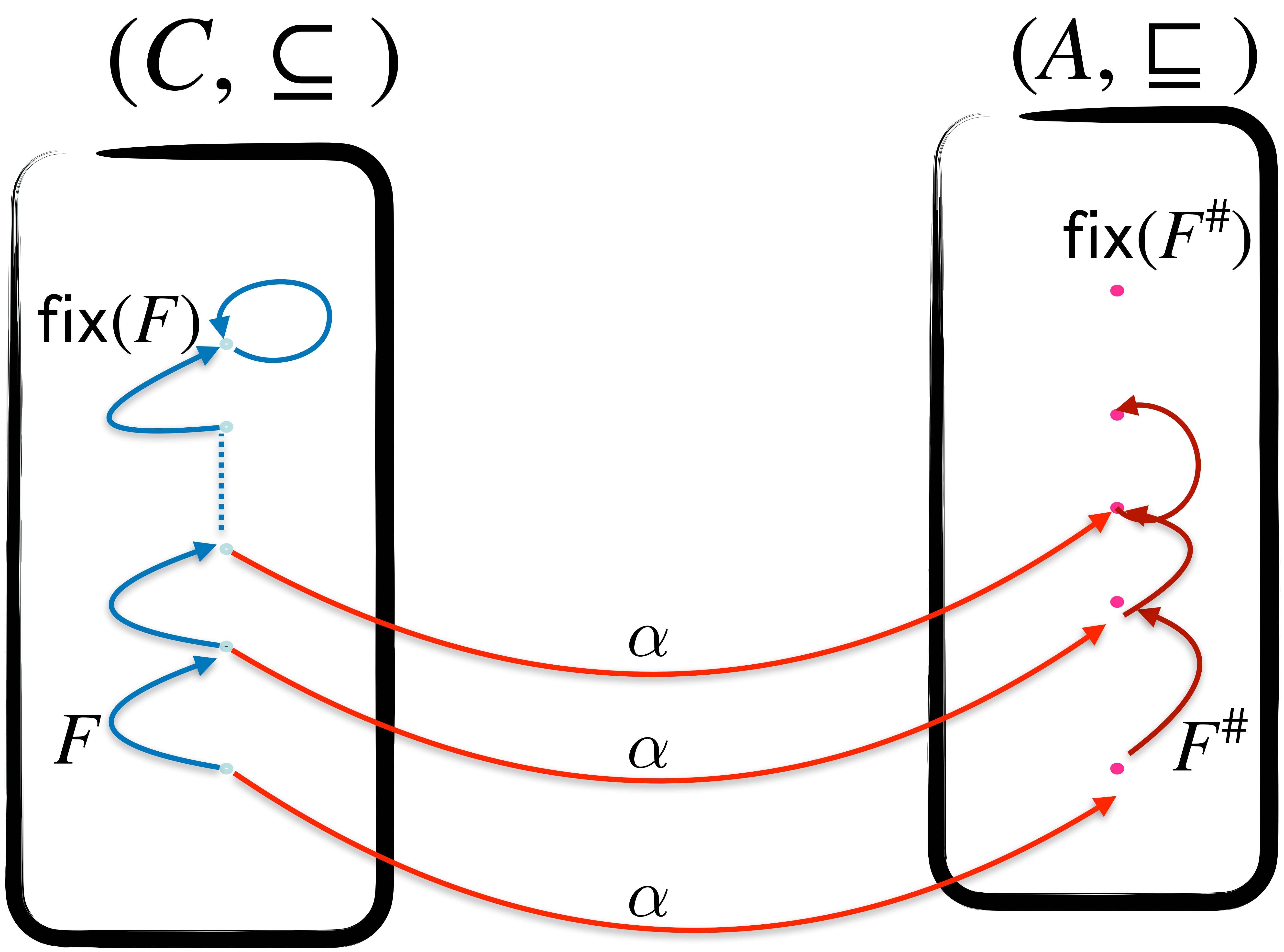
if $F^\#$ is complete, then $\mathsf{fix}(F^\#) = \alpha(\mathsf{fix}(F))$

Fixpoints preserve completeness



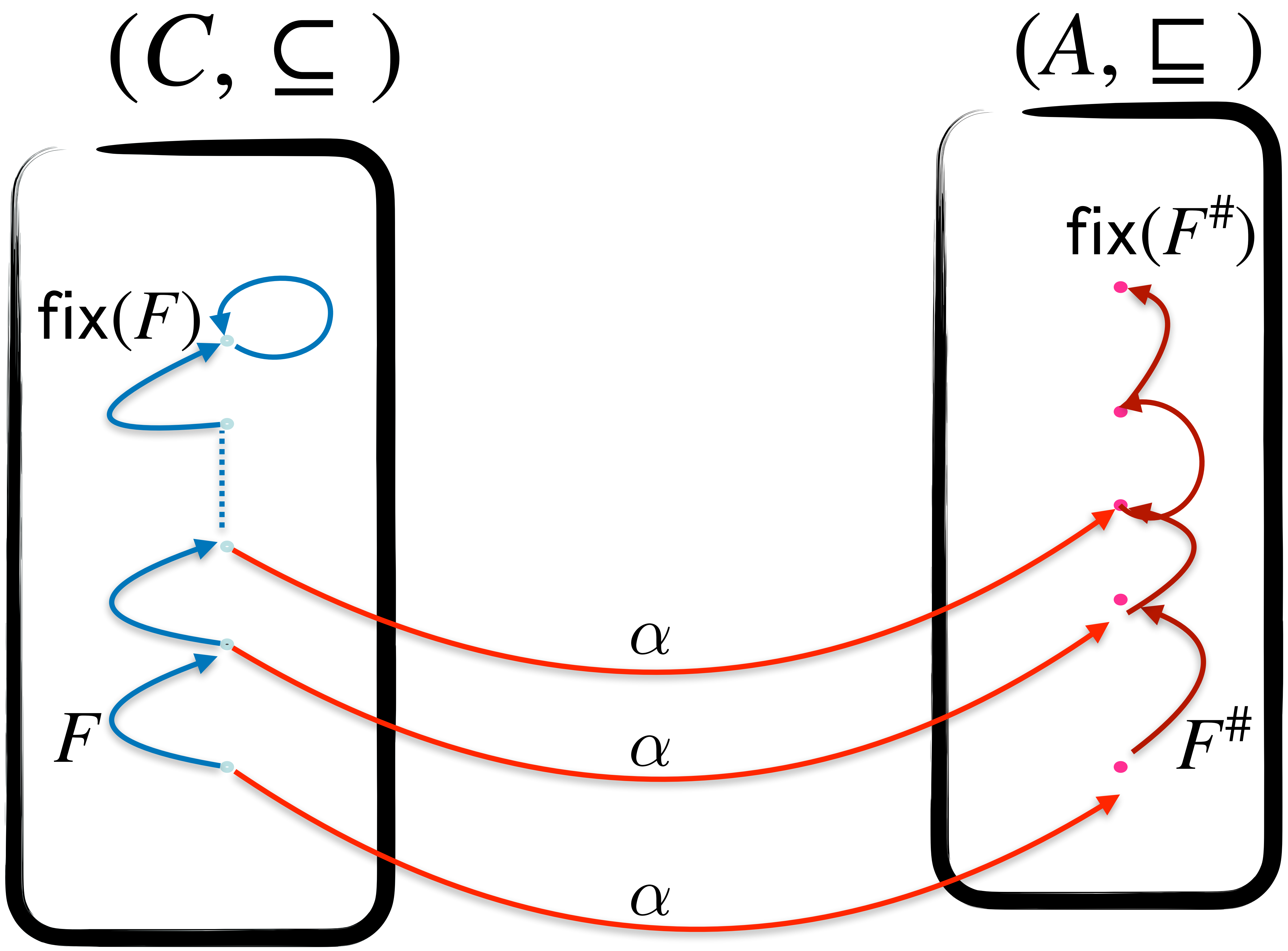
if $F^\#$ is complete, then $\mathsf{fix}(F^\#) = \alpha(\mathsf{fix}(F))$

Fixpoints preserve completeness



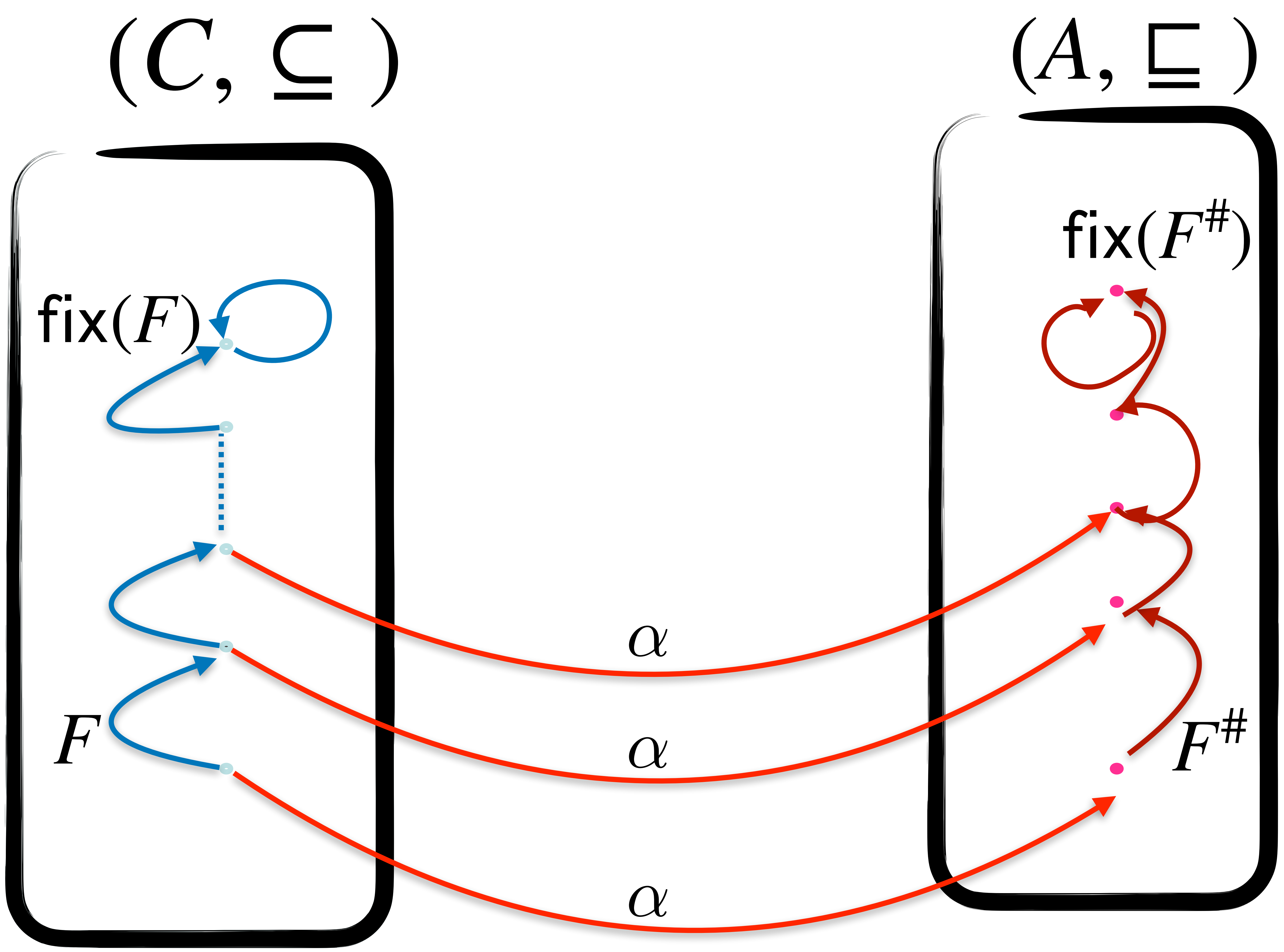
if $F^\#$ is complete, then fix$(F^\#) = \alpha($fix$(F))$

Fixpoints preserve completeness



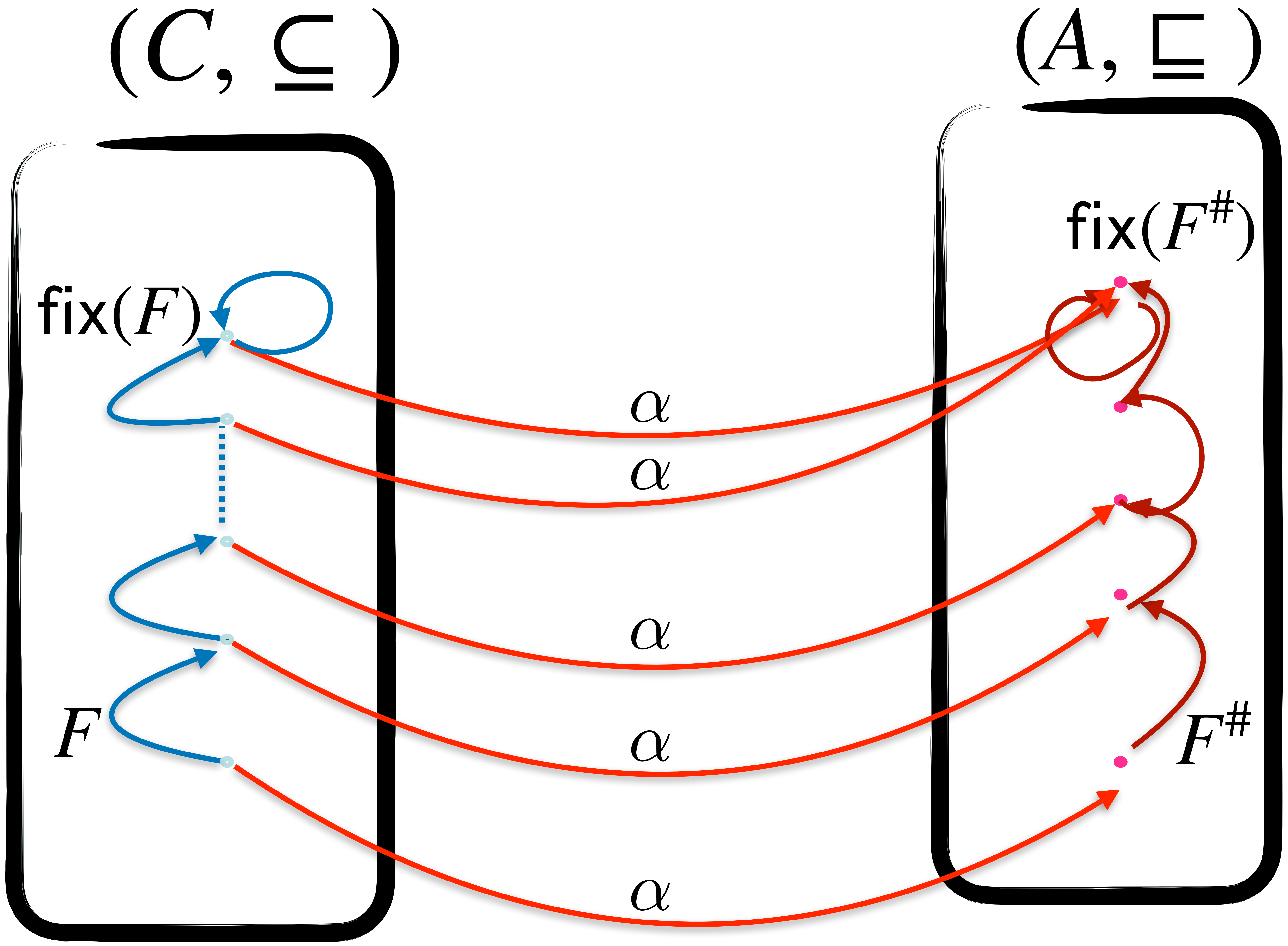
if $F^{\#}$ is complete, then $\mathsf{fix}(F^{\#}) = \alpha(\mathsf{fix}(F))$

Fixpoints preserve completeness



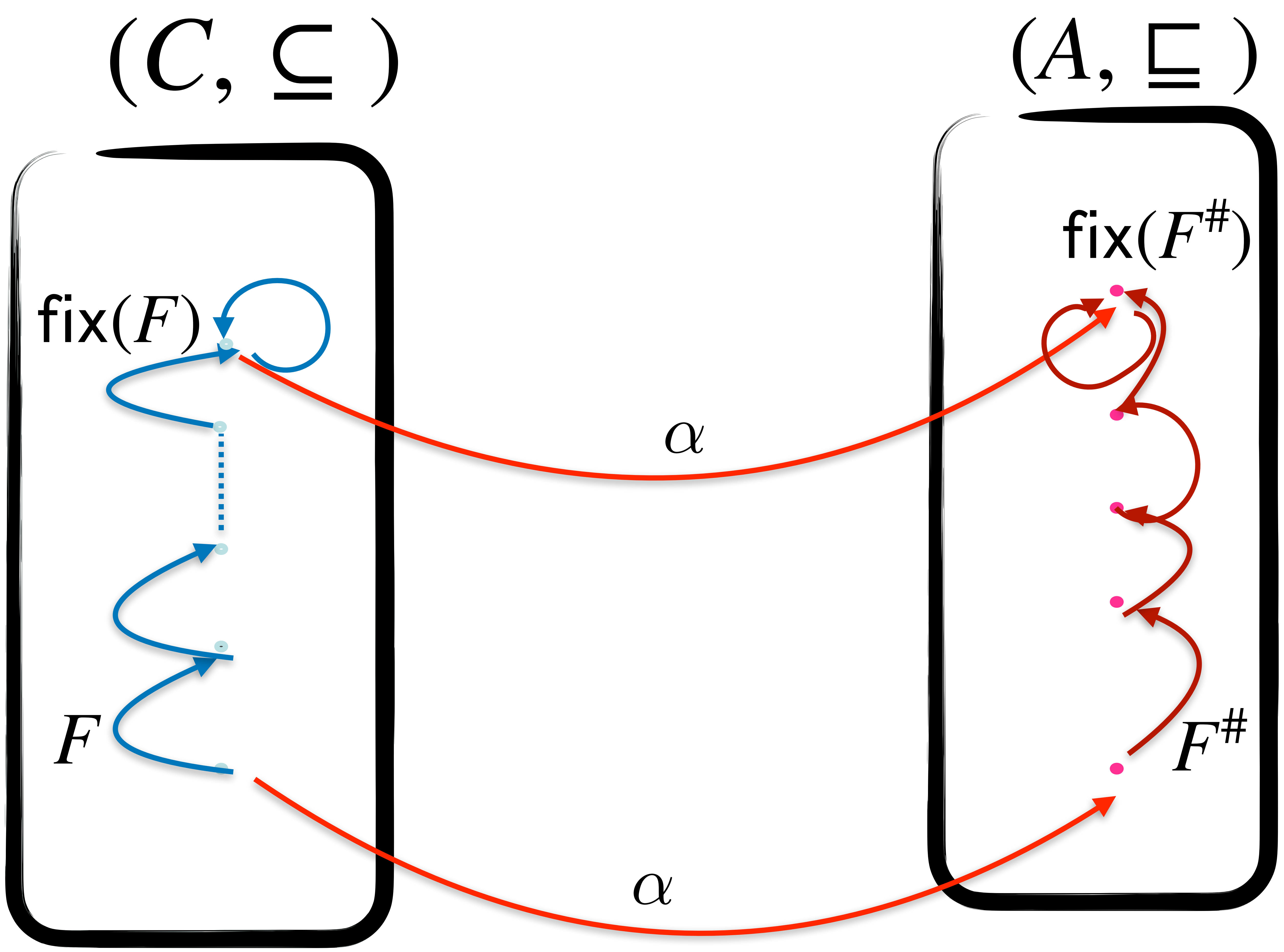
if $F^\#$ is complete, then $\mathsf{fix}(F^\#) = \alpha(\mathsf{fix}(F))$

Fixpoints preserve completeness



if $F^{\#}$ is complete, then $\mathsf{fix}(F^{\#}) = \alpha(\mathsf{fix}(F))$

Not a necessary requirement



we can have $\text{fix}(F^\#) = \alpha(\text{fix}(F))$ when $F^\#$ is just locally complete on $\text{fix}(F)$

Finite unrolling of while loops

$$\text{while } b \text{ do } c \triangleq (b?;c)^\star;\neg b?$$

$$r \triangleq b?;c$$

requires local completeness for $b?;c$

abstract fixpoint!

requires local completeness for $b?;c$

requires local completeness for $b?;c$

local completeness for $\neg b$

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash_A [P]\ r\ [R_1]} \quad \dfrac{\dfrac{\vdots}{\vdash_A [P \lor R_1]\ r\ [R_2]} \quad \dfrac{\dfrac{\vdash_A [P \lor R_1 \lor \ldots \lor R_k]\ r\ [Q] \quad Q \Rightarrow A(P \lor \ldots)}{\vdots}}{\vdash_A [P \lor \ldots]\ r^\star\ [Q]}}{\vdash_A [P \lor R_1]\ r^\star\ [Q]}}{\vdash_A [P]\ (b?;c)^\star\ [Q]} \quad \dfrac{\mathbb{C}^A_Q(\neg b)}{\vdash_A [Q]\ \neg b?\ [Q \land \neg b]}}{\vdash_A [P]\ \text{while } b \text{ do } c\ [Q \land \neg b]}$$

Example

$$r \triangleq x > 0?\,;\; x := x - 2$$

fails!

$\mathsf{Int}[\![x > 0]\!]\mathsf{Int}\{-3,0,3\} = \mathsf{Int}[\![x > 0]\!][-3,3] = \mathsf{Int}[1,3] = [1,3]$

$\mathsf{Int}[\![x > 0]\!]\{-3,0,3\} = \mathsf{Int}\{3\} = [3,3]$

$$\dfrac{\dfrac{\dfrac{\mathbb{C}^{\mathsf{Int}}_{\{-3,0,3\}}(x > 0)}{\vdash_{\mathsf{Int}} [\{-3,0,3\}]\; x > 0?\; [W_1]} \quad \dfrac{\vdots}{\vdash_{\mathsf{Int}} [W_1]\; \ldots\; [R_1]}}{\vdash_{\mathsf{Int}} [\{-3,0,3\}]\; r\; [R_1]} \quad \dfrac{\vdots}{\vdash_{\mathsf{Int}} [P \vee R_1]\; r^\star\; [Q]}}{\vdash_{\mathsf{Int}} [\{-3,0,3\}]\; r^\star\; [Q]} \quad \dfrac{\vdots}{\vdash_{\mathsf{Int}} [Q]\; x \le 0?\; [Q \wedge x \le 0]}$$

$$\vdash_{\mathsf{Int}} [\{-3,0,3\}]\; \text{while } x > 0 \text{ do } x := x - 2\; [Q \wedge x \le 0]$$

Locally complete invariants

local completeness for test b not required!

$$\frac{\vdash_A [P \wedge b]\ c\ [R] \qquad \vdash_A [P \vee R]\ \text{while } b \text{ do } c\ [Q]}{\vdash_A [P]\ \text{while } b \text{ do } c\ [Q]}$$

local completeness for test b

abstract fixpoint!

$$\frac{\mathbb{C}^A_P(b) \qquad \mathbb{C}^A_P(\neg b) \qquad [P \wedge b]\ c\ [Q] \qquad Q \Rightarrow A(P)}{[P]\ \text{while } b \text{ do } c\ [(P \vee Q) \wedge \neg b]}$$

Finite unrolling of while loops

local-completeness proof obligations for guards are necessary just when the abstract fixpoint is reached!

local completeness for test b

abstract fixpoint!

local completeness for test b not required!

local completeness for test b not required!

$$\dfrac{\mathbb{C}^{A}_{P\vee\ldots}(b,\neg b) \qquad \vdash_A [(P\vee\ldots)\wedge b]\ c\ [Q] \qquad Q \Rightarrow A(P\vee\ldots)}{\vdots}$$

$$\dfrac{\dfrac{\vdots}{\vdash_A [P \wedge b]\ c\ [R_1]} \qquad \dfrac{\dfrac{\vdots}{\vdash_A [(P\vee R_1)\wedge b]\ c\ [R_2]} \qquad \dfrac{\vdots}{\vdash_A [P\vee\ldots]\ \text{while } b \text{ do } c\ [(P\vee\ldots\vee Q)\wedge\neg b]}}{\vdash_A [P\vee R_1]\ \text{while } b \text{ do } c\ [(P\vee\ldots\vee Q)\wedge\neg b]}}{\vdash_A [P]\ \text{while } b \text{ do } c\ [(P\vee\ldots\vee Q)\wedge\neg b]}$$

Example

succeeds!

succeed!

succeeds!

abstract fixpoint!

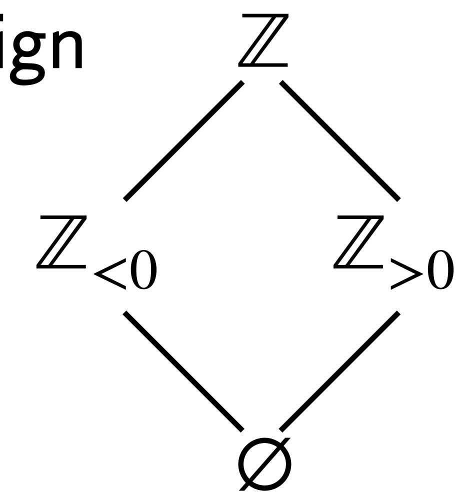
$$\dfrac{\dfrac{\mathbb{C}^{\mathsf{Int}}_{\{3\}}(x-2)}{\vdash_{\mathsf{Int}} [\{3\}]\ x := x-2\ [\{1\}]} \qquad \dfrac{\mathbb{C}^{\mathsf{Int}}_{\{\dots\}}(x \le 0) \qquad \dfrac{\mathbb{C}^{\mathsf{Int}}_{\{1,3\}}(x-2)}{\mathbb{C}^{\mathsf{Int}}_{\{\dots\}}(x > 0) \vdash_{\mathsf{Int}} [\{1,3\}]\ x := x-2\ [\{-1,1\}]} \qquad \{-1,1\} \subseteq [-3,3]}{\vdash_{\mathsf{Int}} [\{-3,0,1,3\}]\ \text{while } x > 0 \text{ do } x := x-2\ [\{-3,-1,0\}]}}{\vdash_{\mathsf{Int}} [\{-3,0,3\}]\ \text{while } x > 0 \text{ do } x := x-2\ [\{-3,-1,0\}]}$$

Refinement

Domain refinement

to satisfy a local completeness requirement, it can be useful to refine the domain

Sign



$$\mathsf{Sign}[\![x \neq 0]\!]\mathsf{Sign}\{0,1\} = \mathsf{Sign}[\![x \neq 0]\!]\mathbb{Z} = \mathsf{Sign}(x \neq 0) = \mathbb{Z}$$

$$\mathsf{Sign}[\![x \neq 0]\!]\{0,1\} = \mathsf{Sign}\{1\} = \mathbb{Z}_{>0}$$

succeed!

fails!

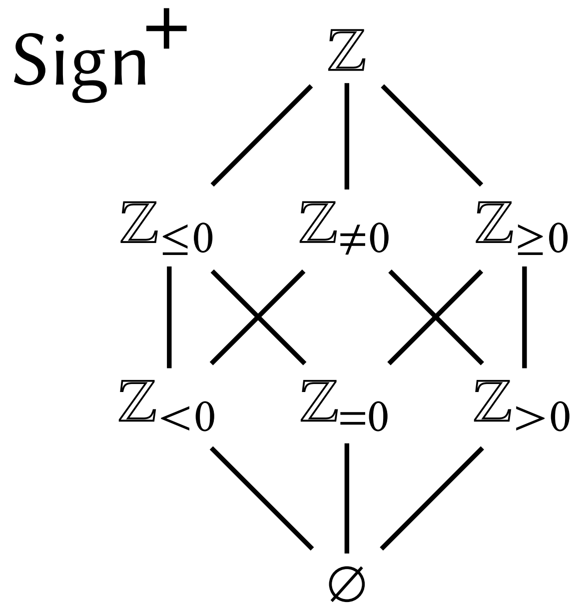
succeed!

abstract fixpoint!

$$\dfrac{\mathbb{C}^{\mathsf{Sign}}_{\{0,1\}}(x \neq 0) \qquad \mathbb{C}^{\mathsf{Sign}}_{\{0,1\}}(x = 0) \qquad \dfrac{\mathbb{C}^{\mathsf{Sign}}_{\{1\}}(x+1)}{\vdash_{\mathsf{Sign}} [\{1\}]\ x := x+1\ [\{2\}]} \qquad \{2\} \subseteq \mathsf{Sign}(\{0,1\}) = \mathbb{Z}}{\vdash_{\mathsf{Sign}} [\{0,1\}]\ \text{while } x \neq 0 \text{ do } x := x+1\ [\{0\}]}$$

Domain refinement

to satisfy a local completeness requirement, it can be useful to refine the domain



$$\mathsf{Sign}^+[\![x \neq 0]\!]\mathsf{Sign}^+\{0,1\} = \mathsf{Sign}^+[\![x \neq 0]\!]\mathbb{Z}_{\geq 0} = \mathsf{Sign}^+\mathbb{Z}_{>0} = \mathbb{Z}_{>0}$$

$$\mathsf{Sign}^+[\![x \neq 0]\!]\{0,1\} = \mathsf{Sign}^+\{1\} = \mathbb{Z}_{>0}$$

succeed!

succeed!

succeed!

abstract fixpoint!

$$\dfrac{\mathbb{C}^{\mathsf{Sign}^+}_{\{0,1\}}(x \neq 0) \quad \mathbb{C}^{\mathsf{Sign}^+}_{\{0,1\}}(x = 0) \quad \dfrac{\mathbb{C}^{\mathsf{Sign}^+}_{\{1\}}(x+1)}{\vdash_{\mathsf{Sign}^+} [\{1\}]\ x := x+1\ [\{2\}]} \quad \{2\} \subseteq \mathsf{Sign}^+(\{0,1\}) = \mathbb{Z}_{\geq 0}}{\vdash_{\mathsf{Sign}^+} [\{0,1\}]\ \text{while } x \neq 0 \text{ do } x := x+1\ [\{0\}]}$$

Domain integration

suppose $\vdash_{A_1} [P]$ r_1 $[R]$ and $\vdash_{A_2} [R]$ r_2 $[Q]$:

can we conclude $\vdash_{A} [P]$ r_1 ; r_2 $[Q]$ for some suitable A ?

$A = A_1 \sqcap A_2$?

not guaranteed to work (some proof obligations may fail)

Conjunctive properties

program verification often requires the use of the conjunction of several basic predicates

concrete states = stores with two variables x, y

intervals abstraction for each variable

abstract state = an interval for each variable

[0,∞] [3,8]

Product domain

$$C \underset{\alpha_0}{\overset{\gamma_0}{\leftrightarrows}} A_0 \qquad\qquad C \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} A_1$$

$$C \underset{\alpha_\times}{\overset{\gamma_\times}{\leftrightarrows}} A_0 \times A_1$$

$$\gamma_\times(a_0, a_1) = \gamma_0(a_0) \cap \gamma_1(a_1)$$

Problem

concrete stores = stores with one variable x

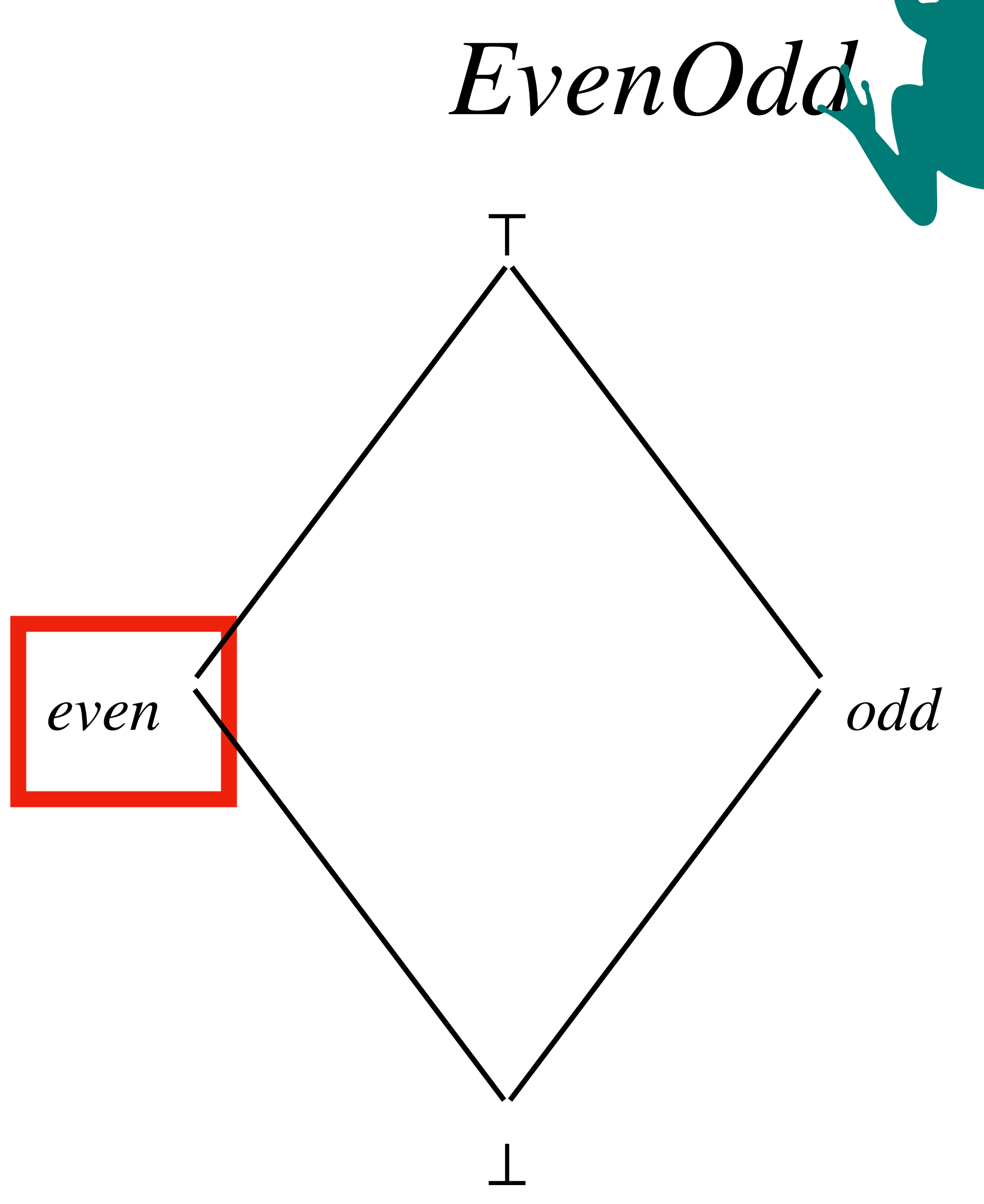
Int × EvenOdd

e.g. an abstract state $(\,[2{,}10]\,,\mathit{even}\,)$

describes **even** values between 2 and 10

but also $(\,[1{,}11]\,,\mathit{even}\,)$ represents the same

concrete set $\{2{,}4{,}6{,}8{,}10\}$!



Reduced product $A_0 \sqcap A_1$

$$C \underset{\alpha_0}{\overset{\gamma_0}{\leftrightarrows}} A_0 \qquad\qquad C \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} A_1$$

$$C \underset{\alpha_\sqcap}{\overset{\gamma_\sqcap}{\leftrightarrows}} (A_0 \times A_1)_\equiv \quad A_0 \sqcap A_1$$

take the equivalence classes

$$(a_0, a_1) \equiv (a_0', a_1') \Leftrightarrow \gamma_\times(a_0, a_1) = \gamma_\times(a_0', a_1')$$

$$\gamma_\sqcap([a_0, a_1]_\equiv) = \gamma_0(a_0) \cap \gamma_1(a_1)$$

Domain integration

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 suitable A ?

Idea: combine more abstract domains in the same derivation,
different abstract domains for different portions of code!

$$\frac{\vdash_{\mathsf{Sign}^+} [P]\ \mathsf{r}_1\ [R] \qquad \vdash_{\mathsf{Int}} [R]\ \mathsf{r}_2\ [Q]}{\vdash_{\mathsf{Sign}} [P]\ \mathsf{r}_1\ ;\ \mathsf{r}_2\ [Q]}$$

Refine rule

select a more precise domain A'

preserve abstraction of pre-conditions

carry the sub proof in the refined domain

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

move the conclusion to the more abstract domain

A triple $\vdash_A [P]\ r\ [Q]$ is **valid** if $Q \subseteq [\![r]\!]P \subseteq A(Q) \cancel{\subseteq} [\![r]\!]^\#_A A(P)$

Pointed refinement

Suppose we want to extend A with a new approximation $u \in C$

$A \cup \{u\}$ is not necessarily an abstract domain!
must be closed under meet (called Moore closure)

$$A_u \triangleq A \cup \{u \cap a \mid a \in A\}$$

$A_u(c) \triangleq u \cap A(c)$ if $c \leq u$
$A_u(c) \triangleq A(c)$ otherwise

Equivalently $A_u \triangleq A \sqcap I_u$ where $I_u \triangleq \{\perp, u, \top\}$

Example

Let us denote by $[x, y]_{\neq 0}$ the interval-with-a-hole $[x, y] \backslash \{0\}$

Then $\mathsf{Int}_{\neq 0} \triangleq \mathsf{Int} \cup \{[x, y]_{\neq 0} \mid [x, y] \in \mathsf{Int}, x < 0 < y\}$

we have, e.g.

$\mathsf{Int}_{\neq 0} \{-10, -5, 7\} = [-10, 7]_{\neq 0}$

$\mathsf{Int}_{\neq 0} \{-10, -5, 0, 7\} = [-10, 7]$

$\mathsf{Int}_{\neq 0} \{-10, -5\} = [-10, -5]$

Example

Let us denote by $\mathbb{Z}_{\geq 0}$ the set of non-negative integers

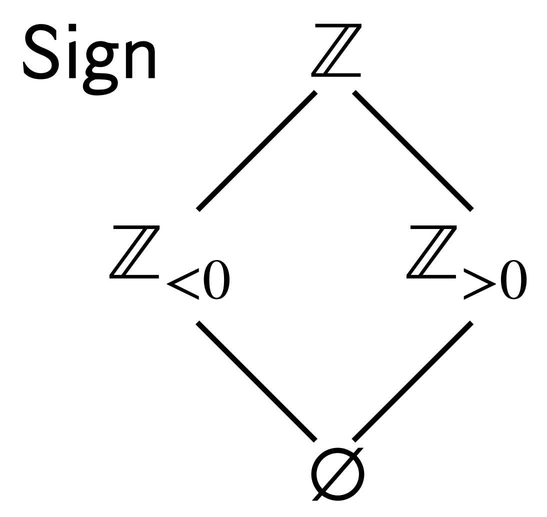
Then $\mathrm{Sign}_{\geq 0} \triangleq \mathrm{Sign} \cup \{\mathbb{Z}_{\geq 0}\}$

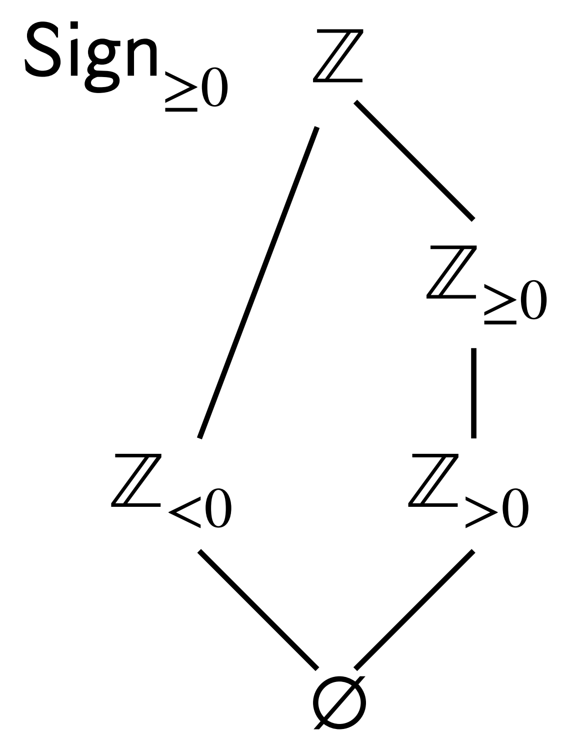
we have, e.g.

$\mathrm{Sign}_{\geq 0}\,\{0\} = \mathbb{Z}_{\geq 0}$

$\mathrm{Sign}_{\geq 0}\,\{1,7\} = \mathbb{Z}_{>0}$

$\mathrm{Sign}_{\geq 0}\,\{-7,0\} = \mathbb{Z}$





Example

$y \in [-100,100]$

$y \in [-199,201]$
odd(y)

$y \in [1,201]$
odd(y)

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

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

$y \in [1,201]$
odd(y)

$x = y \in [1,201]$
odd(y)

$x = y = 1$

Example

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

incomplete in Int

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

Int is non relational

$$\dfrac{\dfrac{\vdash_{\mathsf{Int}_{\neq 0}} [P]\ r_1\ [S]}{\vdash_{\mathsf{Int}} [P]\ r_1\ [S]}\text{[refine]} \qquad \dfrac{\vdash_{\mathsf{Oct}} [S]\ r_2\ [Q]}{\vdash_{\mathsf{Int}} [S]\ r_2\ [Q]}\text{[refine]}}{\vdash_{\mathsf{Int}} [P]\ r_1 ; r_2\ [Q]}\text{[seq]}$$

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

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

Refinement strategy

problems related to automation (ingenuity required):

when and how to apply the consequence rule relax?

when and how to apply the rule refine?

it would be nice to select automatically the most abstract domain where the correctness proof can be completed…

Abstract Interpretation Repair
(AIR)

PLDI 2022

Abstract Interpretation Repair

Roberto Bruni
University of Pisa, Pisa, Italy
roberto.bruni@unipi.it

Roberto Giacobazzi
University of Verona, Verona, Italy
roberto.giacobazzi@univr.it

Roberta Gori
University of Pisa, Pisa, Italy
roberta.gori@unipi.it

Francesco Ranzato
University of Padova, Padova, Italy
francesco.ranzato@unipd.it

Abstract

Abstract interpretation is a sound-by-construction method for program verification: any erroneous program will raise some alarm. However, the verification of correct programs may yield false-alarms, namely it may be *incomplete*. Ideally, one would like to perform the analysis on the most abstract domain that is precise enough to avoid false-alarms. We show how to exploit a weaker notion of completeness, called *local completeness*, to optimally refine abstract domains and thus enhance the precision of program verification. Our main result establishes necessary and sufficient conditions for the existence of an optimal, locally complete refinement, called *pointed shell*. On top of this, we define two repair strategies to remove all false-alarms along a given abstract computation: the first proceeds forward, along with the concrete computation, while the second moves backward within the abstract computation. Our results pave the way for a novel *modus operandi* for automating program verification that we call Abstract Interpretation Repair (AIR): instead of choosing beforehand the right abstract domain, we can start in any abstract domain and progressively repair its local incompleteness as needed. In this regard, AIR is for abstract interpretation what CEGAR is for abstract model checking.

CCS Concepts: • **Theory of computation → Logic and verification**; **Abstraction**; **Semantics and reasoning**; **Program analysis**.

Keywords: abstract interpretation, program analysis, program verification, local completeness, CEGAR.

ACM Reference Format:
Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2022. Abstract Interpretation Repair. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '22), June 13–17, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3519939.3523453

PLDI '22, June 13–17, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9265-5/22/06.
https://doi.org/10.1145/3519939.3523453

1 Introduction

It is widely acknowledged that the chance of formally verifying programs is fundamental to effectively rise the confidence level that the code we use is correct [23]. However, as emerged in the last decades, this approach to program correctness becomes socially acceptable when these proofs are not only rigorous but also explainable, meaning that they have to rely upon a largely recognized proof method which has to be simple and inspectable [22]. As advocated by Vardi [61], checking program correctness "*is a cost that must be justified by the benefits*". The last 50 years have shown an impressive flourishing of formal methods and tools for achieving this ambitious goal [32]. These include, among the others: Certified compilers [42], certified analyzers [39], advanced type checkers [49, 50], sophisticated static analyzers [6, 19, 25] and software model checkers [3, 37].

A high degree of confidence in the correctness of a software system, and of its most critical components, can be obtained when the code is certified by a *sound* and *complete* (viz. precise) static analyzer [14, 25]. Abstract interpretation [17] was introduced with this purpose in mind: simplify the proof of correctness by interpreting the program in a simplified, *abstract*, domain. This provides a general methodology for the design of *sound-by-construction* analysis tools.

The Problem. The soundness of an abstract interpreter, or program analyzer, means that all true-alarms are caught. However, it is often the case that some false-alarms are reported. Actually, when false-alarms overwhelm true ones, then the program analyzer may become poorly trustworthy. This is a consequence of the approximation inherent in the making of an otherwise undecidable analysis decidable. As all alarm systems, program analysis is credible when few false-alarms are reported, ideally none. The problem we address in this paper is *how to derive the most abstract domain to decide program correctness without raising false-alarms.*

The absence of false-alarms in program analysis is closely related to the property of *completeness in abstract interpretation* [33]. As an illustrative example, consider the program

"AIR is for abstract interpretation what CEGAR is for abstract model checking"

CEGAR in a nutshell

Model checking

A model, a (large) finite state transition system $\langle \Sigma, \rightarrow\ , I \rangle$

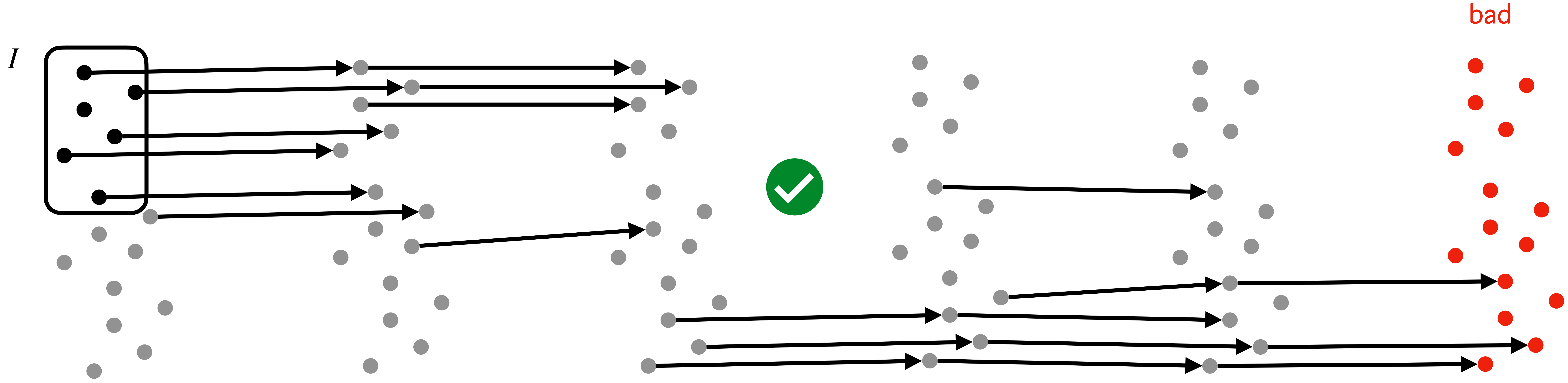
A temporal logic specification φ (e.g. AG ¬bad)

no bad state is reachable

Does the model satisfy φ?

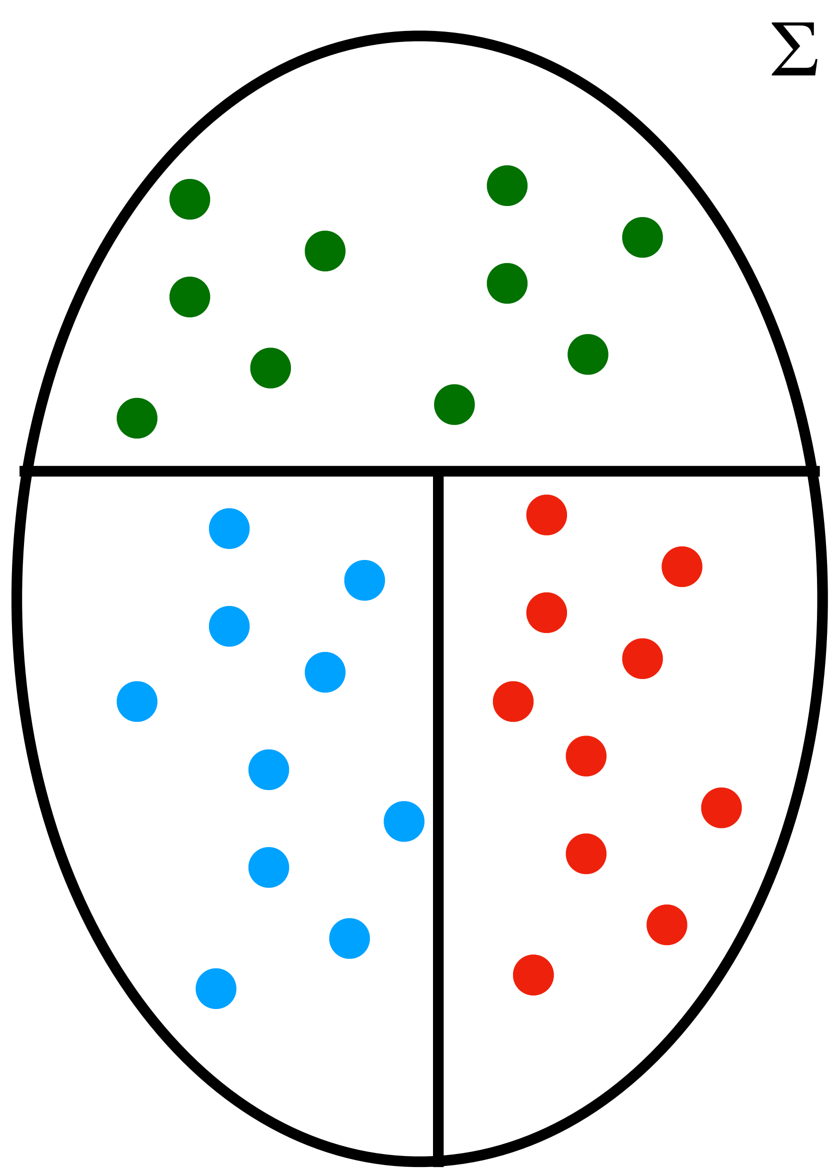
yes

no, here is a counterexample $s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n$



Abstract transition system

A partition $[\,\cdot\,]_\#$ of Σ

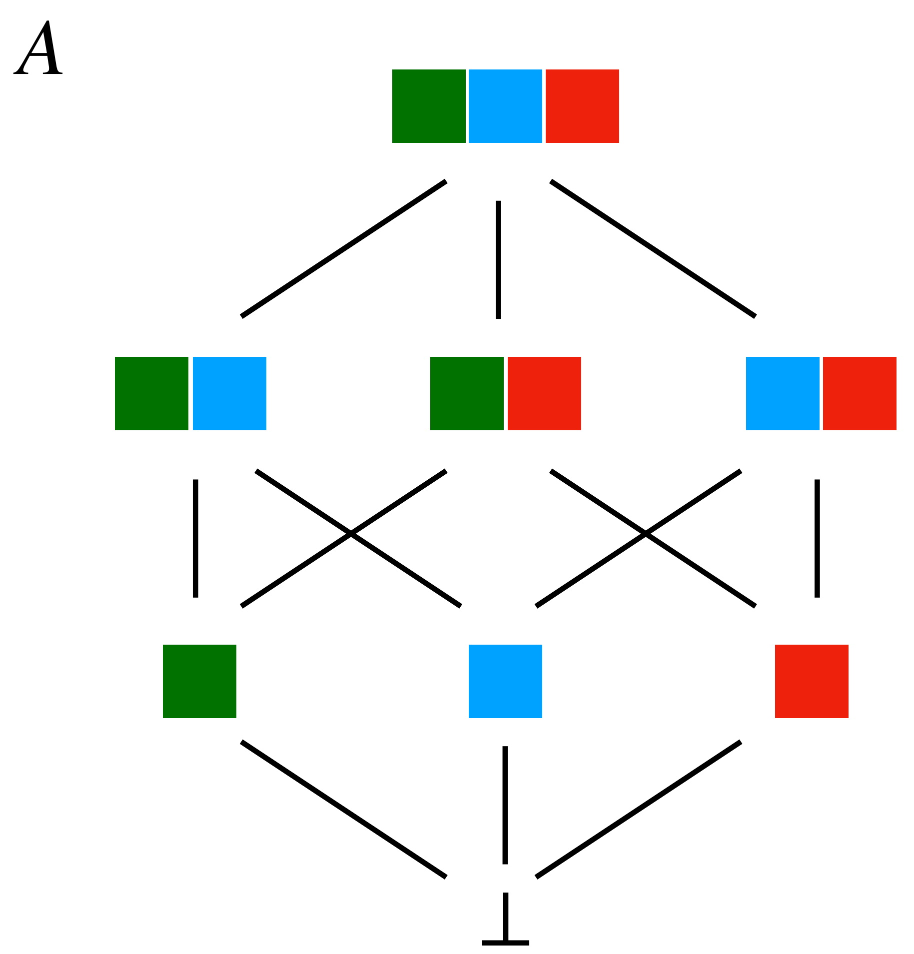


A partitioning abstraction A of $\wp(\Sigma)$

$$A(X) \triangleq \bigcup_{x \in X} [x]_\#$$

Existential abstract transition relation

$$X \rightarrow^\# [y]_\# \Leftrightarrow \exists x \in X\,.\; x \rightarrow y$$



$$\langle A, \rightarrow^\#, A(I) \rangle$$

Abstract model checking

An abstract model $\langle A, \rightarrow^{\#}, A(I) \rangle$

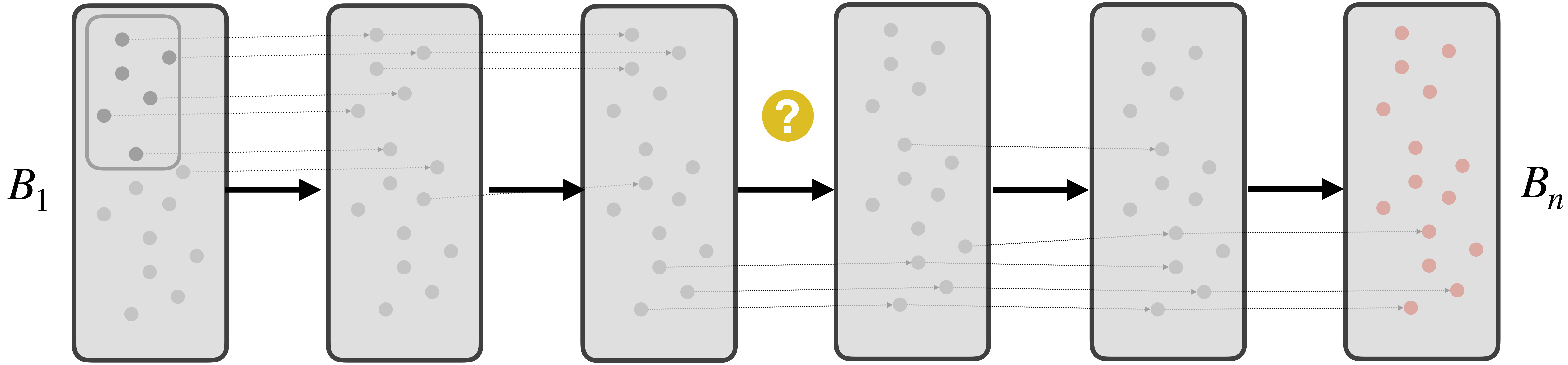
A temporal logic specification φ (e.g. AG ¬bad)

Does the model satisfy φ?

yes

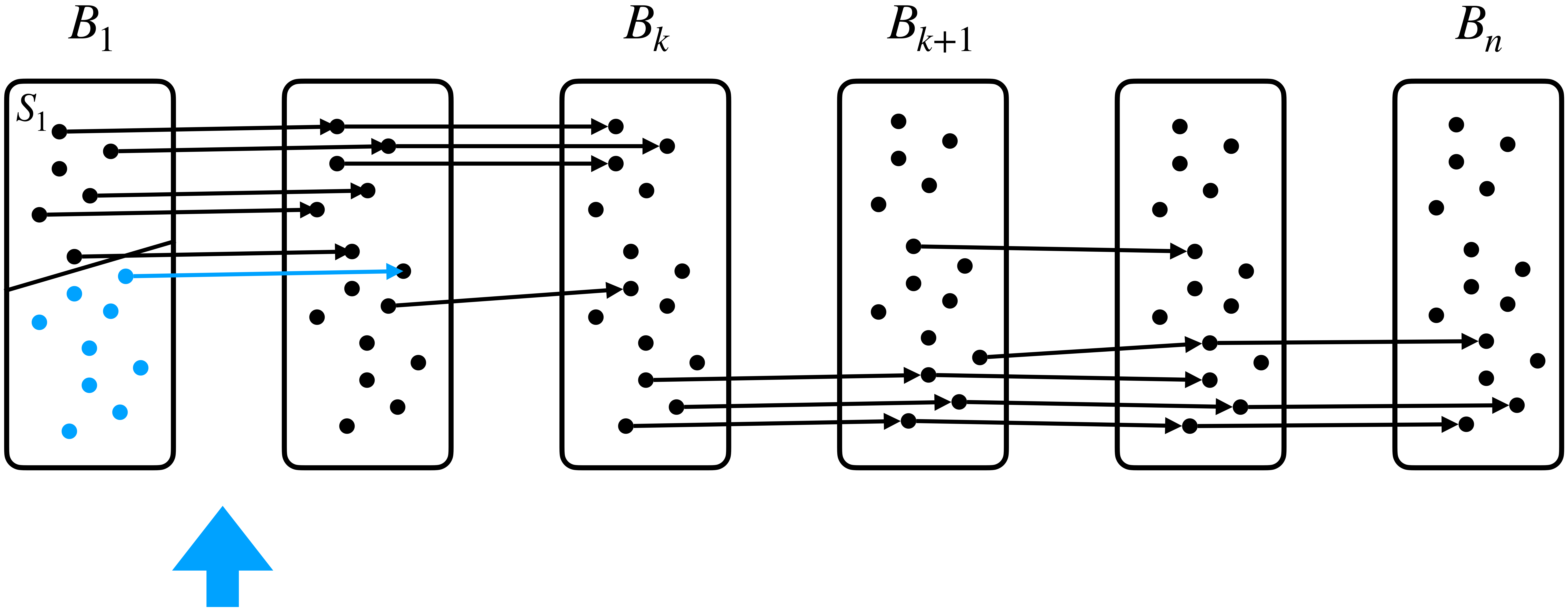
no, here is a possibly spurious abstract counterexample

$B_1 \rightarrow B_2 \rightarrow \ldots \rightarrow B_n$



CEGAR

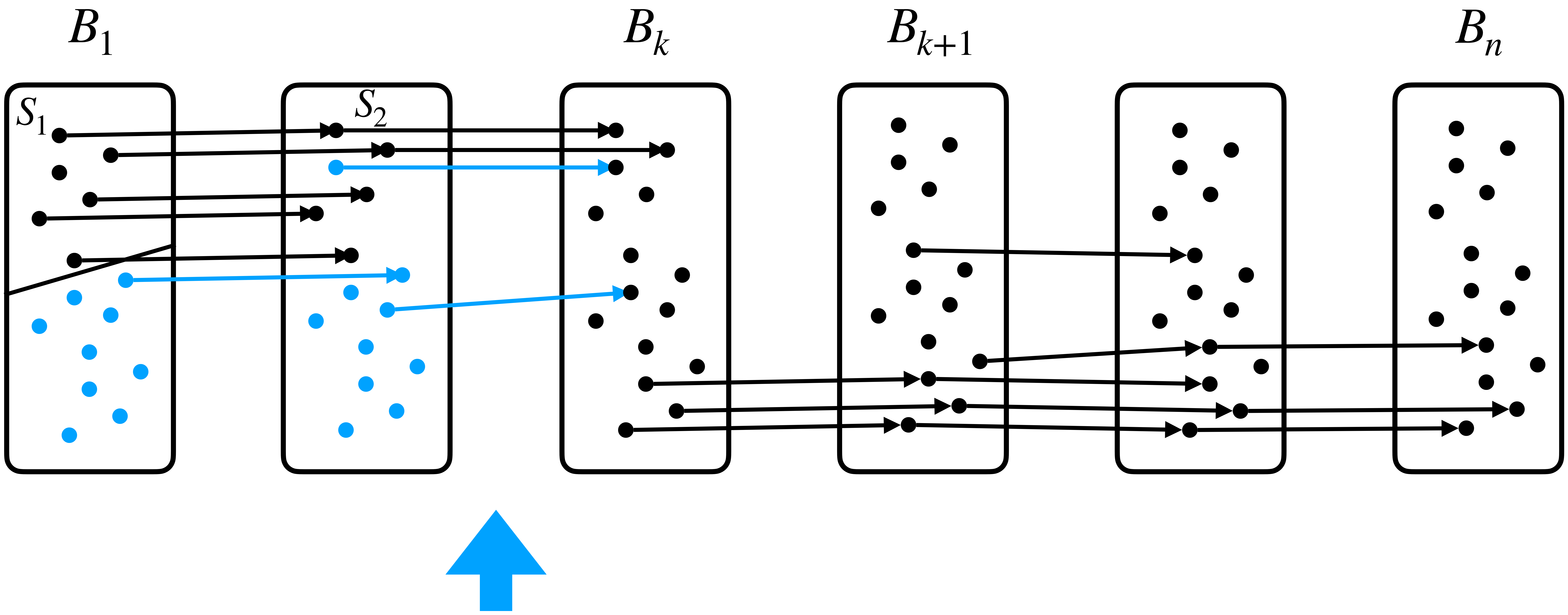
CounterExample Guided Abstraction Refinement:
If the counterexample is spurious, refine the partition to eliminate the abstract path and repeat the analysis



S_i are the reachable states within B_i

CEGAR

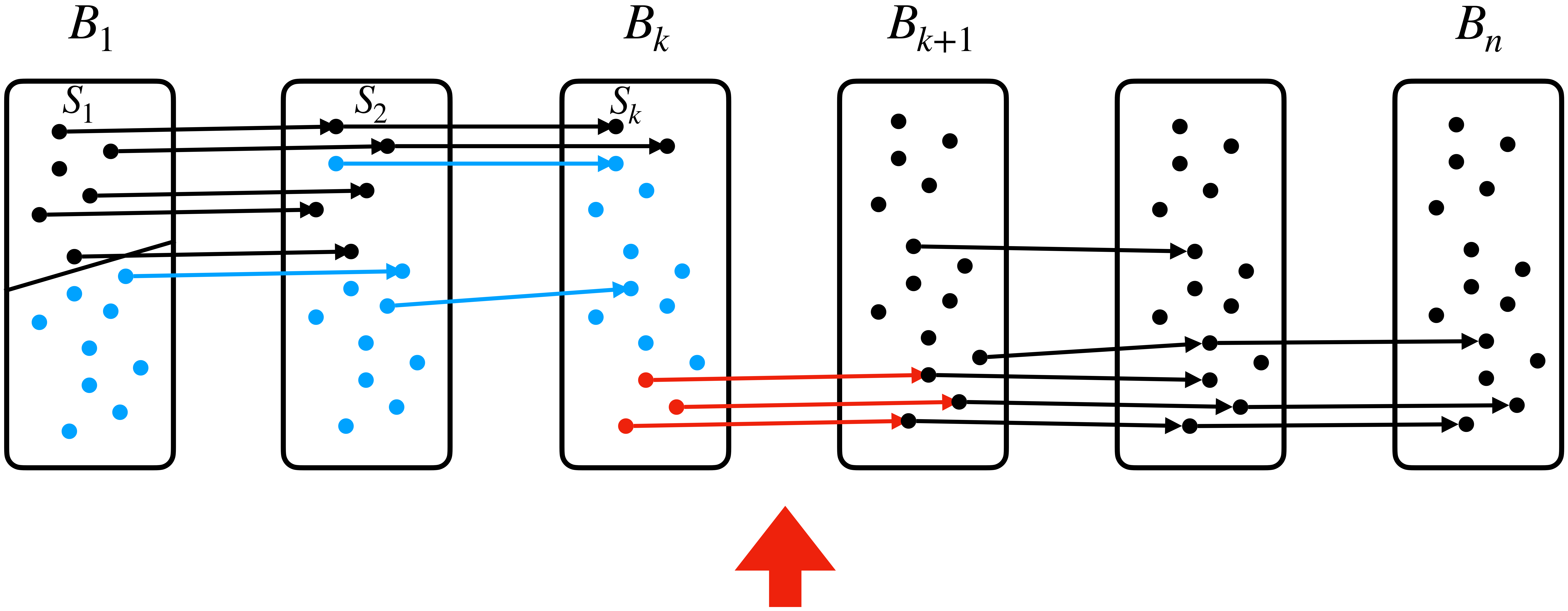
CounterExample Guided Abstraction Refinement:
If the counterexample is spurious, refine the partition to eliminate the abstract path and repeat the analysis



S_i are the reachable states within B_i

CEGAR

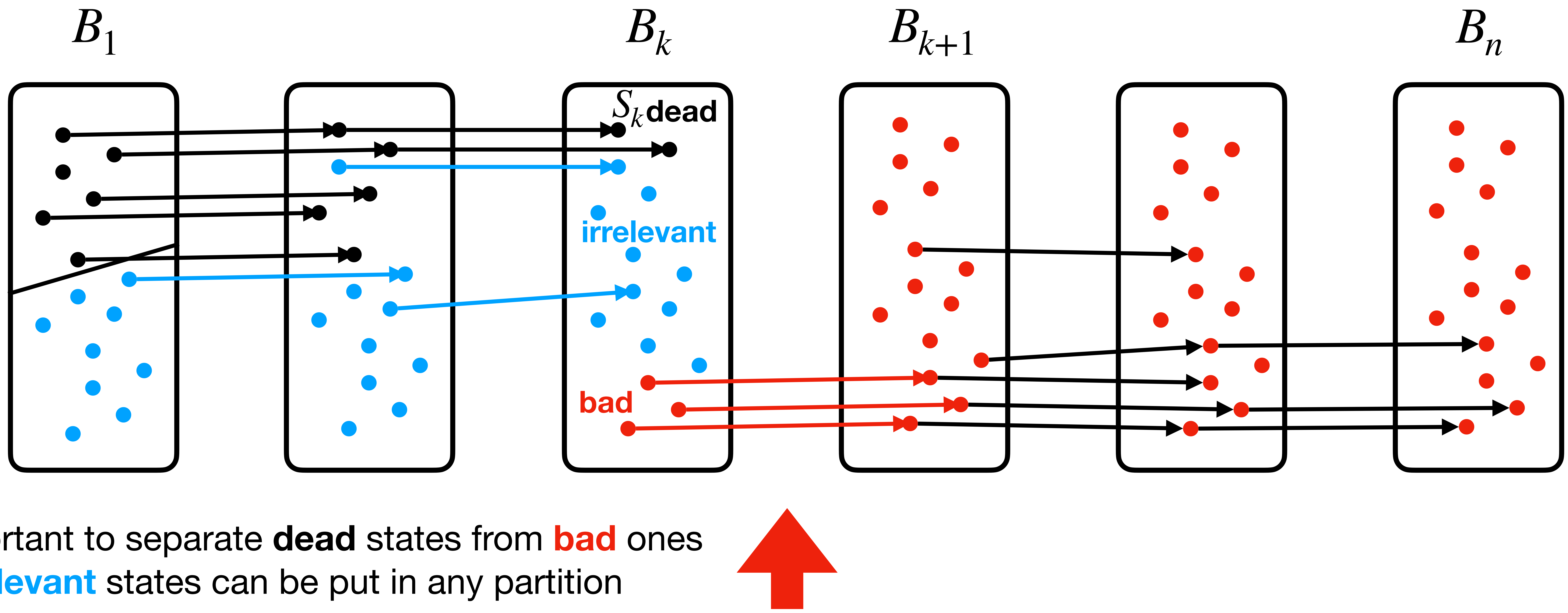
CounterExample Guided Abstraction Refinement:
If the counterexample is spurious, refine the partition to eliminate the abstract path and repeat the analysis



S_i are the reachable states within B_i

CEGAR

CounterExample Guided Abstraction Refinement:
If the counterexample is spurious, refine the partition to eliminate the abstract path and repeat the analysis



it is important to separate **dead** states from **bad** ones
irrelevant states can be put in any partition

CEGAR and local completeness

Let $\pi = \langle B_1, \ldots, B_n \rangle$ and abstract counterexample and let $\mathsf{post}(X) \triangleq \{t \mid \exists s \in X \,.\, s \to t\}$ be the usual successor transformer

Define $\mathsf{post}_{\pi_i}(X) \triangleq \mathsf{post}(X) \cap B_{i+1}$ and the sequence of reachable states $S_1 \triangleq I \cap B_1 \neq \varnothing$ and $S_{i+1} \triangleq \mathsf{post}_{\pi_i}(S_i) = \mathsf{post}(S_i) \cap B_{i+1}$

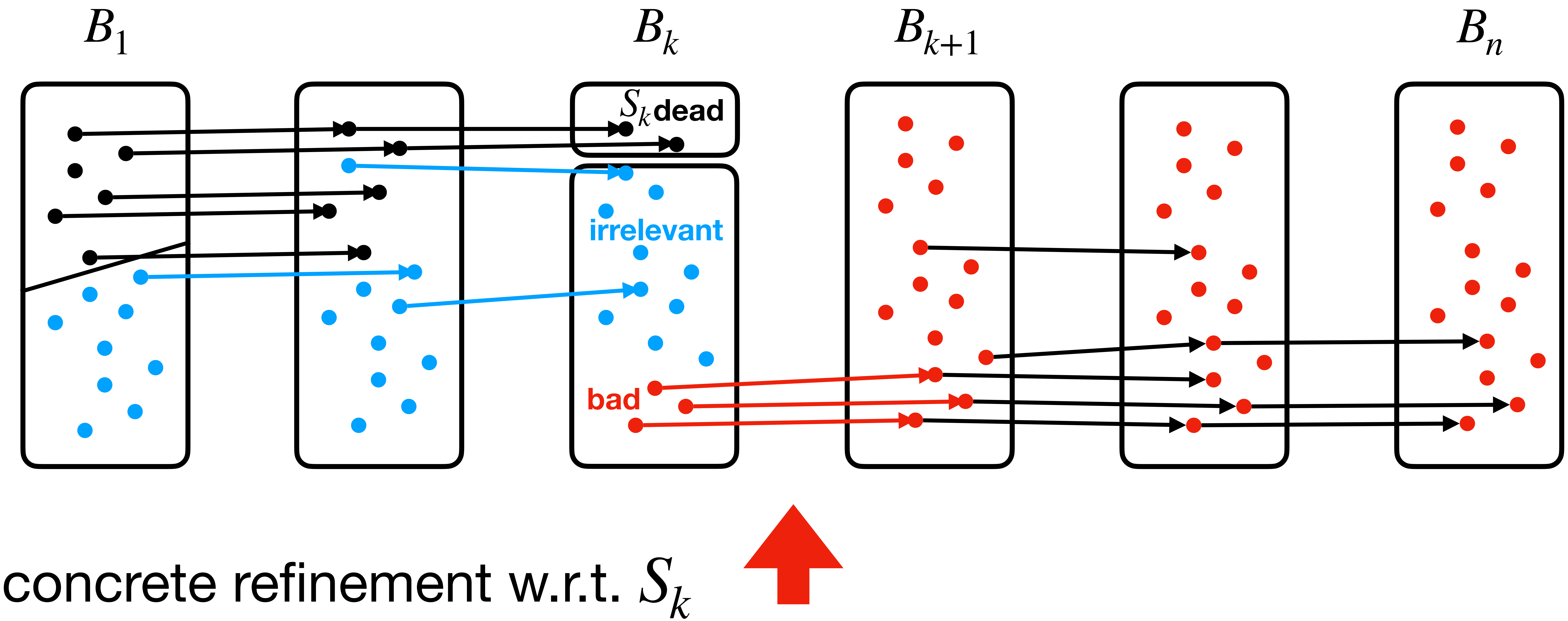
Lemma.

π is not spurious iff $\mathbb{C}^A_{S_i}(\mathsf{post}_{\pi_i})$ for all $i \in [1, n-1]$

(i.e. iff each post_{π_i} is locally complete in A for S_i)

Partition refinement

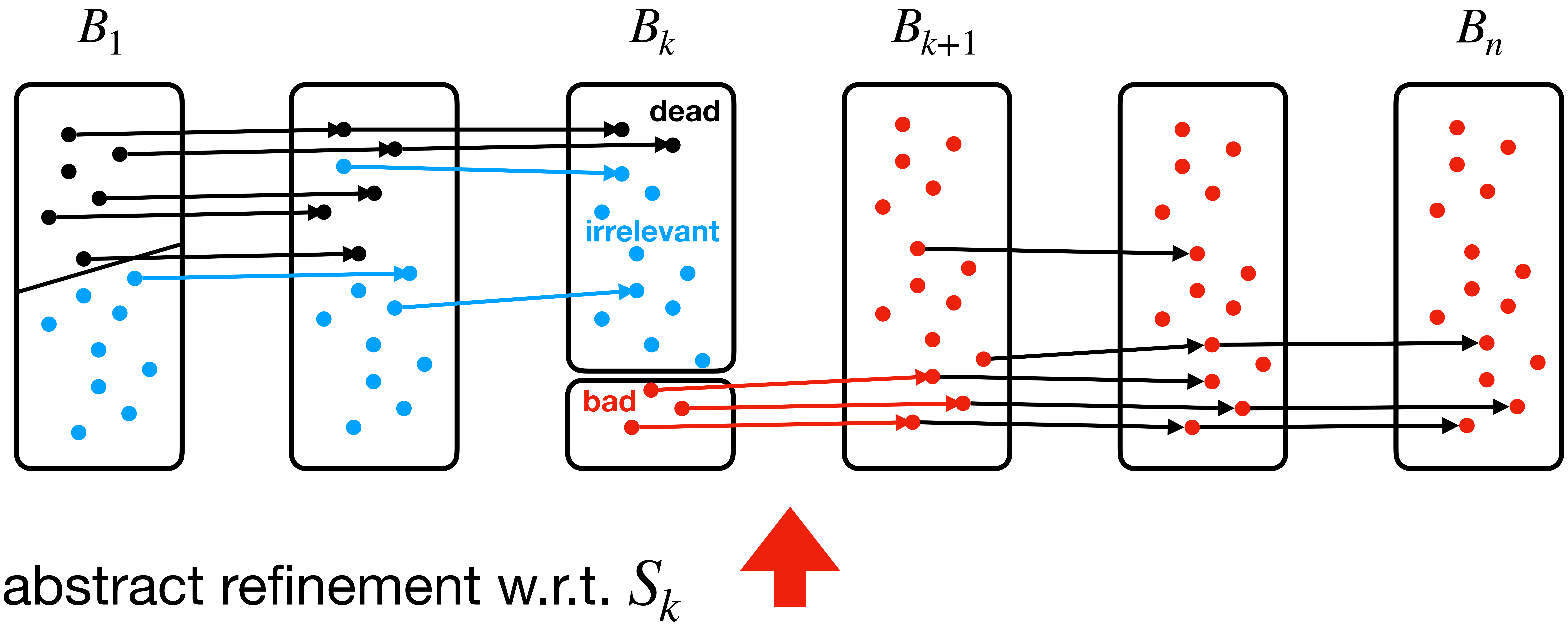
To eliminate the spurious counterexample we can refine the current abstraction $A(S_k) = B_k$



most concrete refinement w.r.t. S_k

Partition refinement

To eliminate the spurious counterexample we can refine the current abstraction $A(S_k) = B_k$



most abstract refinement w.r.t. S_k

Forward repair

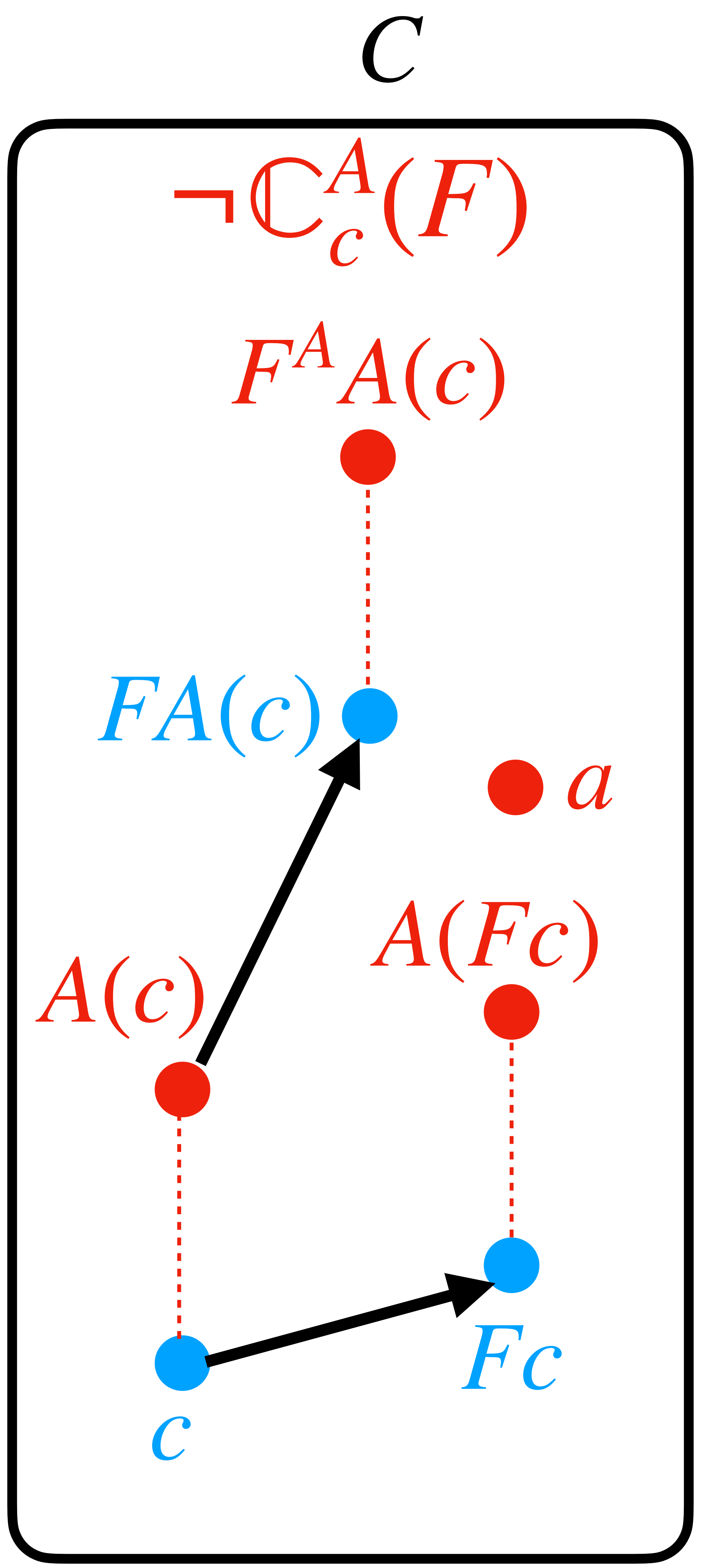
From CEGAR to program analysis

Consider the verification problem $Fc \leq a$ for some expressible $a = A(a)$

We have seen that $Fc \leq a \Leftrightarrow A(Fc) \leq a$

Moreover, if $\mathbb{C}_c^A(F)$ then $Fc \leq a \Leftrightarrow F^A A(c) \leq a$

A spurious counterexample for the abstract analysis arises when $Fc \leq a$ but $F^A A(c) \not\leq a$ because $\neg\mathbb{C}_c^A(F)$



From CEGAR to program analysis

Suppose $F \triangleq F_n \circ \ldots \circ F_1$, the equality $F^A A(c) = A(Fc)$ follows as a consequence of n local completeness proof obligations $AF_k A(c_k) = A(F_k c_k)$ where $c_1 \triangleq c$ and $c_{k+1} \triangleq F_k c_k$

From CEGAR to program analysis

Suppose $F \triangleq F_n \circ \ldots \circ F_1$, the equality $F^A A(c) = A(Fc)$ follows as a consequence of n local completeness proof obligations $AF_k A(c_k) = A(F_k c_k)$ where $c_1 \triangleq c$ and $c_{k+1} \triangleq F_k c_k$

From CEGAR to program analysis

Suppose $F \triangleq F_n \circ \ldots \circ F_1$, the equality $F^A A(c) = A(Fc)$ follows as a consequence of n local completeness proof obligations $AF_k A(c_k) = A(F_k c_k)$ where $c_1 \triangleq c$ and $c_{k+1} \triangleq F_k c_k$

From CEGAR to program analysis

Suppose $F \triangleq F_n \circ \ldots \circ F_1$, the equality $F^A A(c) = A(Fc)$ follows as a consequence of n local completeness proof obligations $AF_k A(c_k) = A(F_k c_k)$ where $c_1 \triangleq c$ and $c_{k+1} \triangleq F_k c_k$

From CEGAR to program analysis

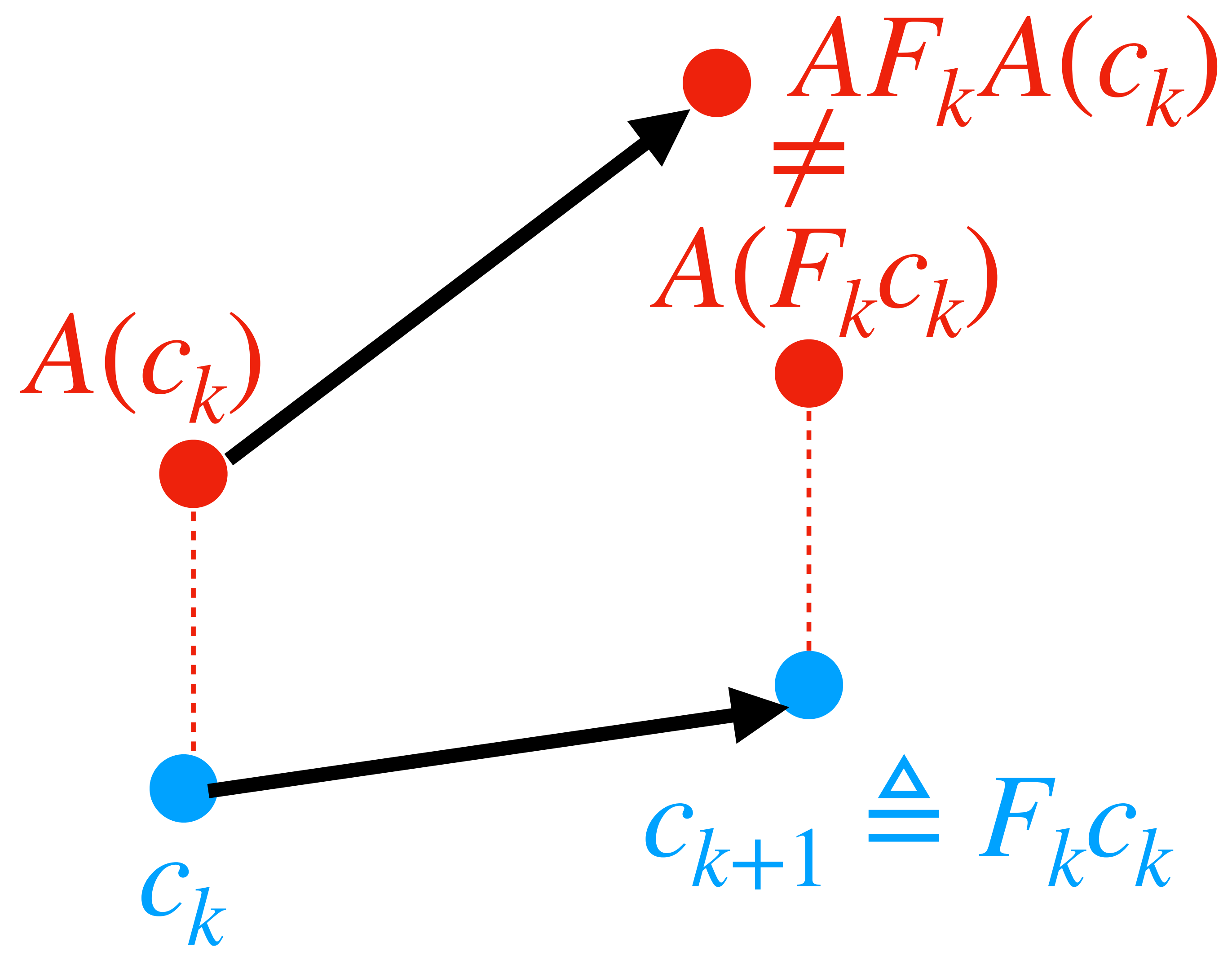
Suppose $F \triangleq F_n \circ \ldots \circ F_1$, the equality $F^A A(c) = A(Fc)$ follows as a consequence of n local completeness proof obligations

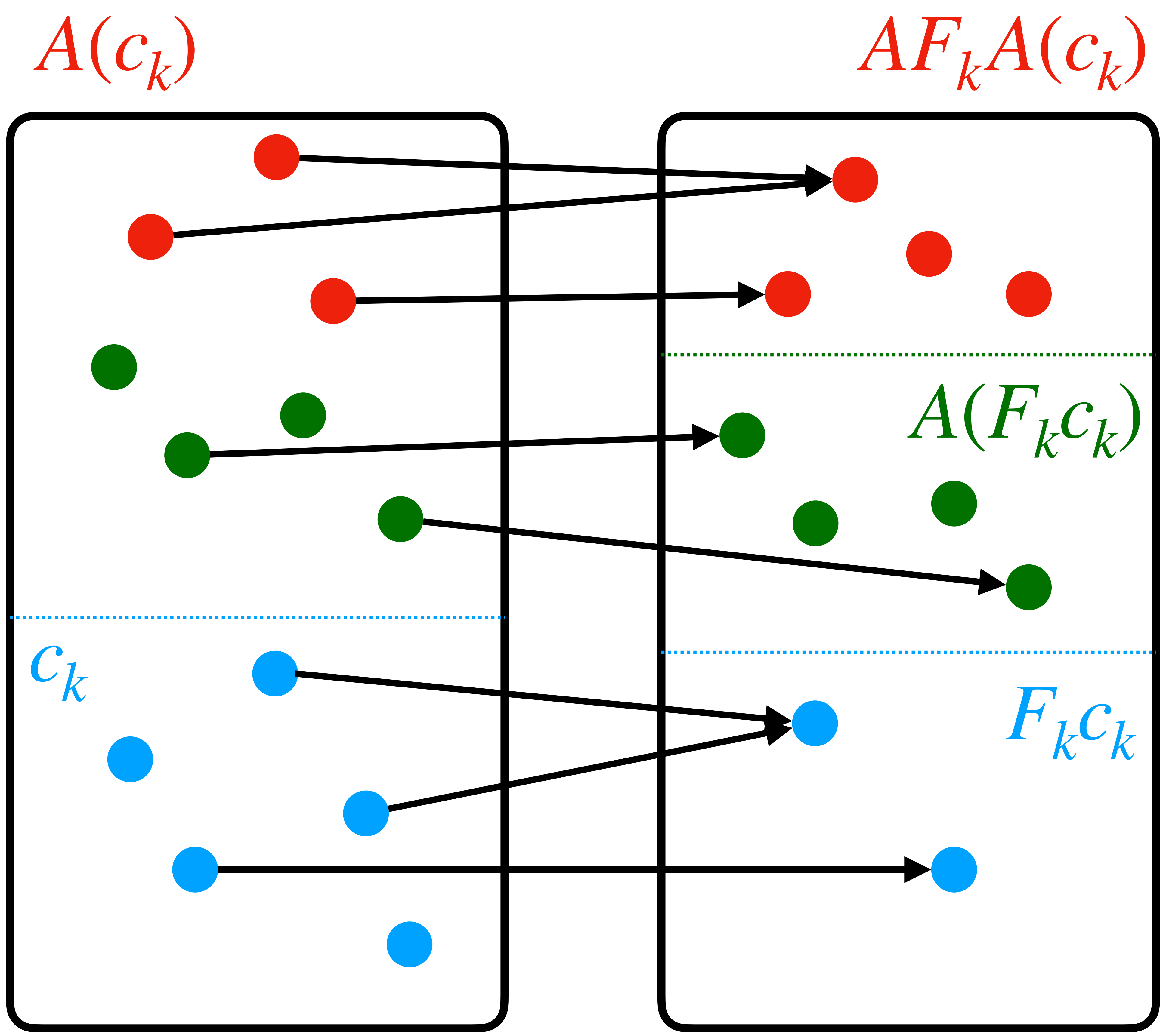
$AF_k A(c_k) = A(F_k c_k)$ where $c_1 \triangleq c$ and $c_{k+1} \triangleq F_k c_k$

BCA repair

Imagine $F_k \triangleq [\![e_k]\!]$ for some atomic command e_k

$\neg \mathbb{C}^{A}_{c_k}(F_k)$ ✖





$AF_kA(c_k) \setminus A(F_kc_k)$
red states are the sources of incompleteness

we would like to introduce a better approximation u than $A(c_k)$ for c_k such that:

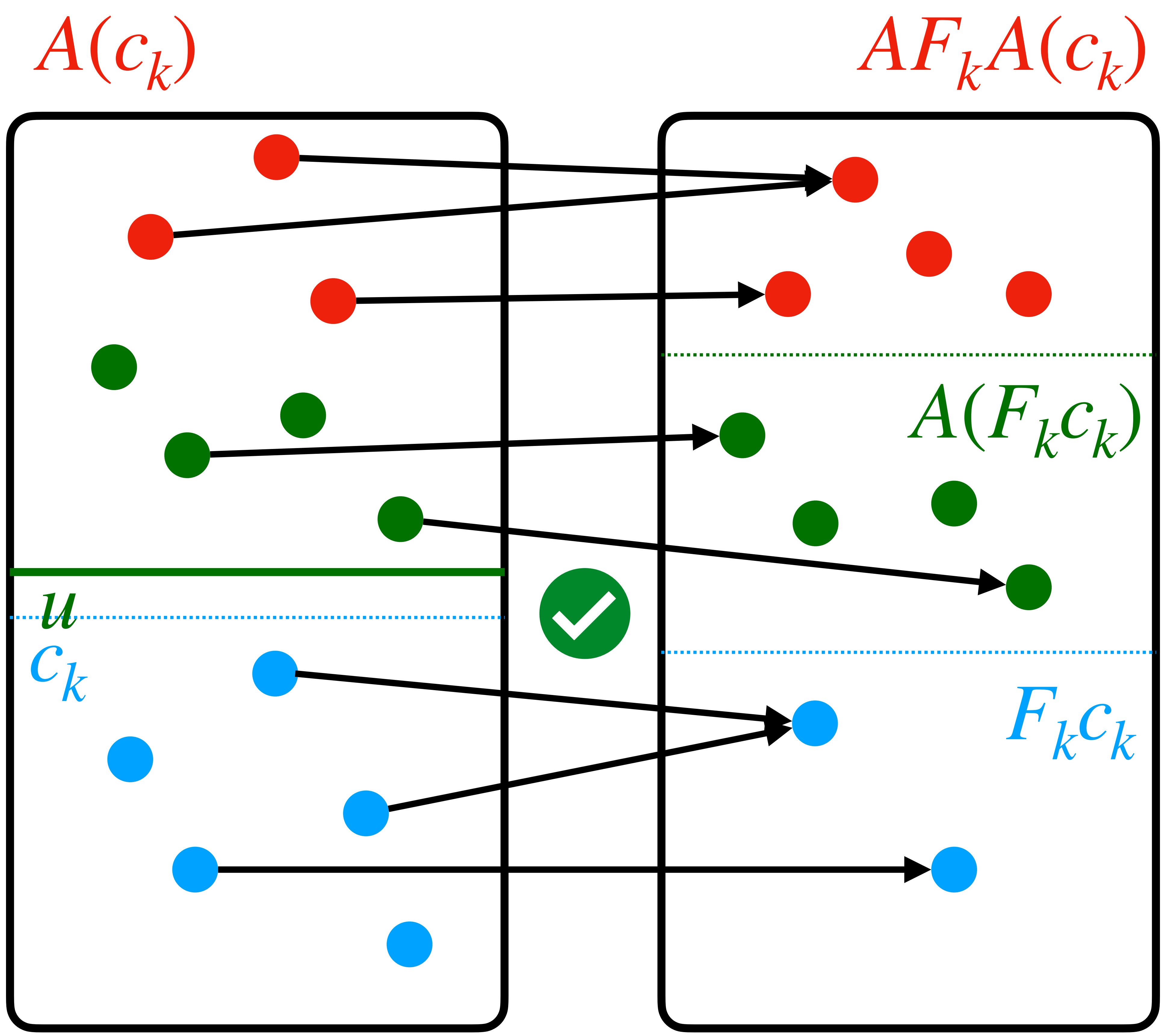
$c_k \leq u \leq A(c_k)$ and

$A_uF_ku = A_uF_kc_k$

pointed refinement

pointed refinement

BCA repair



$AF_kA(c_k) \setminus A(F_kc_k)$

red states are the sources of incompleteness

we would like to introduce a better approximation u than $A(c_k)$ for c_k such that:

$c_k \leq u \leq A(c_k)$ and

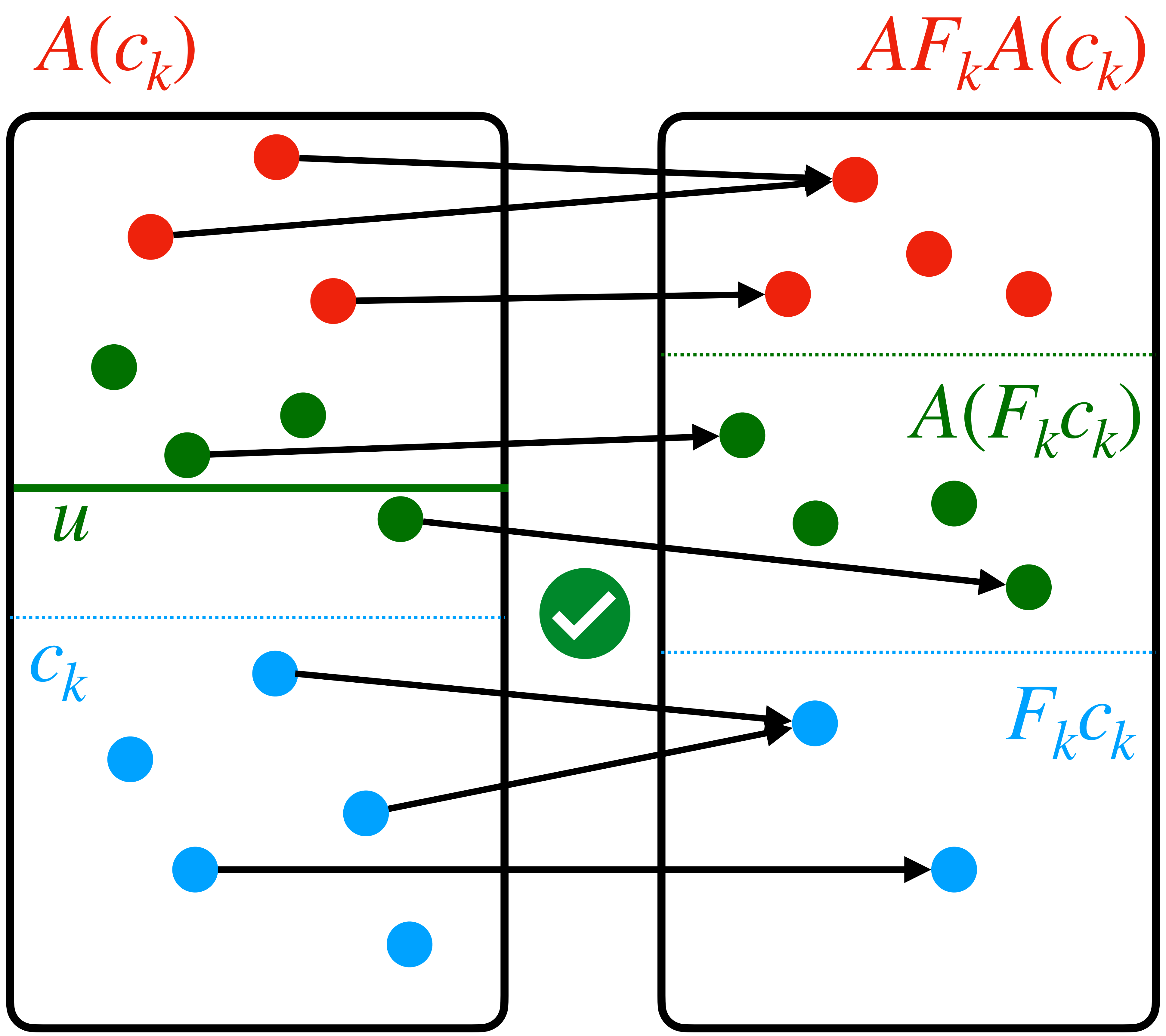
$A_uF_ku = A_uF_kc_k$

pointed refinement

pointed refinement

most concrete refinement: $u \triangleq c_k$

BCA repair



$AF_kA(c_k) \setminus A(F_kc_k)$

red states are the sources of incompleteness

we would like to introduce a better approximation u than $A(c_k)$ for c_k such that:

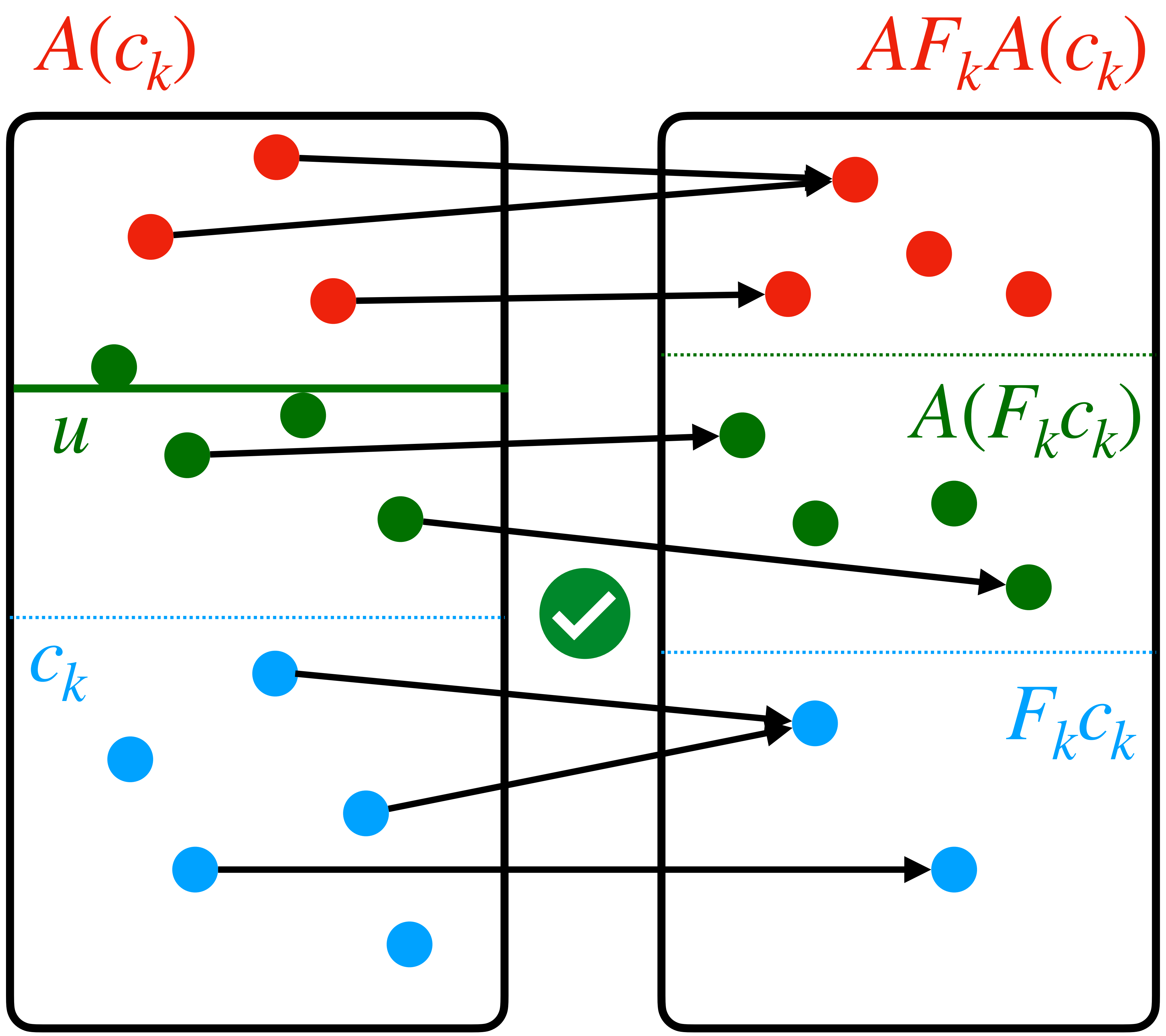
$c_k \leq u \leq A(c_k)$ and

$A_uF_ku = A_uF_kc_k$

pointed refinement

pointed refinement

BCA repair



$AF_kA(c_k)\setminus A(F_kc_k)$
red states are the sources of incompleteness

we would like to introduce a better approximation u than $A(c_k)$ for c_k such that:

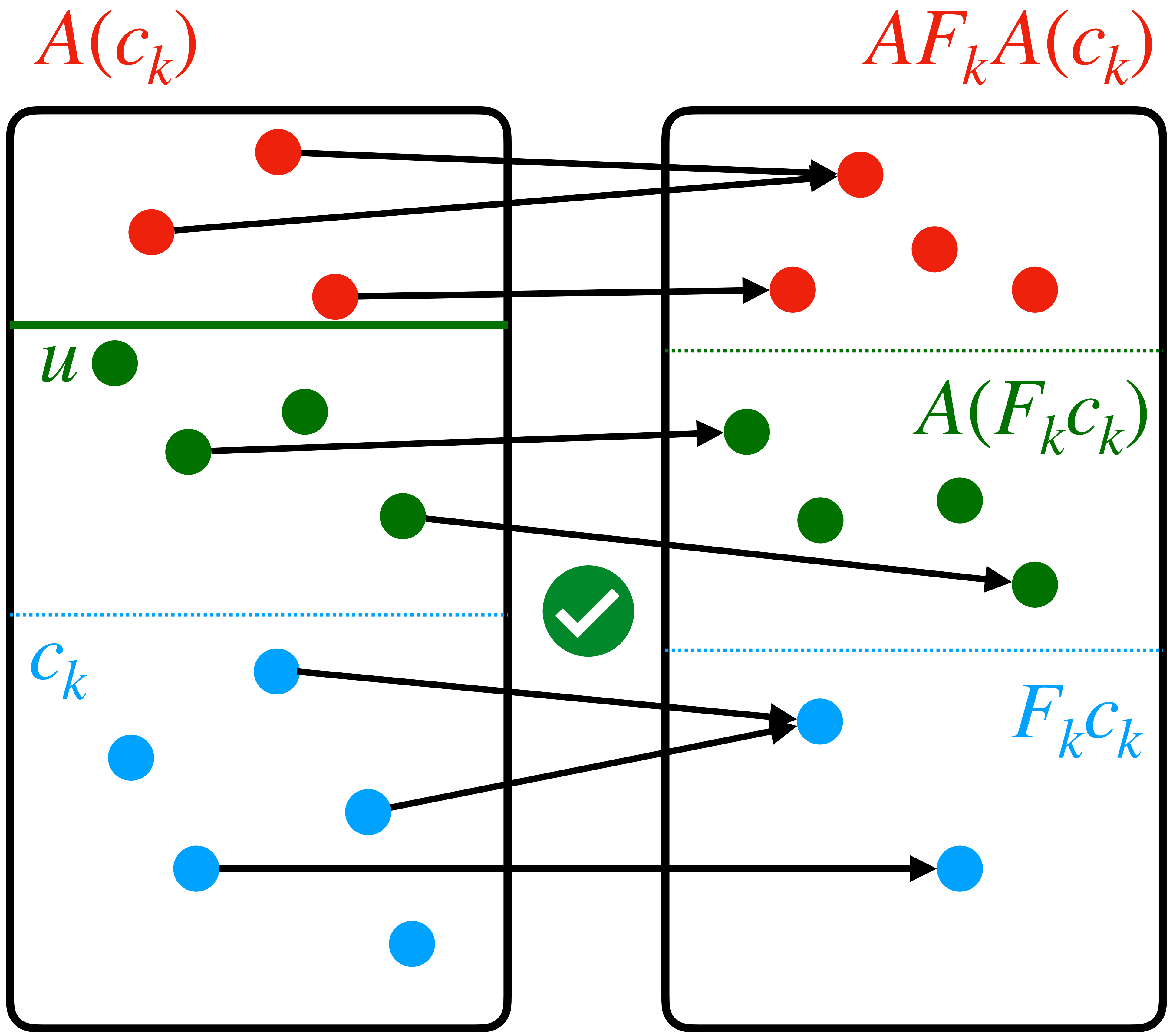
$c_k \leq u \leq A(c_k)$ and

$A_uF_ku = A_uF_kc_k$

pointed refinement

pointed refinement

BCA repair



$AF_kA(c_k) \setminus A(F_kc_k)$
red states are the sources of incompleteness

we would like to introduce a better approximation u than $A(c_k)$ for c_k such that:

$c_k \leq u \leq A(c_k)$ and

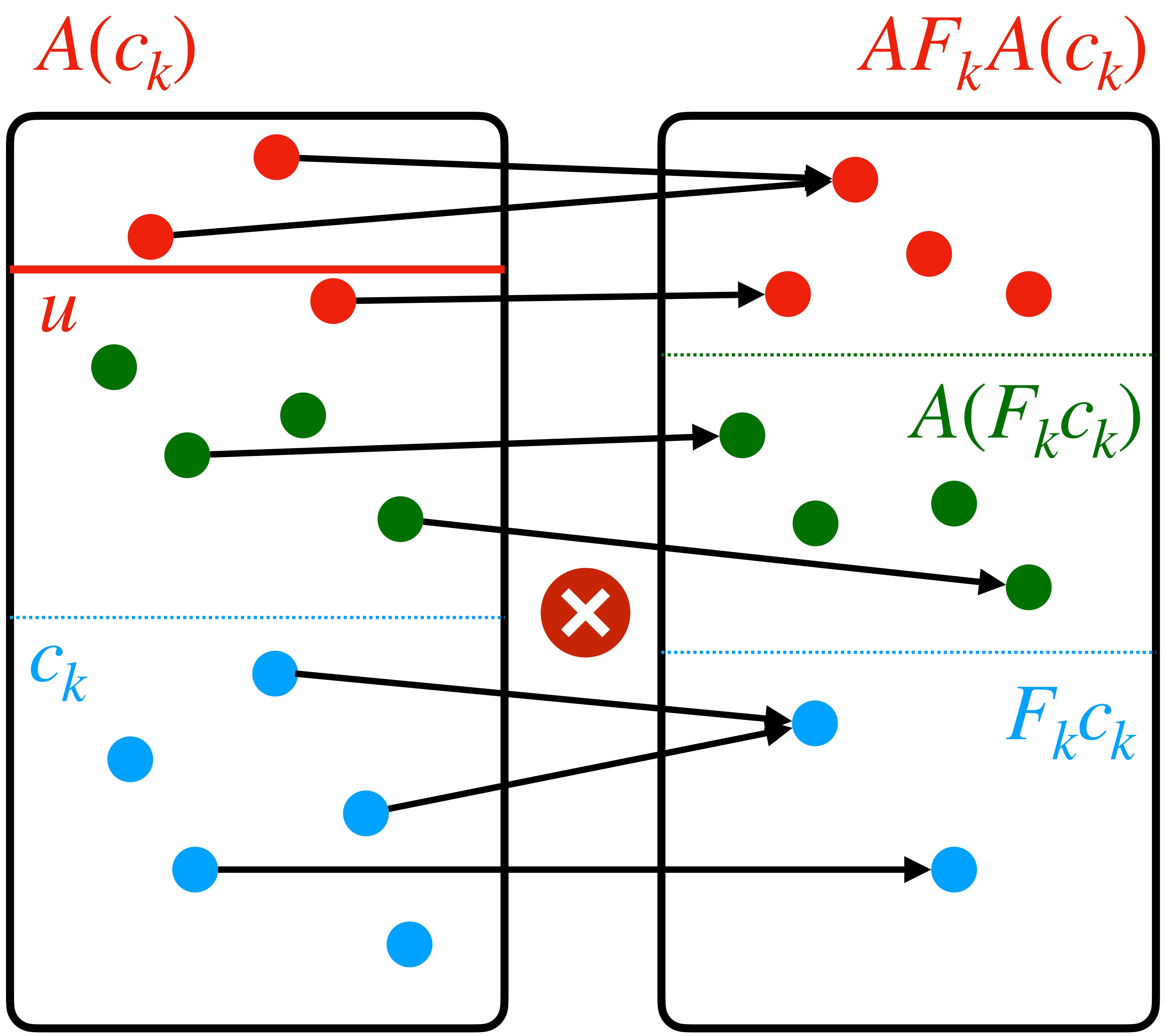
$A_uF_ku = A_uF_kc_k$

pointed refinement

pointed refinement

most abstract possible refinement

BCA repair



$AF_kA(c_k) \setminus A(F_kc_k)$

red states are the sources of incompleteness

we would like to introduce a better approximation u than $A(c_k)$ for c_k such that:

$c_k \leq u \leq A(c_k)$ and

$A_uF_ku = A_uF_kc_k$

pointed refinement

pointed refinement

erroneous refinement

Pointed shell

Which refinement A_u when a proof obligation $\mathbb{C}_c^A(F)$ fails?

Candidates: $\{x \in C \mid x \leq A(c), \mathbb{C}_c^{A_x}(F)\}$

Most concrete solution: $u \triangleq c$

Most abstract solution: $u \in \max\{x \in C \mid x \leq A(c), \mathbb{C}_c^{A_x}(F)\}$

In the case of guards (when $\mathbb{C}_P^A(b)$ fails):

$u \triangleq (A(P \wedge b) \wedge b) \vee (A(P \wedge \neg b) \wedge \neg b)$

A forward repair strategy for LCL

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

Forward repair strategy

additional points, if any

Initial call to $\text{fRepair}_A(\varnothing, P, r)$

```
Function fRepair_A(N, P, r)
    found := false;
    do
        out := find_A(N, P, r);
        switch out do
            case Q do found := true;                 // underapprox.
            case ⟨R, e⟩ do N := refine_A(N, R, e);   // incompl.
    while (¬found);
    return ⟨N, out⟩;
```

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

found Q such that $\vdash_{A_N} [P]$ r $[Q]$

failed proof obligation $\neg\mathbb{C}_R^{A_N}(\text{e})$

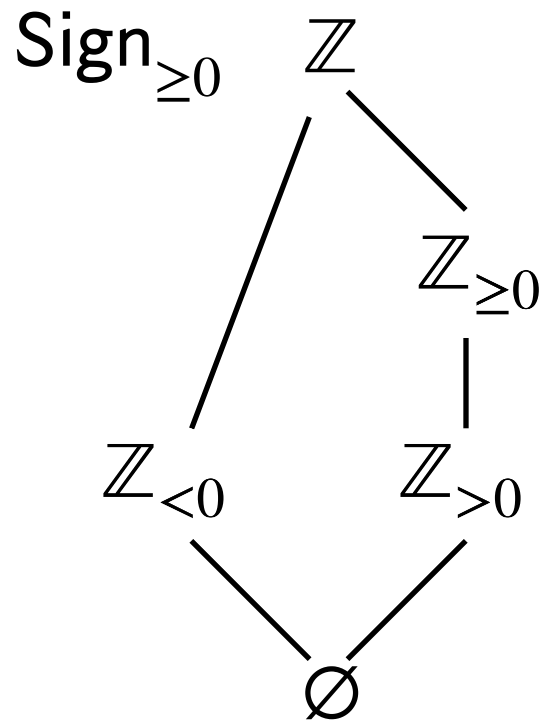
update N and retry

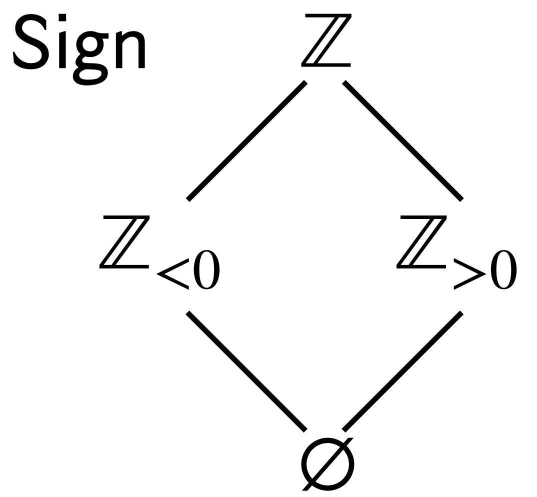
select new pointed refinement

returns the latest N and Q such that $\vdash_{A_N} [P]$ r $[Q]$

Example

$$u \triangleq (\mathsf{Sign}(\{0,1\} \wedge x \neq 0) \wedge x \neq 0) \vee (\mathsf{Sign}(\{0,1\} \wedge x = 0) \wedge x = 0)$$
$$= (\mathsf{Sign}\{1\} \wedge x \neq 0) \vee (\mathsf{Sign}\{0\} \wedge x = 0)$$
$$= (x > 0 \wedge x \neq 0) \vee (\top \wedge x = 0)$$
$$= (x > 0 \vee x = 0)$$
$$= (x \geq 0)$$





$$\mathsf{Sign}[\![x \neq 0]\!]\mathsf{Sign}\{0,1\} = \mathsf{Sign}[\![x \neq 0]\!]\mathbb{Z} = \mathsf{Sign}(x \neq 0) = \mathbb{Z}$$
$$\mathsf{Sign}[\![x \neq 0]\!]\{0,1\} = \mathsf{Sign}\{1\} = \mathbb{Z}_{>0}$$

fails!

succeed!

succeed!

abstract fixpoint!

$$\dfrac{\mathbb{C}^{\mathsf{Sign}}_{\{0,1\}}(x \neq 0) \qquad \mathbb{C}^{\mathsf{Sign}}_{\{0,1\}}(x = 0) \qquad \dfrac{\mathbb{C}^{\mathsf{Sign}}_{\{1\}}(x+1)}{\vdash_{\mathsf{Sign}} [\{1\}]\ x := x + 1\ [\{2\}]} \qquad \{2\} \subseteq \mathsf{Sign}(\{0,1\}) = \mathbb{Z}}{\vdash_{\mathsf{Sign}} [\{0,1\}]\ \text{while } x \neq 0 \text{ do } x := x + 1\ [\{0\}]}$$

Example

Note that $\mathsf{Sign}_{\geq 0} = \{\ \bot\ , \mathbb{Z}_{>0}, \mathbb{Z}_{<0}, \mathbb{Z}_{\geq 0}, \mathbb{Z}\}$

is "smaller" than $\mathsf{Sign}^{+} = \{\ \bot\ , \mathbb{Z}_{>0}, \mathbb{Z}_{=0}, \mathbb{Z}_{<0}, \mathbb{Z}_{\geq 0}, \mathbb{Z}_{\neq 0}, \mathbb{Z}_{\leq 0}, \mathbb{Z}\}$

where we carried out the proof previously

Assuming $\mathit{Spec} \triangleq \mathbb{Z}_{>0}$ we now know that 0 is a true positive, differently from the abstract analysis $[\![\text{while } x \neq 0 \text{ do } x := x + 1]\!]^{\#}_{\mathsf{Sign}_{\geq 0}} \mathsf{Sign}_{\geq 0}\{0,1\} = \mathbb{Z}_{\geq 0}$

succeed!

succeed!

succeed!

abstract fixpoint!

$$\frac{\mathbb{C}^{\mathsf{Sign}_{\geq 0}}_{\{0,1\}}(x \neq 0) \qquad \mathbb{C}^{\mathsf{Sign}_{\geq 0}}_{\{0,1\}}(x = 0) \qquad \dfrac{\mathbb{C}^{\mathsf{Sign}_{\geq 0}}_{\{1\}}(x+1)}{\vdash_{\mathsf{Sign}_{\geq 0}} [\{1\}]\ x := x + 1\ [\{2\}]} \qquad \{2\} \subseteq \mathsf{Sign}_{\geq 0}(\{0,1\}) = \mathbb{Z}_{\geq 0}}{\vdash_{\mathsf{Sign}_{\geq 0}} [\{0,1\}]\ \text{while } x \neq 0 \text{ do } x := x + 1\ [\{0\}]}$$

Questions

Question 1

What is the most abstract pointed refinement of Int to use when $\neg\mathbb{C}^{\mathsf{Int}}_{\{-7,7\}}(x > 4)$?

Int_u where:

$$
\begin{aligned}
u &\triangleq (\mathsf{Int}(\{-7,7\} \wedge x > 4) \wedge x > 4) \vee (\mathsf{Int}(\{-7,7\} \wedge x \leq 4) \wedge x \leq 4) \\
&= (\mathsf{Int}\{7\} \wedge x > 4) \vee (\mathsf{Int}\{-7\} \wedge x \leq 4) \\
&= ([7,7] \wedge x > 4) \vee ([-7,-7] \wedge x \leq 4) \\
&= ([7,7] \vee [-7,-7]) \\
&= \{-7,7\}
\end{aligned}
$$

Question 2

Can you find a derivation for the LCL triple

$\vdash_{\mathsf{Sign}^+} [x > 0]\ x := x + 1\ ;\ x := x - 1\ [x \geq 0]$?

No, $x \geq 0$ is not a valid under-approximation

* Exam 10

Can you find a derivation for the LCL triple

$$\vdash_{\mathsf{Sign}^{+}} [x > 0]\; x := x + 1\,;\; x := x - 1\; [x > 0]$$

repairing the domain if necessary?

Special prize

Can you find a derivation for the LCL triple

$$\vdash_{\mathsf{Int}} [\exists k > 0.\ x = 2^k]\ ((\mathsf{even}(x))?\,;\ x := x + 2)^\star\,;\ (x = 3)?\ [\mathsf{false}]$$

repairing the domain if necessary?

Backward repair

PLDI 2022

Abstract Interpretation Repair

Roberto Bruni
University of Pisa, Pisa, Italy
roberto.bruni@unipi.it

Roberto Giacobazzi
University of Verona, Verona, Italy
roberto.giacobazzi@univr.it

Roberta Gori
University of Pisa, Pisa, Italy
roberta.gori@unipi.it

Francesco Ranzato
University of Padova, Padova, Italy
francesco.ranzato@unipd.it

Abstract

Abstract interpretation is a sound-by-construction method for program verification: any erroneous program will raise some alarm. However, the verification of correct programs may yield false-alarms, namely it may be *incomplete*. Ideally, one would like to perform the analysis on the most abstract domain that is precise enough to avoid false-alarms. We show how to exploit a weaker notion of completeness, called *local completeness*, to optimally refine abstract domains and thus enhance the precision of program verification. Our main result establishes necessary and sufficient conditions for the existence of an optimal, locally complete refinement, called *pointed shell*. On top of this, we define two repair strategies to remove all false-alarms along a given abstract computation: the first proceeds forward, along with the concrete computation, while the second moves backward within the abstract computation. Our results pave the way for a novel *modus operandi* for automating program verification that we call Abstract Interpretation Repair (AIR): instead of choosing beforehand the right abstract domain, we can start in any abstract domain and progressively repair its local incompleteness as needed. In this regard, AIR is for abstract interpretation what CEGAR is for abstract model checking.

CCS Concepts: • **Theory of computation** → **Logic and verification**; **Abstraction**; **Semantics and reasoning**; **Program analysis**.

Keywords: abstract interpretation, program analysis, program verification, local completeness, CEGAR.

ACM Reference Format:
Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2022. Abstract Interpretation Repair. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '22), June 13–17, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3519939.3523453

PLDI '22, June 13–17, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9265-5/22/06.
https://doi.org/10.1145/3519939.3523453

1 Introduction

It is widely acknowledged that the chance of formally verifying programs is fundamental to effectively rise the confidence level that the code we use is correct [23]. However, as emerged in the last decades, this approach to program correctness becomes socially acceptable when these proofs are not only rigorous but also explainable, meaning that they have to rely upon a largely recognized proof method which has to be simple and inspectable [22]. As advocated by Vardi [61], checking program correctness "*is a cost that must be justified by the benefits*". The last 50 years have shown an impressive flourishing of formal methods and tools for achieving this ambitious goal [32]. These include, among the others: Certified compilers [42], certified analyzers [39], advanced type checkers [49, 50], sophisticated static analyzers [6, 19, 25] and software model checkers [3, 37].

A high degree of confidence in the correctness of a software system, and of its most critical components, can be obtained when the code is certified by a *sound* and *complete* (viz. precise) static analyzer [14, 25]. Abstract interpretation [17] was introduced with this purpose in mind: simplify the proof of correctness by interpreting the program in a simplified, *abstract*, domain. This provides a general methodology for the design of *sound-by-construction* analysis tools.

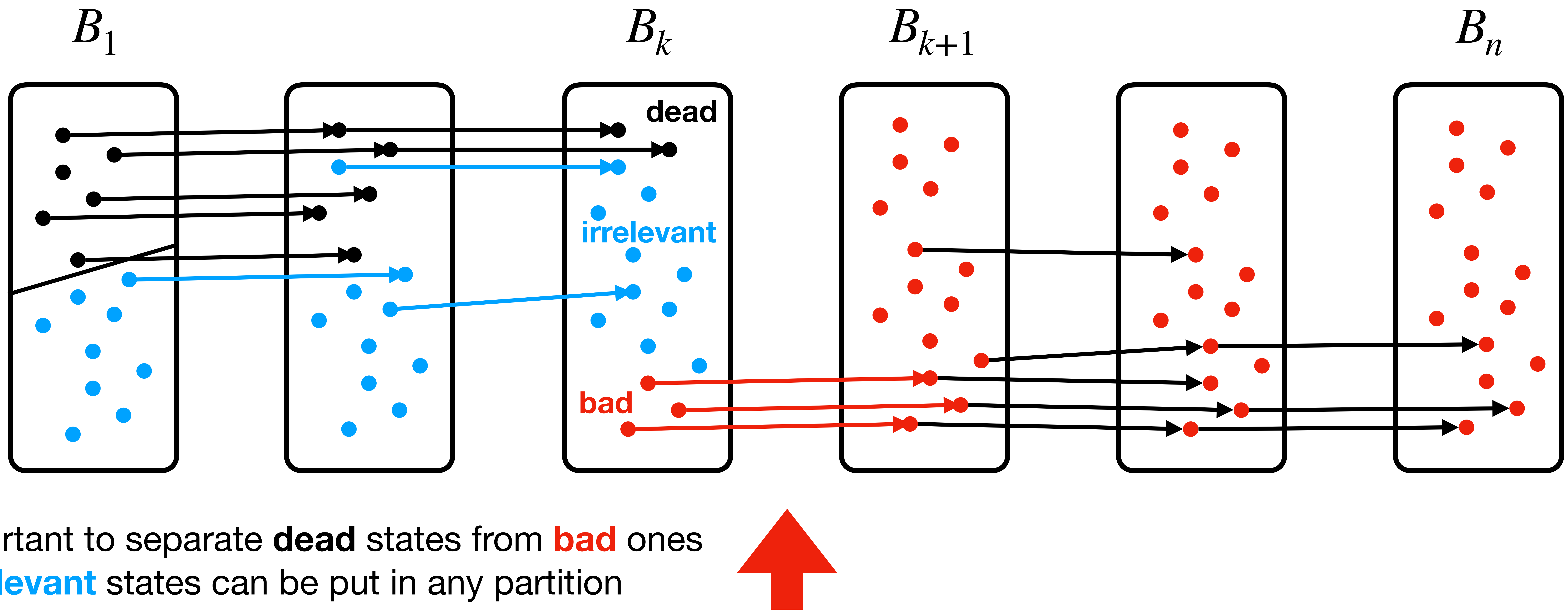
The Problem. The soundness of an abstract interpreter, or program analyzer, means that all true-alarms are caught. However, it is often the case that some false-alarms are reported. Actually, when false-alarms overwhelm true ones, then the program analyzer may become poorly trustworthy. This is a consequence of the approximation inherent in the making of an otherwise undecidable analysis decidable. As all alarm systems, program analysis is credible when few false-alarms are reported, ideally none. The problem we address in this paper is *how to derive the most abstract domain to decide program correctness without raising false-alarms.*

The absence of false-alarms in program analysis is closely related to the property of *completeness in abstract interpretation* [33]. As an illustrative example, consider the program

"we aim to derive the most abstract domain to decide program correctness without raising false-alarms "

CEGAR, again

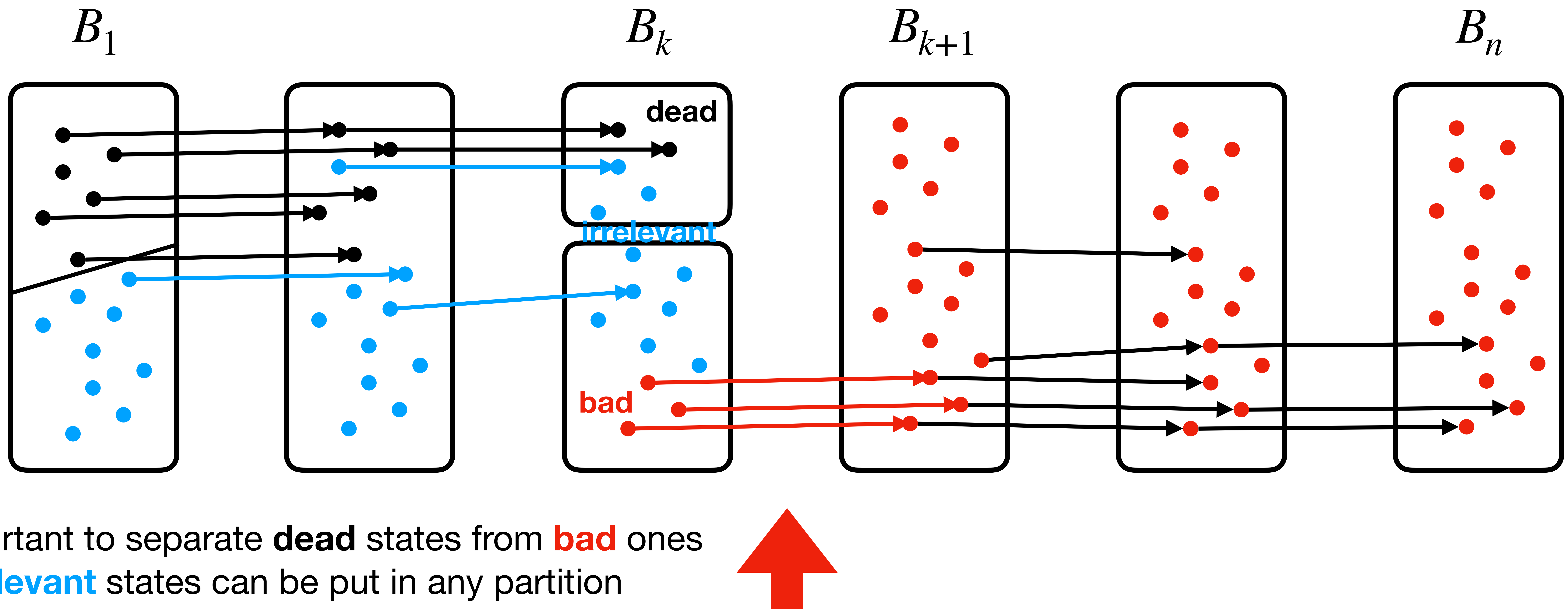
CounterExample Guided Abstraction Refinement:
If the counterexample is spurious, refine the partition to eliminate the abstract path and repeat the analysis



it is important to separate **dead** states from **bad** ones
irrelevant states can be put in any partition

CEGAR, again

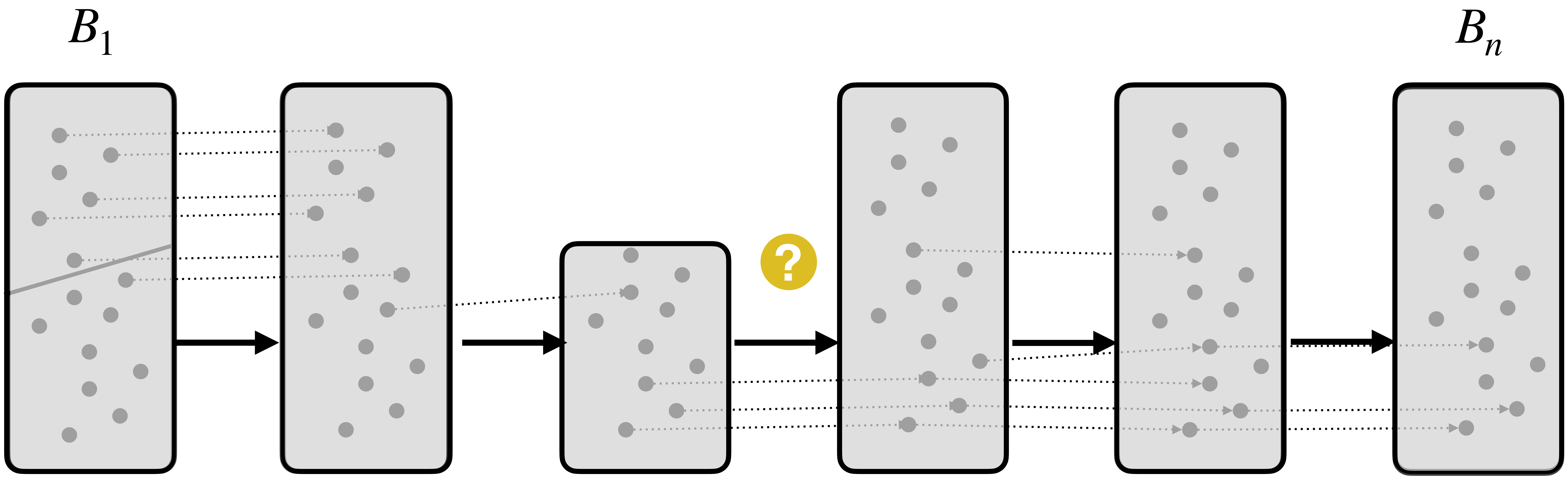
The efficacy depends on the chosen refinement



it is important to separate **dead** states from **bad** ones
irrelevant states can be put in any partition

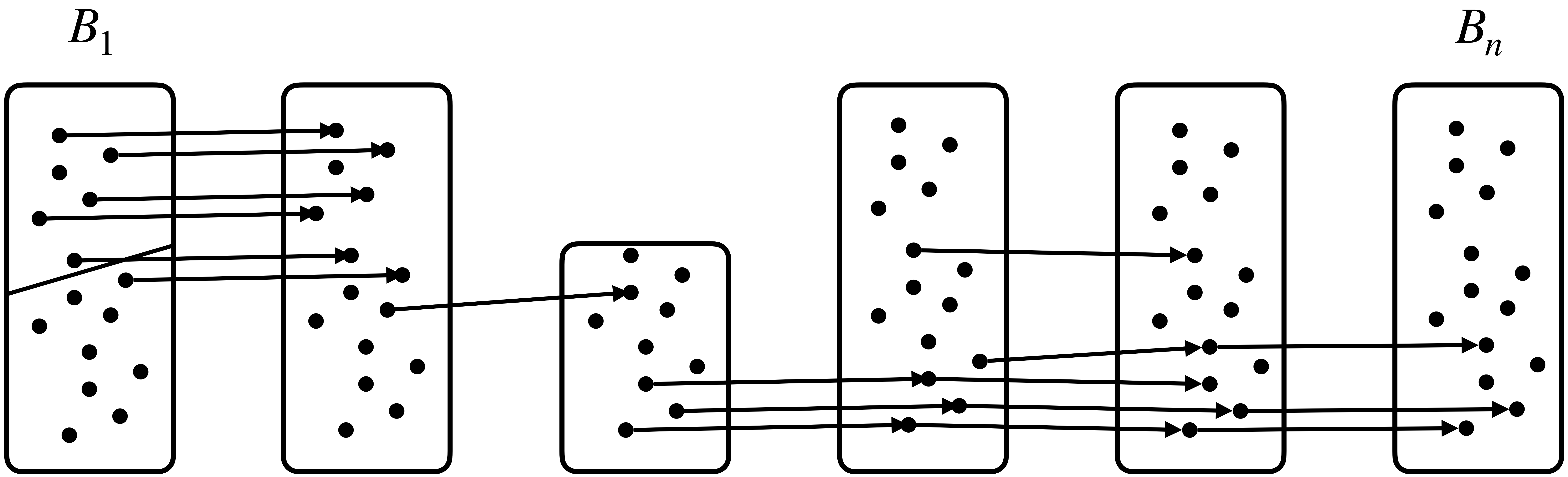
CEGAR, again

Here a slightly different spurious counterexample remains



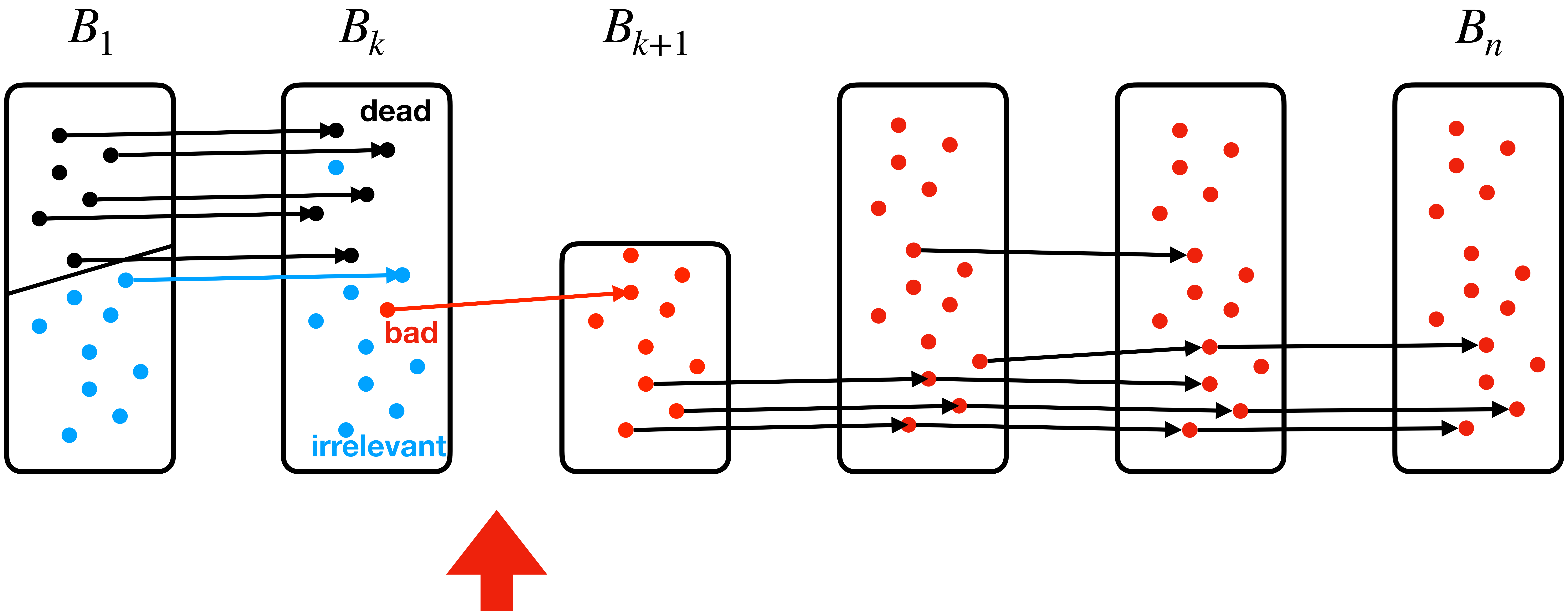
CEGAR, again

Here a slightly different spurious counterexample remains



CEGAR, again

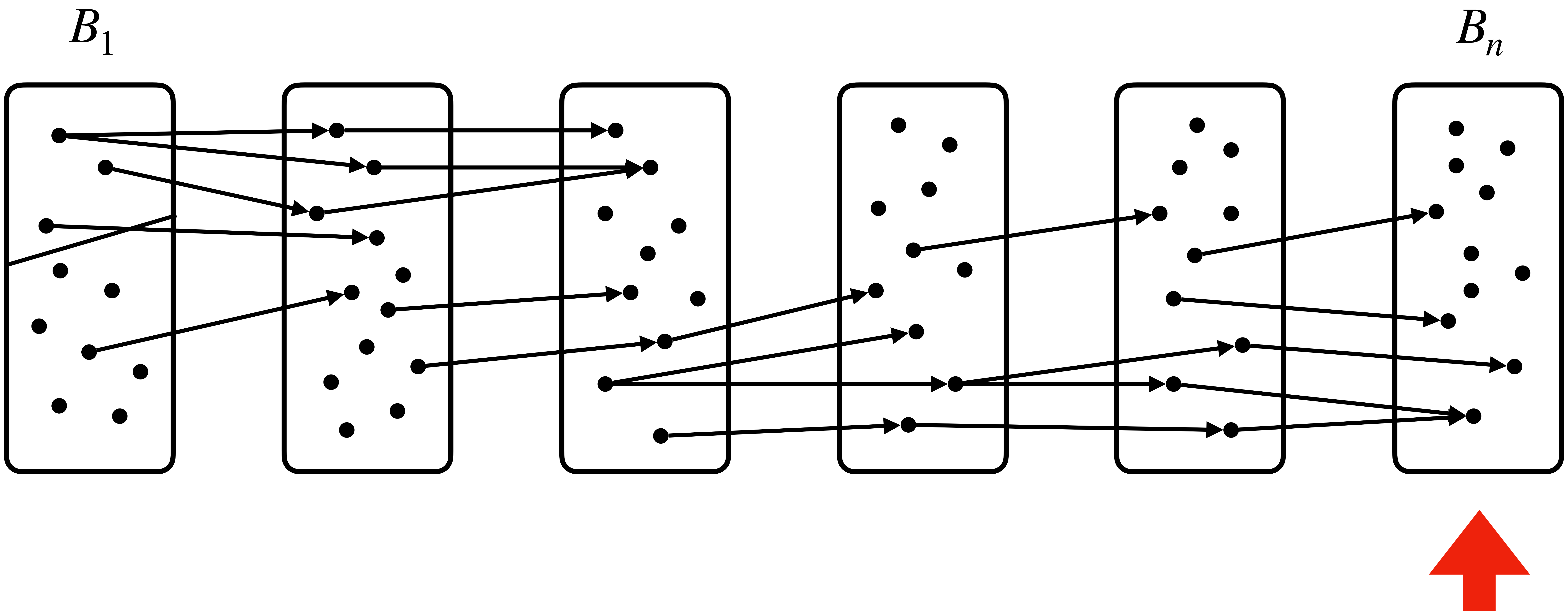
Here a slightly different spurious counterexample remains



CEGAR, backward

What if we start from the end of the trace?

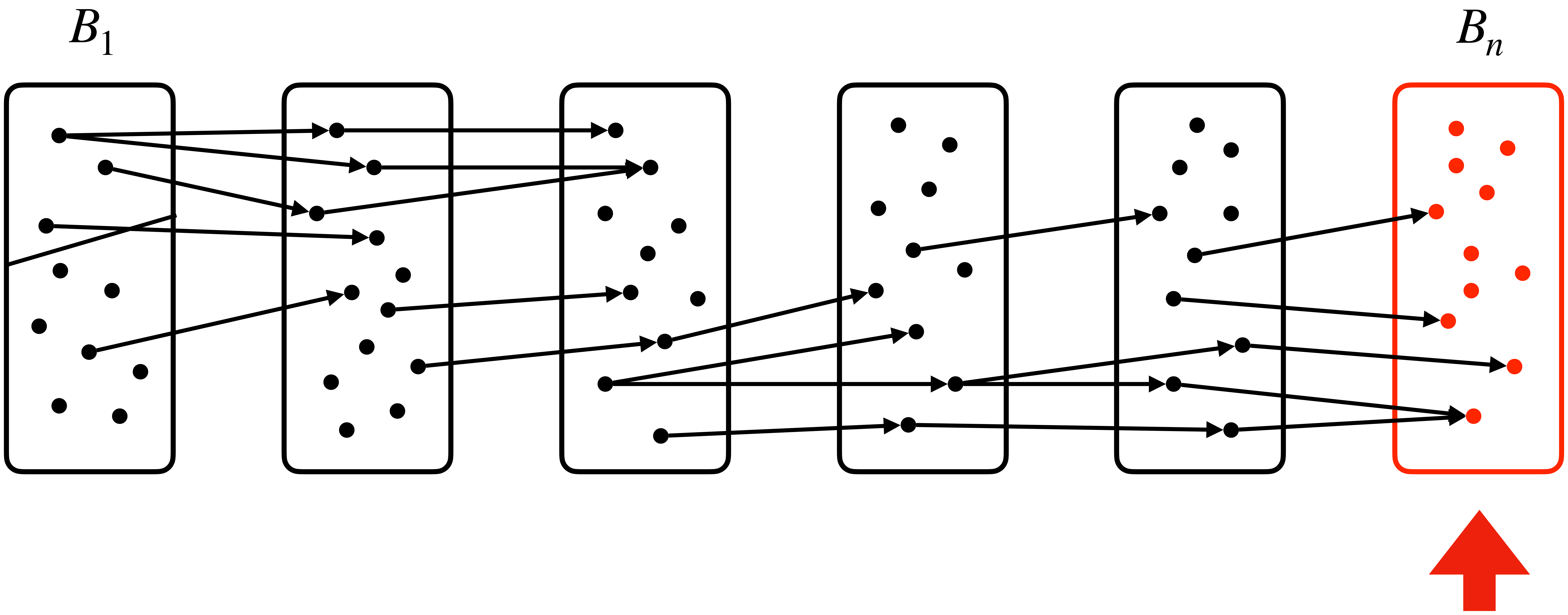
All states in B_n are bad ones!



CEGAR, backward

What if we start from the end of the trace?

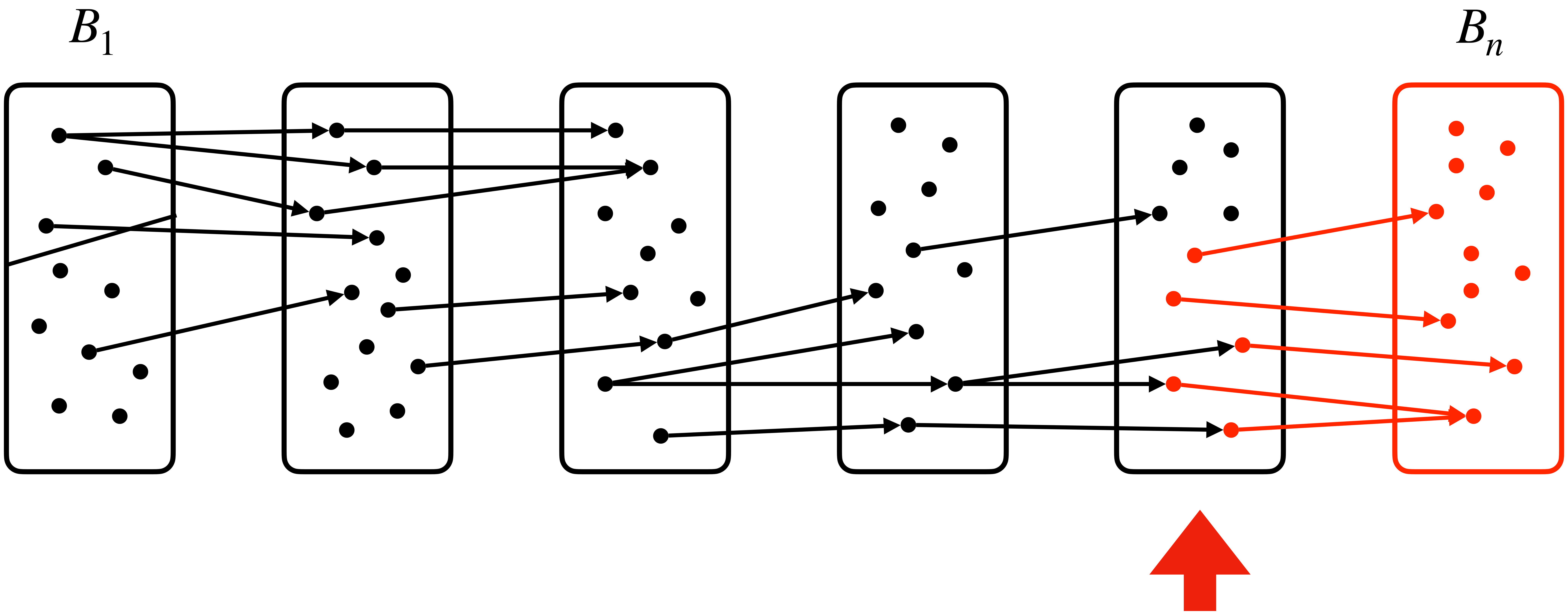
All states in B_n are bad ones!



CEGAR, backward

What if we start from the end of the trace?

All states in B_{n-1} that leads to B_n are bad ones!

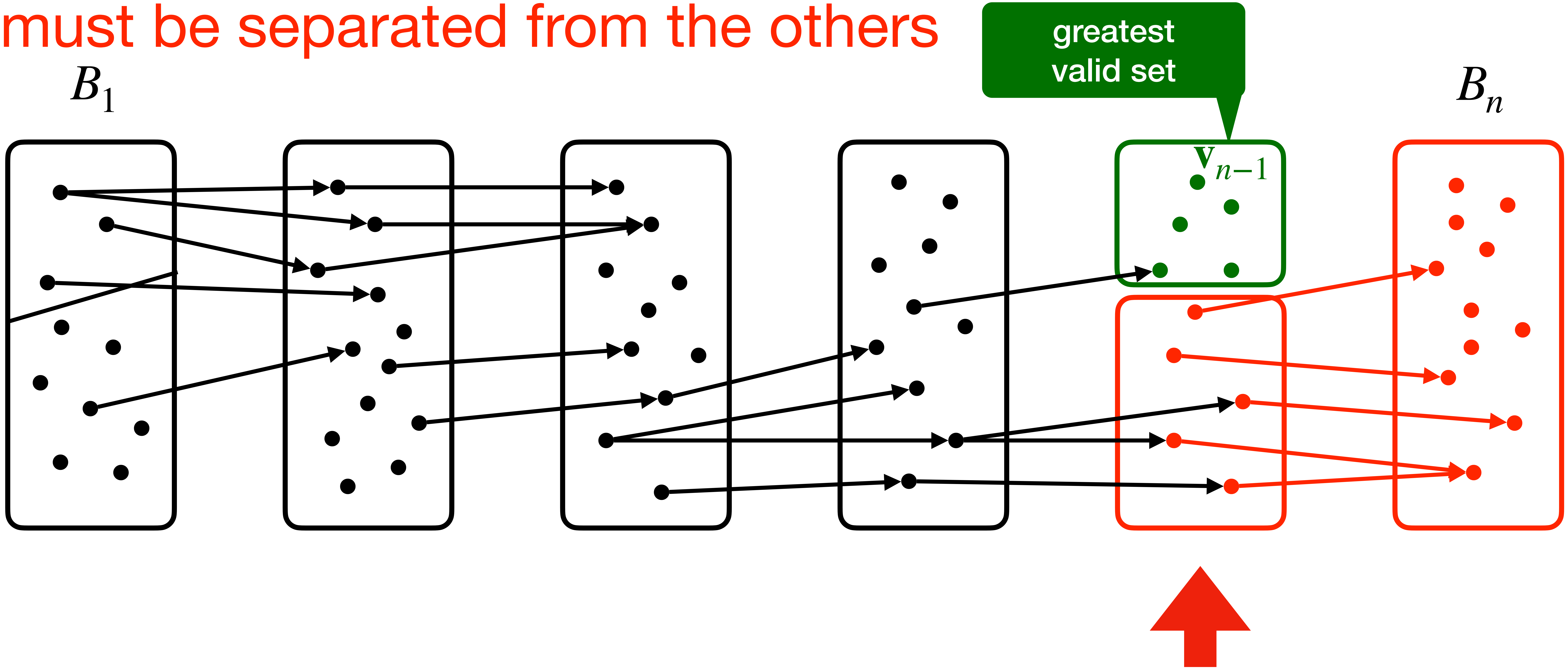


CEGAR, backward

What if we start from the end of the trace?

All states in B_{n-1} that leads to B_n are bad ones!

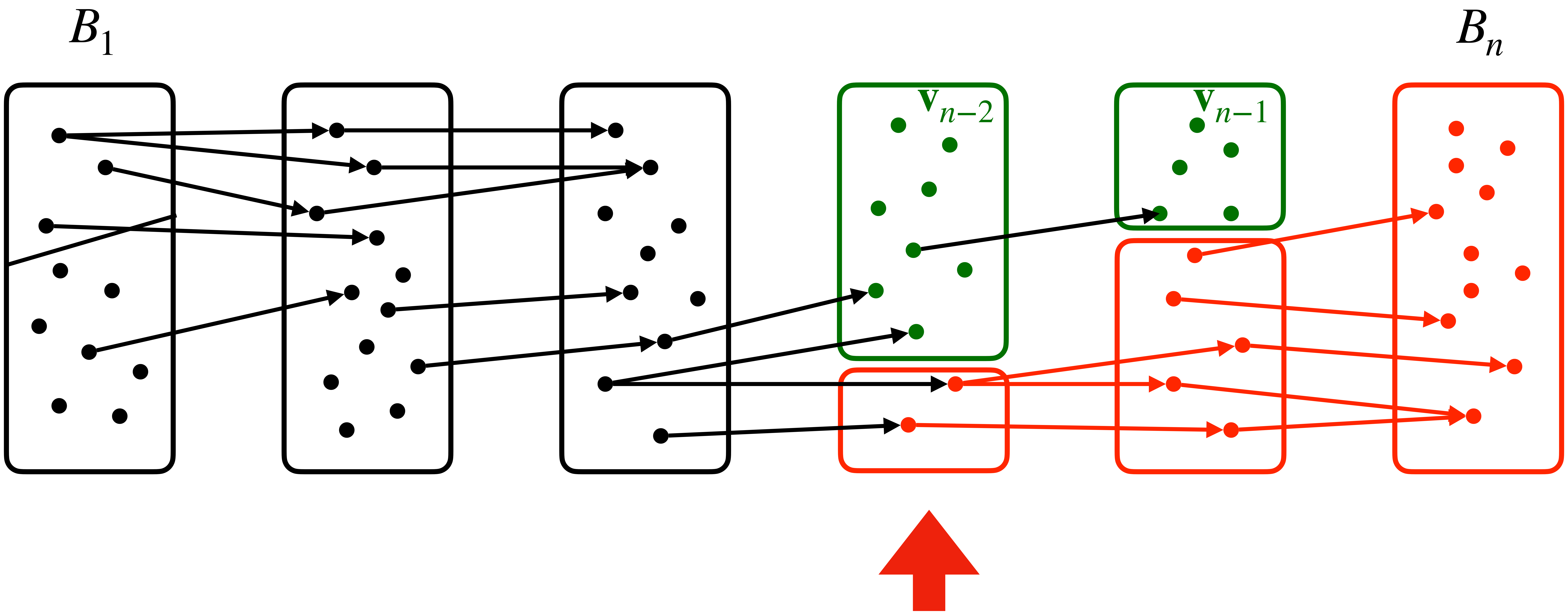
and must be separated from the others



CEGAR, backward

What if we start from the end of the trace?

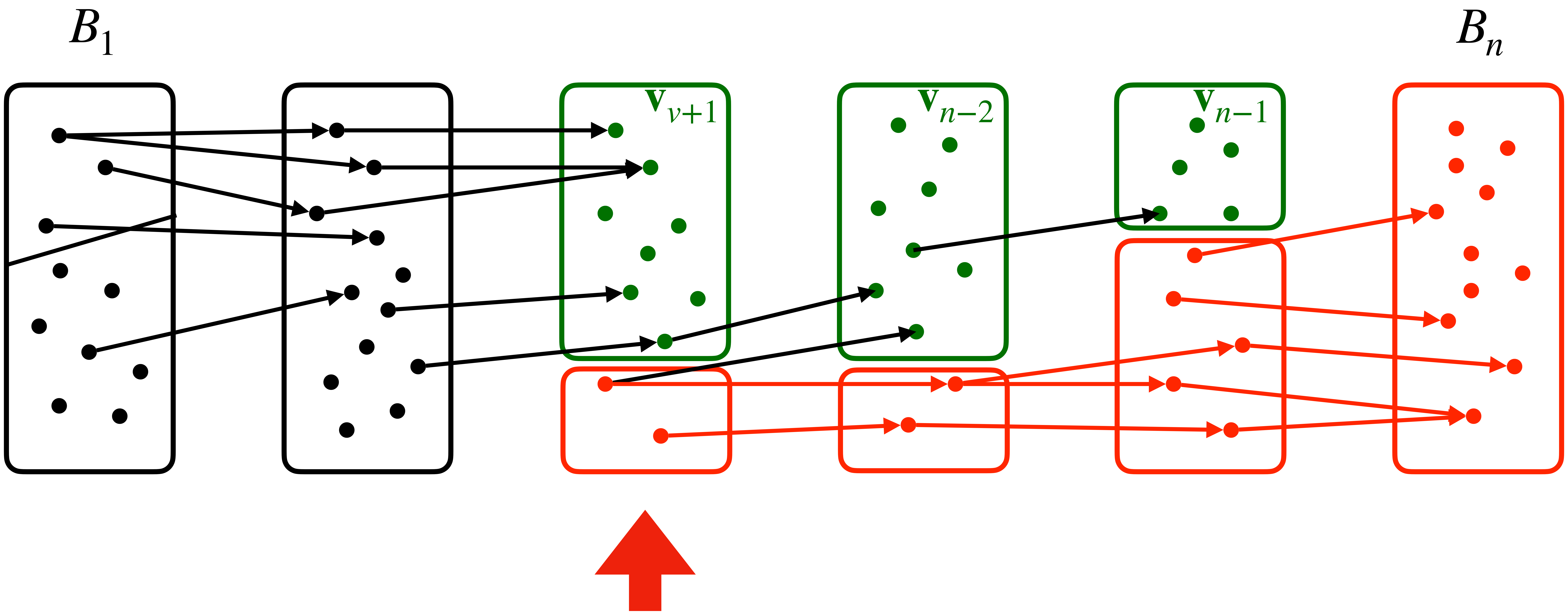
We then iterate the partitioning



CEGAR, backward

What if we start from the end of the trace?

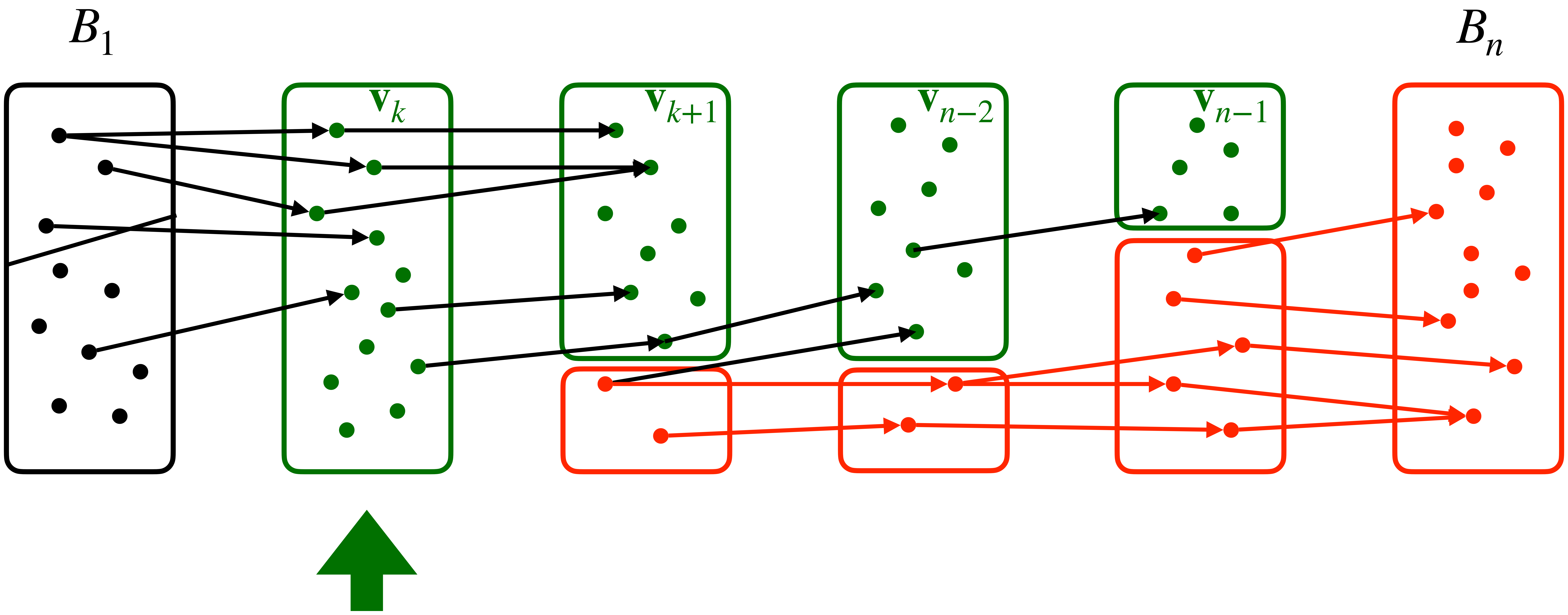
We then iterate the partitioning



CEGAR, backward

What if we start from the end of the trace?

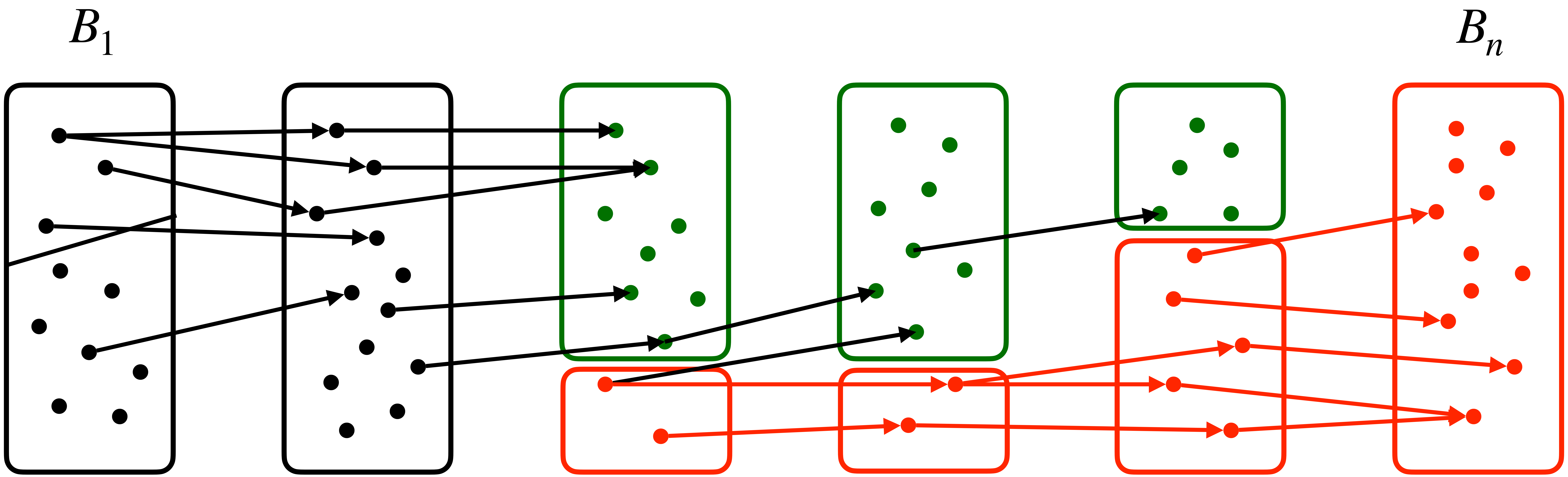
Until necessary



CEGAR, backward

What if we start from the end of the trace?

No more spurious counterexamples from that trace!



When to backward repair

You want to check if $[\![r]\!]P \leq Spec$

You select an abstract domain such that $A(Spec) = Spec$

and run the abstract interpreter, but $[\![r]\!]^{\#}_{A}A(P) \not\leq Spec$

and cannot tell if the abstract interpretation is complete in A

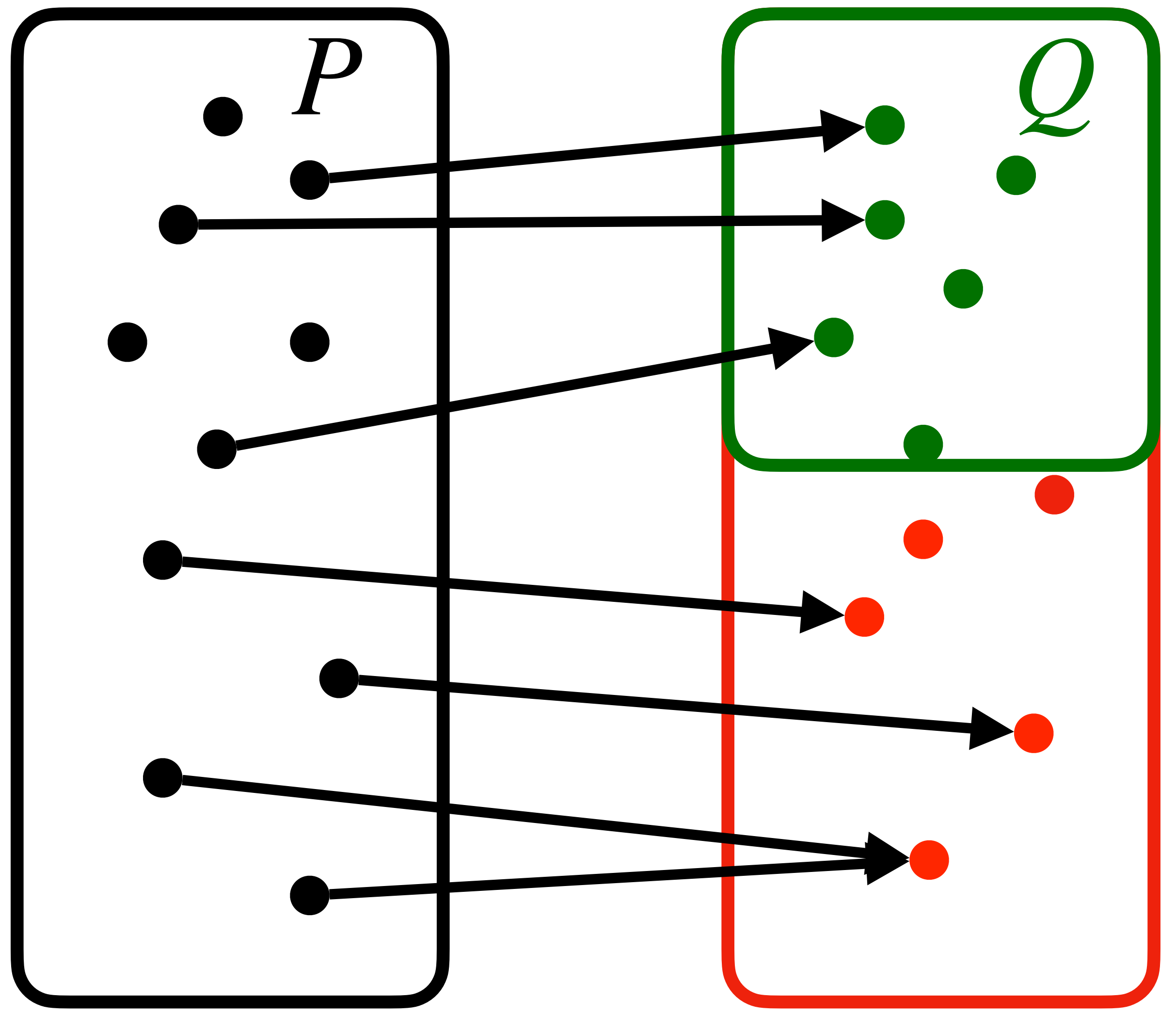
Which errors are spurious? Which ones are due to P?

The aim of backward repair is to find (the most abstract) pointed refinement A_N such that for all $X \subseteq A(P)$ (and thus also for P)

$$[\![r]\!]^{\#}_{A_N}A_N(X) \leq Spec \Leftrightarrow [\![r]\!]X \leq Spec \qquad (\dagger)$$

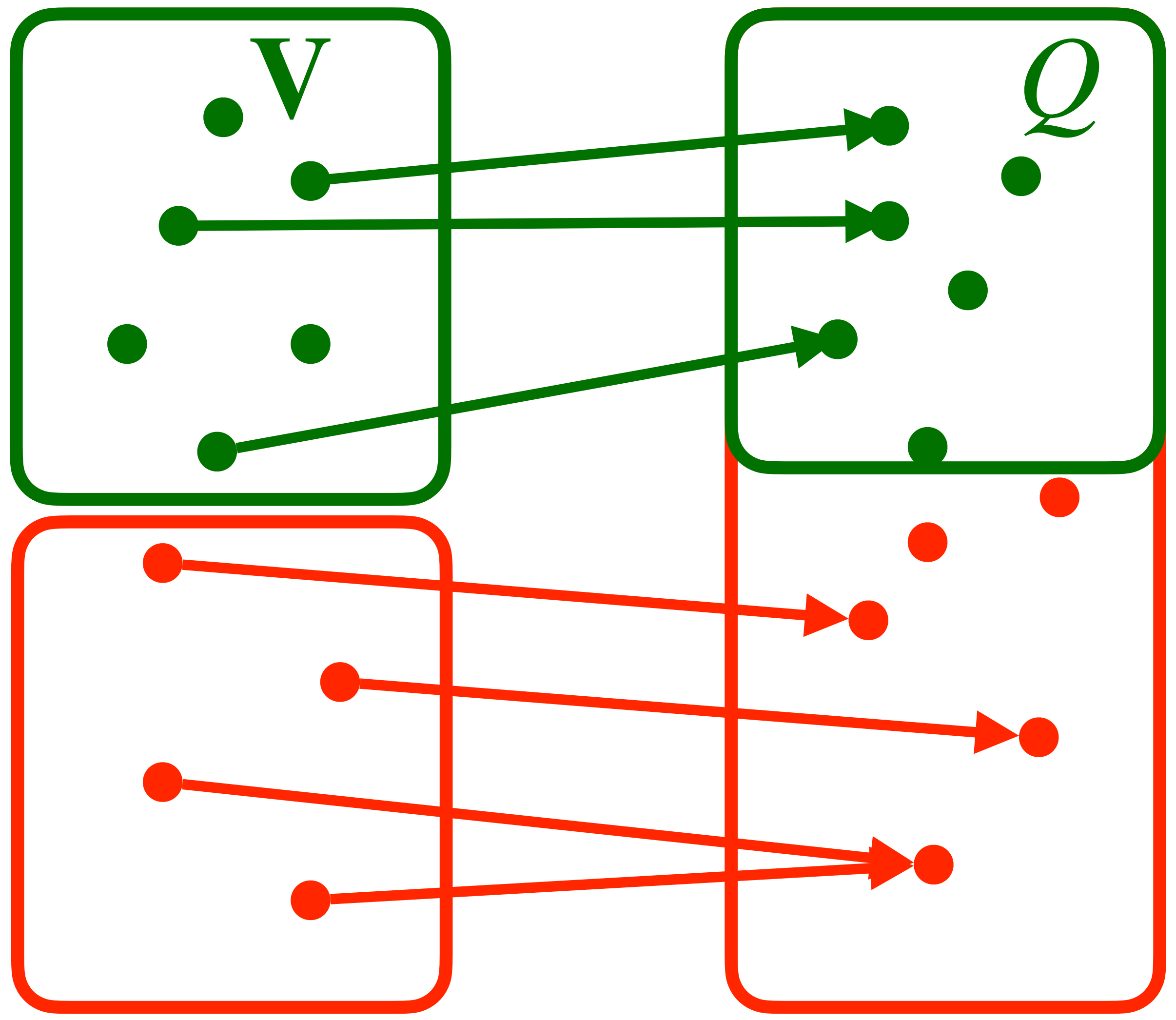
Aim of backward repair

Let $\mathbf{V}\langle P, r, Q\rangle \triangleq P \cap wlp([\![r]\!], Q)$ be the greatest valid input set (it is the largest subset $X \subseteq P$ such that $[\![r]\!]X \subseteq Q$)



Aim of backward repair

Let $\mathbf{V}\langle P, r, Q\rangle \triangleq P \cap wlp(\llbracket r \rrbracket, Q)$ be the greatest valid input set (it is the largest subset $X \subseteq P$ such that $\llbracket r \rrbracket X \subseteq Q$)



Th.

Condition (†) holds iff $\llbracket r \rrbracket^{\#}_{A_N} A_N(\mathbf{V}\langle A(P), r, Spec\rangle) \leq Spec$

which in turn implies $\mathbf{V}\langle A(P), r, Spec\rangle$ being expressible in A_N

Backward repair

additional points, if any

a point in A

greatest valid input

pointed refinement

Th.

$\mathrm{bRepair}_A(\varnothing, A(P), r, Spec)$ returns a pair $\langle V, N \rangle$ such that

$V = \mathbf{V}\langle A(P), r, Spec\rangle \in A_N$ and $[\![r]\!]^{\#}_{A_N} V \leq Spec$

(but may not terminate)

Cor. [*Program (in)correctness*]

If $\langle V, N \rangle = \mathrm{bRepair}_A(\varnothing, A(P), r, Spec)$ then

$[\![r]\!]P \leq Spec \Leftrightarrow P \leq V$

The most convenient case is when $V = A(P)$

Backward repair

```
Function bRepair_A(N, P̂, r, S)
    if (⟦r⟧♯_{A⊞N} P̂ ≤ S) then return ⟨P̂, N⟩;
    switch r do
        case e do                                   // basic expression
            V := V⟨P̂, e, S⟩;  Q := S ∧ ⟦e⟧♯_{A⊞N} P̂;
            return ⟨V, N ∪ {V, Q}⟩;
        case r0; r1 do                              // sequential
            ⟨V1, N1⟩ := bRepair_A(N, ⟦r0⟧♯_{A⊞N} P̂, r1, S);
            ⟨V0, N0⟩ := bRepair_A(N, P̂, r0, V1);
            return ⟨V0, N0 ∪ N1⟩;
        case r0 ⊕ r1 do                             // choice
            ⟨V0, N0⟩ := bRepair_A(N, P̂, r0, S);
            ⟨V1, N1⟩ := bRepair_A(N, P̂, r1, S);
            Q := S ∧ ⟦r⟧♯_{A⊞N} P̂;
            return ⟨V0 ∧ V1, N0 ∪ N1 ∪ {Q}⟩;
        case r0* do                                 // Kleene star
            R̂ := ⟦r0⟧♯_{A⊞N} P̂;
            if (R̂ ≤ P̂) then return inv_A(N, P̂, r0, S);
            else                                    // unroll
                ⟨V1, N1⟩ := bRepair_A(N, P̂ ∨_{A⊞N} R̂, r0*, S);
                return ⟨P̂ ∧ V1, N1⟩
```

```
Function inv_A(N, P̂, r, V1)                        // loop invariants
    do
        V0 := P̂ ∧ V1;  N0 := N ∪ {V0};
        ⟨V1, N1⟩ := bRepair_A(N0, V0, r, V0);
    while (V1 ≠ V0);
    return ⟨V1, N1⟩;
```

Example

$[\![c]\!]\ \top \leq (z = 0)$?

$$
\begin{aligned}
c \triangleq \textbf{do}\ \{\ & z := 0;\ x := y; \\
& \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{ \\
& \qquad x := x + 1;\ z := 1 \\
& \} \\
\}\ \textbf{while}\ & (x \neq y)
\end{aligned}
$$

Example

w=0

$$[\![c]\!]\ \top \leq (z = 0)\ ?$$

$$
\begin{aligned}
\mathrm{c} \triangleq \textbf{do}\ \{\ & z := 0;\ x := y; \\
& \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{ \\
& \qquad x := x + 1;\ z := 1 \\
& \} \\
\}\ \textbf{while}\ & (x \neq y)
\end{aligned}
$$

Example

w=0

z=0 , x=y

$$
\begin{aligned}
c \triangleq \textbf{do}\ \{\ & z := 0;\ x := y; \\
& \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{ \\
& \qquad x := x + 1;\ z := 1 \\
& \} \\
\}\ \textbf{while}\ & (x \neq y)
\end{aligned}
$$

$[\![c]\!]\ \top \leq (z = 0)$?

Example

w=0

$$
\begin{aligned}
c \triangleq \textbf{do}\ \{\ & z := 0;\ x := y; \\
& \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{ \\
& \quad x := x + 1;\ z := 1 \\
& \} \\
\}\ \textbf{while}\ & (x \neq y)
\end{aligned}
$$

z=0 , x=y

z=0 , x=y

$[\![c]\!]\ \top \leq (z = 0)$?

Example

w=0

$$
\begin{aligned}
c \triangleq \textbf{do } \{\, & z := 0;\ x := y; \\
& \quad \textbf{if } (w \neq 0) \textbf{ then } \{ \\
& \qquad x := x + 1;\ z := 1 \\
& \quad \} \\
\}\ \textbf{while } & (x \neq y)
\end{aligned}
$$

z=0 , x=y

z=0 , x=y

z=0 , x=y

$[\![c]\!]\ \top \leq (z = 0)$?

Example

w=0

w≠0

$[\![c]\!]\ \top \leq (z = 0)$?

z=0 , x=y

$$
\begin{aligned}
c \triangleq \textbf{do}\ \{\ & z := 0;\ x := y; \\
& \quad \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{ \\
& \quad\quad x := x + 1;\ z := 1 \\
& \quad \}
\end{aligned}
$$

z=0 , x=y

$$\}\ \textbf{while}\ (x \neq y)$$

z=0 , x=y

Example

w=0

w≠0

$$[\![c]\!]\ \top \le (z = 0)\ ?$$

$$\begin{aligned}
c \triangleq \textbf{do}\ \{\ & z := 0;\ x := y; \\
& \quad \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{ \\
& \qquad x := x + 1;\ z := 1 \\
& \quad \} \\
\}\ \textbf{while}\ & (x \neq y)
\end{aligned}$$

z=0 , x=y

z=0 , x=y

z=0 , x=y

Example

w=0

w≠0

$$c \triangleq \textbf{do}\ \{\ z := 0;\ x := y;$$

z=0 , x=y

$$\quad \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{$$

$$\qquad x := x + 1;\ z := 1$$

$$\quad \}$$

z=1 , x=y+1

$$\}\ \textbf{while}\ (x \neq y)$$

z=0 , x=y

$[\![c]\!]\ \top \leq (z = 0)$?

Example

w=0

w≠0

z=1 , x=y+1

$$c \triangleq \textbf{do}\ \{\ z := 0;\ x := y;$$

z=0 , x=y

$$\textbf{if}\ (w \neq 0)\ \textbf{then}\ \{$$

$$x := x + 1;\ z := 1$$

$$\}$$

z=1 , x=y+1

$$\}\ \textbf{while}\ (x \neq y)$$

z=0 , x=y

$$[\![c]\!]\ \top \leq (z = 0)\ ?$$

Example

w=0

w≠0

z=1 , x=y+1

z=0 , x=y

$$
\begin{aligned}
c \triangleq \textbf{do}\ \{\ & z := 0;\ x := y; \\
& \quad \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{ \\
& \qquad x := x + 1;\ z := 1 \\
& \quad \} \\
\}\ \textbf{while}\ & (x \neq y)
\end{aligned}
$$

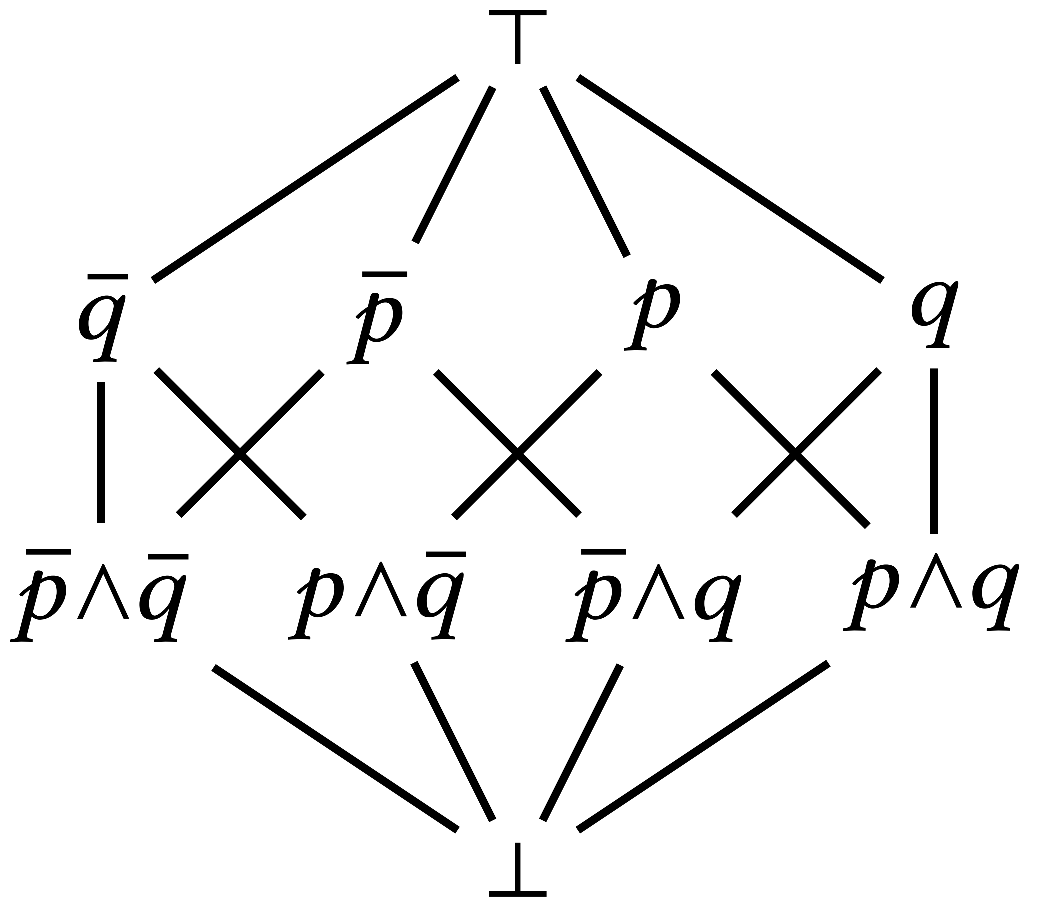
z=1 , x=y+1

z=0 , x=y

$[\![c]\!]\ \top \leq (z = 0)$?

$p \triangleq (z = 0)$

$q \triangleq (x = y)$



Example

w=0

w≠0

z=1 , x=y+1

z=0 , x=y

$$
\begin{aligned}
c \triangleq \textbf{do}\ \{\ & z := 0;\ x := y; \\
& \quad \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{ \\
& \qquad x := x + 1;\ z := 1 \\
& \quad \} \\
\}\ \textbf{while}\ & (x \neq y)
\end{aligned}
$$

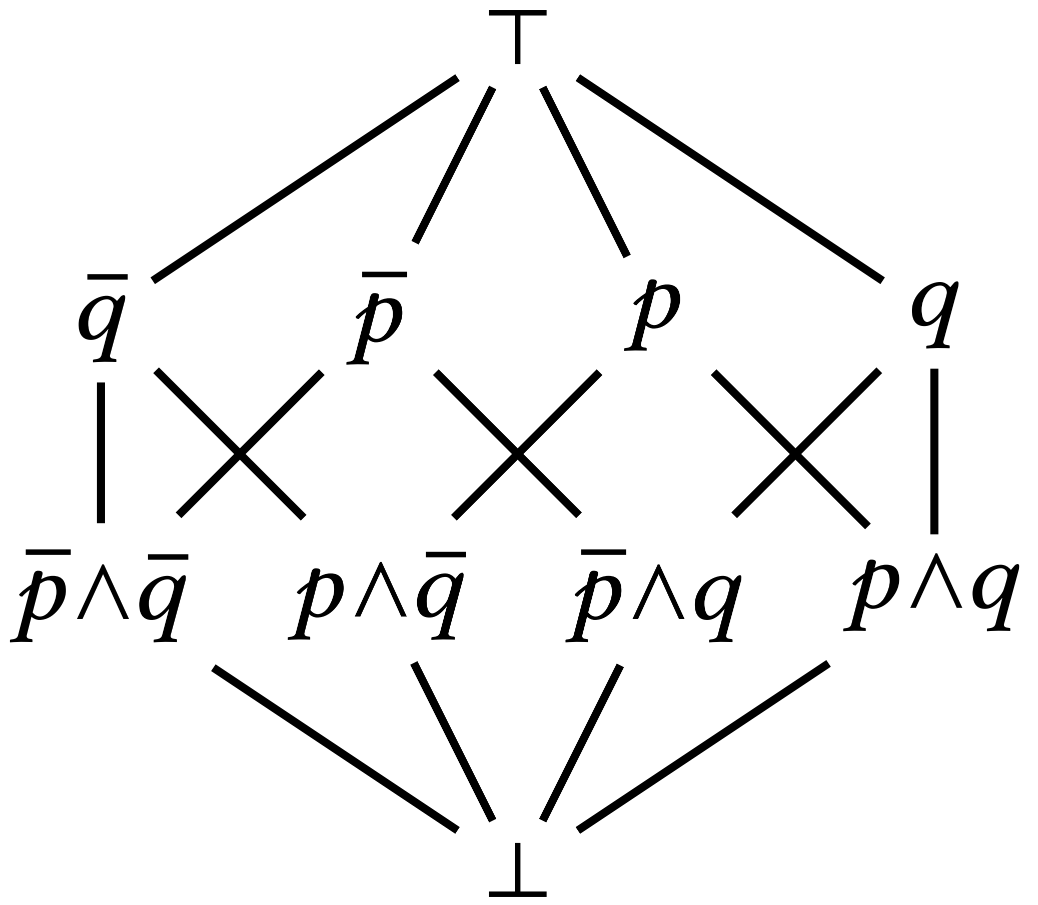
z=1 , x=y+1

z=0 , x=y

$[\![c]\!]\ \top \leq (z = 0)$?

$p \triangleq (z = 0)$

$q \triangleq (x = y)$



$[\![c]\!]^{\#}_{A}\ \top = q \not\leq p$

Example

w=0

w≠0

z=1 , x=y+1

z=0 , x=y

$$\text{c} \triangleq \textbf{do}\ \{\ z := 0;\ x := y;$$
$$\quad \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{$$
$$\qquad x := x + 1;\ z := 1$$
$$\quad \}$$
$$\}\ \textbf{while}\ (x \neq y)$$

z=1 , x=y+1

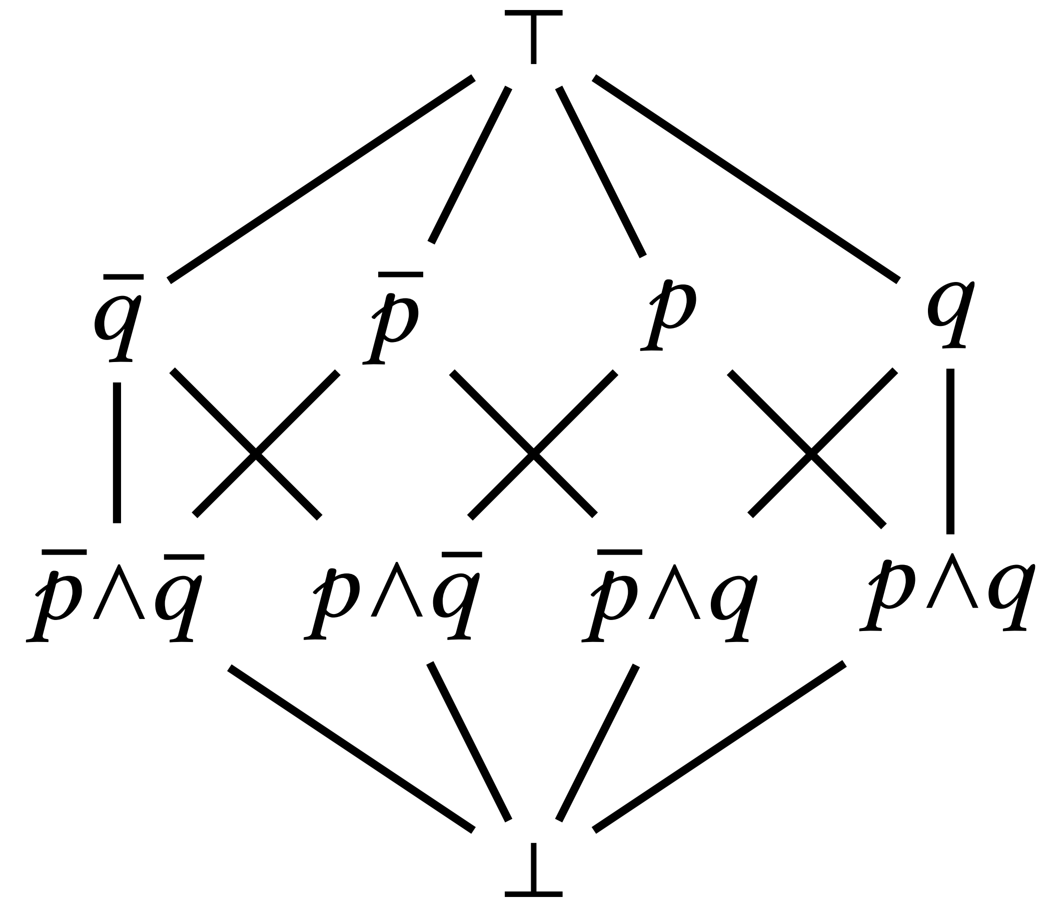
z=0 , x=y

$$\mathrm{bRepair}_A(\varnothing, \top, c, p) = \langle \top, \{q \Rightarrow p\} \rangle$$

$$[\![c]\!]\, \top \leq (z = 0)\ ?$$

$$p \triangleq (z = 0)$$

$$q \triangleq (x = y)$$



$$[\![c]\!]^{\#}_{A}\, \top = q \not\leq p$$

Example

w=0

w≠0

z=1 , x=y+1

z=0 , x=y

$$
\begin{aligned}
c \triangleq \textbf{do}\ \{\ & z := 0;\ x := y; \\
& \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{ \\
& \qquad x := x + 1;\ z := 1 \\
& \}
\end{aligned}
$$

z=1 , x=y+1

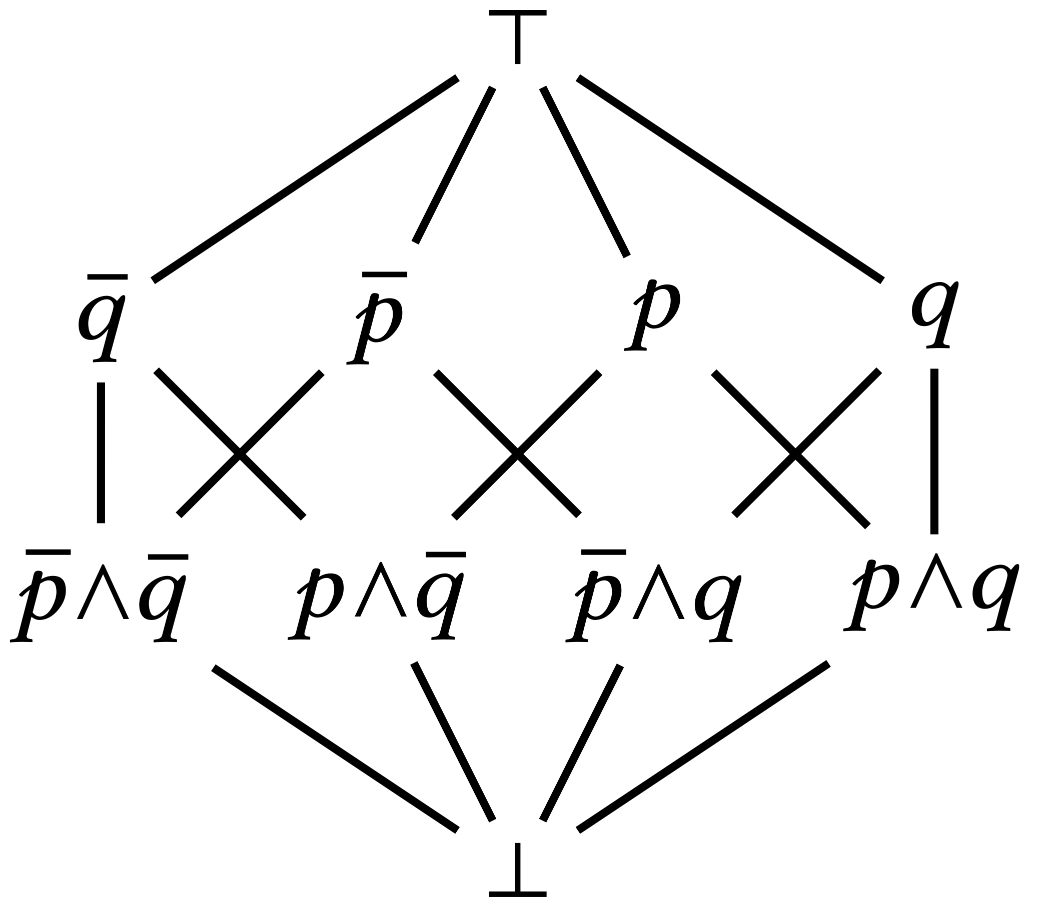
$$\}\ \textbf{while}\ (x \neq y)$$

z=0 , x=y

$$[\![c]\!]\ \top \leq (z = 0)\ ?$$

$$p \triangleq (z = 0)$$

$$q \triangleq (x = y)$$



$$[\![c]\!]^{\#}_{A}\ \top = q \not\leq p$$

$$\text{bRepair}_A(\varnothing, \top, c, p) = \langle \top, \{q \Rightarrow p\}\rangle$$

$$[\![c]\!]^{\#}_{A_{q \Rightarrow p}}\ \top = p \wedge q \leq p$$

Example

w=0

w≠0

z=1 , x=y+1

z=0 , x=y

$$c \triangleq \textbf{do}\ \{\ z := 0;\ x := y;$$
$$\quad \textbf{if}\ (w \neq 0)\ \textbf{then}\ \{$$
$$\qquad x := x + 1;\ z := 1$$
$$\quad \}$$
$$\}\ \textbf{while}\ (x \neq y)$$

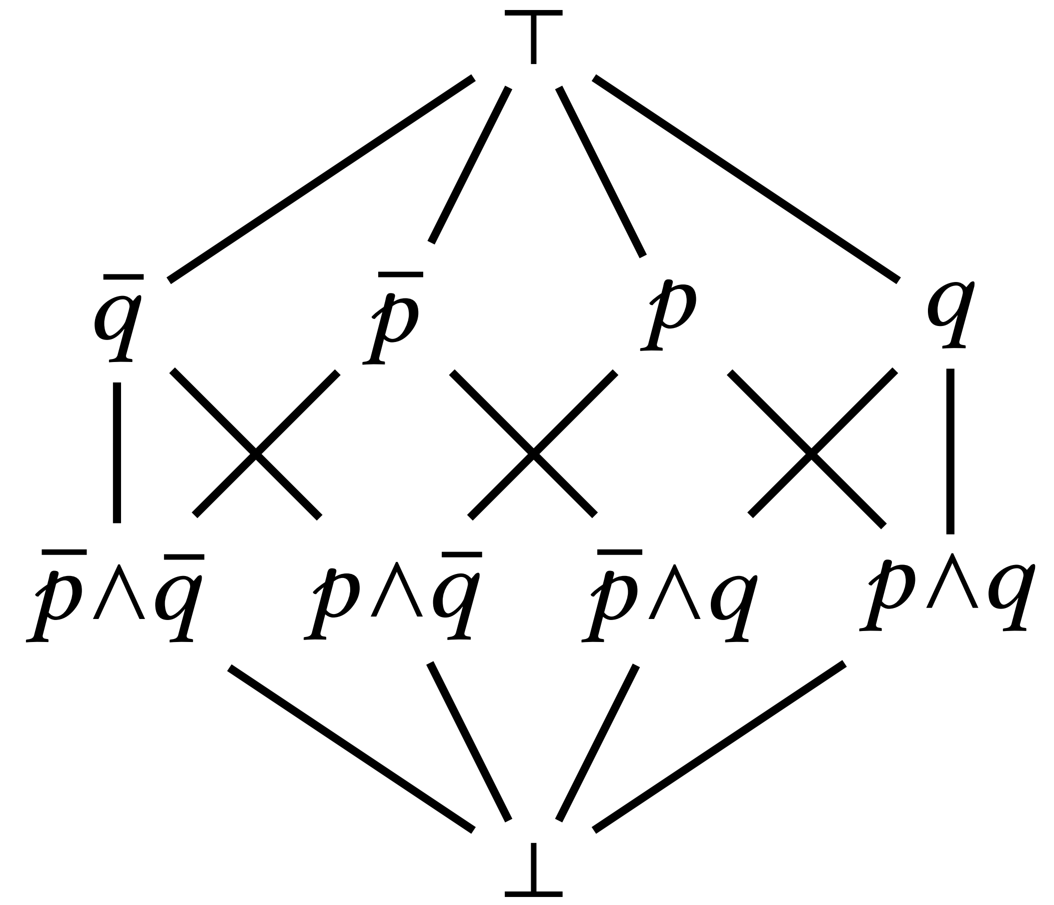
z=1 , x=y+1

z=0 , x=y

$$[\![c]\!]\ \top \leq (z = 0)\ ?$$

$$p \triangleq (z = 0)$$

$$q \triangleq (x = y)$$



$$[\![c]\!]^{\#}_{A}\ \top = q \not\leq p$$

$$\mathsf{bRepair}_A(\varnothing, \top, c, p) = \langle\ \top, \{q \Rightarrow p\}\rangle$$

$$[\![c]\!]^{\#}_{A_{q \Rightarrow p}}\ \top = p \wedge q \leq p$$

The domain refinement used in [4, 5] is the reduced disjunctive completion of A, which is isomorphic to the Boolean abstraction $B \triangleq \langle \wp(\{p\wedge q, p\wedge\bar{q}, \bar{p}\wedge q, \bar{p}\wedge\bar{q}\}), \subseteq\rangle$. The analysis with B leads exactly to the same analysis with $A_1 \triangleq A \boxplus \{p \leftrightarrow q\}$, namely $[\![c]\!]^{\#}_{A_1} \top = p \wedge q$.

Questions

Question 1

Which is the greatest valid input set $\mathbf{V}\langle P, b?, Q\rangle$?

(recall that $\mathbf{V}\langle P, r, Q\rangle \triangleq P \cap wlp([\![r]\!], Q)$)

$$\mathbf{V}\langle P, b?, Q\rangle = P \wedge (Q \vee \neg b)$$

e.g.

$$\mathbf{V}\langle (x \geq 0), x \neq 0?, (x < 5)\rangle = \{0,1,2,3,4\}$$