

A photograph of a building with a green wall and a stone path leading to a doorway. The building is covered in dense green foliage, and a stone path leads to a doorway. The sky is blue, and there are trees in the background.

# Program analysis: from proving correctness to proving incorrectness

**Roberto Bruni, Roberta Gori  
(University of Pisa)  
Lecture #08**

**BISS 2024  
March 11-15, 2024**

# Backward Analysis

# Regular commands

regular  
command

atomic  
command

choice

$r ::=$

$e$

|

$r_1; r_2$

|

$r_1 + r_2$

|

$r^*$

Kleene  
star

$e ::= \text{skip} \mid x := a \mid b? \mid \dots$

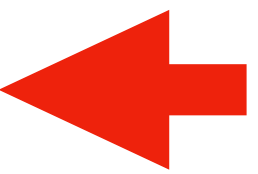
Syntactic sugar

if  $b$  then  $c_1$  else  $c_2 \triangleq (b?; c_1) + (\neg b?; c_2)$



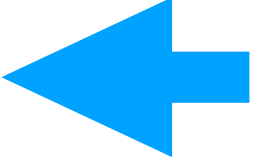

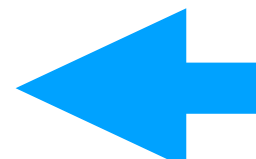
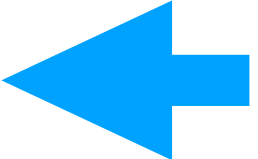
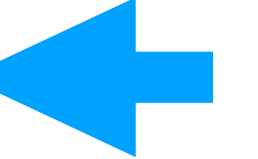
while  $b$  do  $c \triangleq (b?; c)^*; \neg b?$

# Backward analysis

## Forward Analysis

```
int Simple (bool b)
{ int z;
  if (b)
    z := 12;  $z \in [12,12]$ 
  else
    z := -12;  $z \in [-12,-12]$ 
     $z \in [-12,12]$ 
  return 1/z;  Possible
  division by 0
}
```

## Backward Analysis

```
int Simple (bool b)
{ int z;
  if (b) 
     z := 12;   $z \neq 0$ 
  else
     z := -12;   $z \neq 0$ 
  return 1/z;   $z \neq 0$ 
    $z \neq 0$ 
}
```

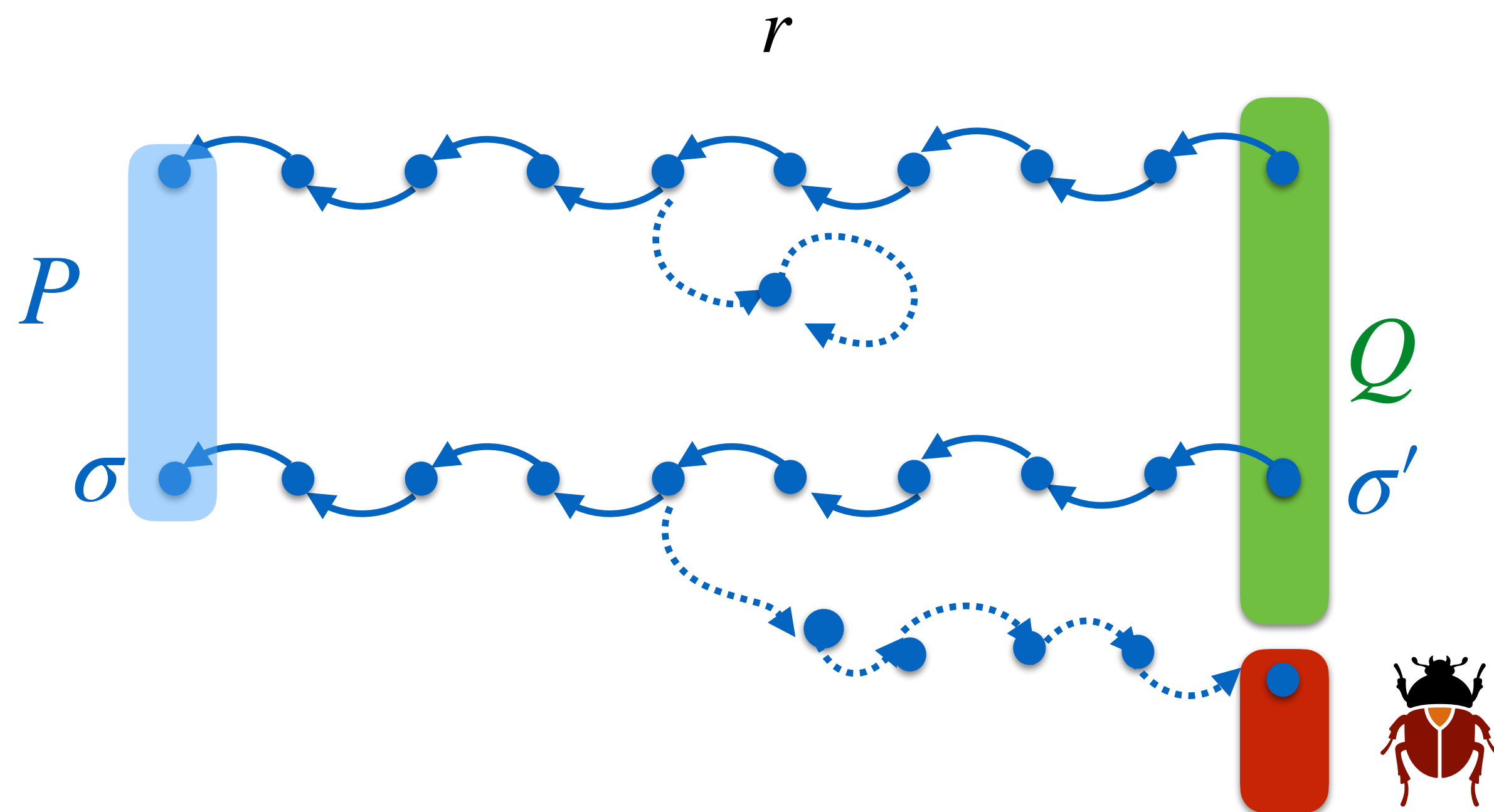
# Backward semantics

$$[[\overleftarrow{r}]]\sigma' \triangleq \{\sigma \mid \sigma' \in [[r]]\sigma\}$$

$$\sigma \in [[\overleftarrow{r}]]\sigma' \Leftrightarrow \sigma' \in [[r]]\sigma$$

Different from WLP!

As before we can extend it to sets



$$[[\overleftarrow{r}]]Q = P$$

**Necessary conditions (NC)**

# VMCAI 2013

## Automatic Inference of Necessary Preconditions

Patrick Cousot<sup>1</sup>, Radhia Cousot<sup>2</sup>, Manuel Fähndrich<sup>3</sup>, and Francesco Logozzo<sup>3</sup>

<sup>1</sup> NYU, ENS, CNRS, INRIA  
pcousot@cims.nyu.edu  
<sup>2</sup> CNRS, ENS, INRIA  
rcousot@ens.fr  
<sup>3</sup> Microsoft Research  
{maf,logozzo}@microsoft.com

**Abstract.** We consider the problem of *automatic* precondition inference. We argue that the common notion of *sufficient* precondition inference (*i.e.*, under which precondition is the program correct?) imposes too large a burden on callers, and hence it is unfit for automatic program analysis. Therefore, we define the problem of *necessary* precondition inference (*i.e.*, under which precondition, if violated, will the program *always* be incorrect?). We designed and implemented several new abstract interpretation-based analyses to infer atomic, disjunctive, universally and existentially quantified necessary preconditions.

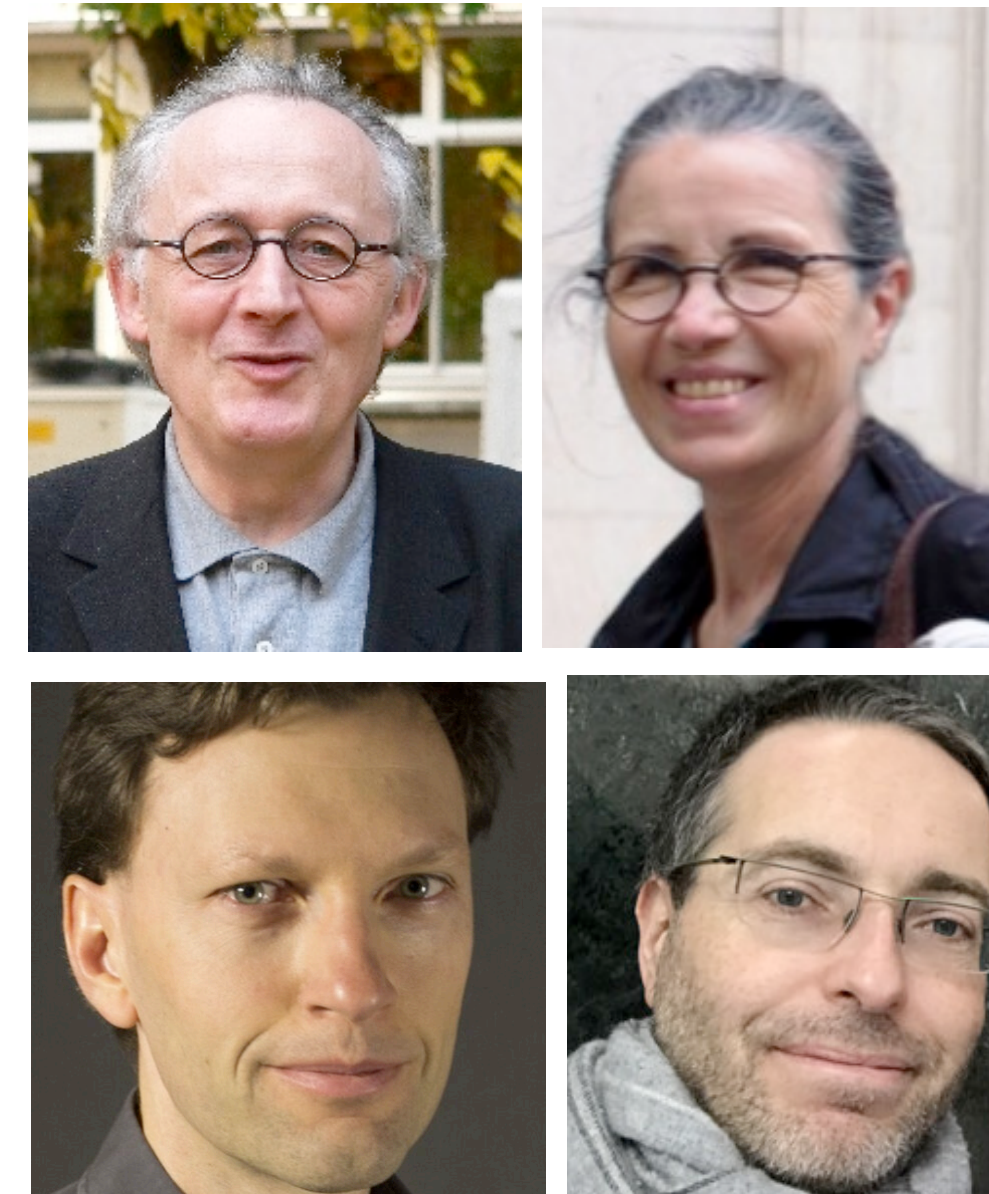
We experimentally validated the analyses on large scale industrial code. For unannotated code, the inference algorithms find necessary preconditions for almost 64% of methods which contained warnings. In 27% of these cases the inferred preconditions were also *sufficient*, meaning all warnings within the method body disappeared. For annotated code, the inference algorithms find necessary preconditions for over 68% of methods with warnings. In almost 50% of these cases the preconditions were also sufficient. Overall, the precision improvement obtained by precondition inference (counted as the additional number of methods with no warnings) ranged between 9% and 21%.

### 1 Introduction

Design by Contract [28] is a programming methodology which systematically requires the programmer to provide the preconditions, postconditions and object invariants (collectively called contracts) at design time. Contracts allow automatic generation of documentation, amplify the testing process, and naturally enable assume/guarantee reasoning for divide and conquer static program analysis and verification. In the real world, relatively few methods have contracts that are sufficient to prove the method correct. Typically, the precondition of a method is weaker than necessary, resulting in unproven assertions within the method, but making it easier to prove the precondition at call-sites. Inference has been advocated as the holy grail to solve this problem.

In this paper we focus on the problem of computing *necessary preconditions* which are *inevitable checks* from within the method that are hoisted to the

“Under which **precondition**, if violated, will the **program** always be **incorrect**?”



# Necessary (pre)conditions

**Sufficient Precondition:** if it holds, the code is correct

**Necessary Precondition:** if it does not hold the code is never correct

But **Sufficient preconditions** impose too large a burden to a callers!!



# Necessary preconditions

```
int Example2 (object[] a)
{ for (int i=0; i<=a.length; i++)

    { a[i]=f(a[i]);

      if ( nondet() )
        return;
    }
}
```

Sufficient precondition: *false*

The function may fail  
So eliminate all runs!

Necessary precondition:  $0 < a.length$

If  $0 == a.length$  then it will always fail!

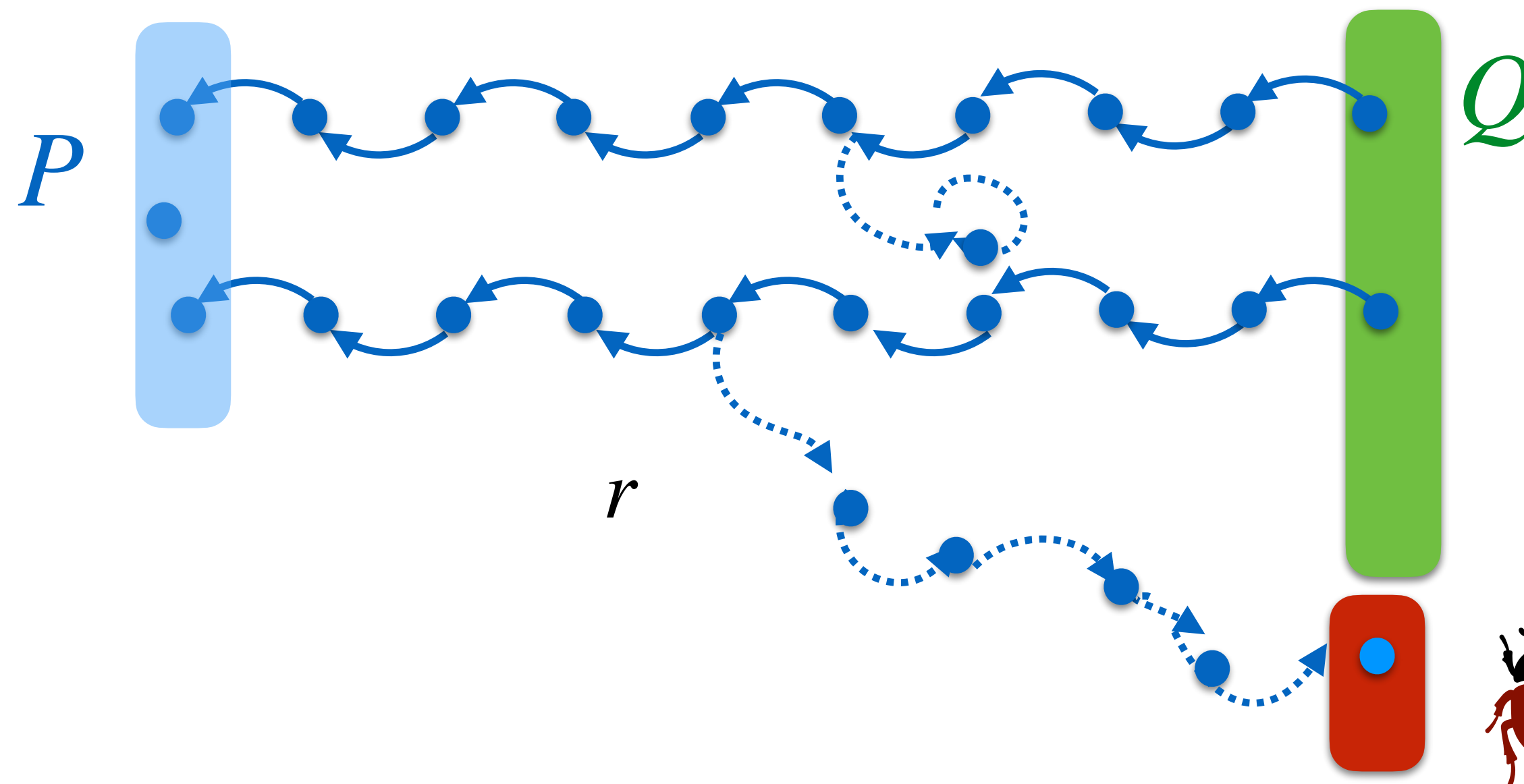
# Necessary preconditions

The idea of NC is to prevent the invocation of the function with arguments that **will inevitably lead to some error**

Given  $Q$  the set of good final states, the NC triple

$$(P) r (Q)$$

means that any state  $\sigma \in P$  may admit at least one non-erroneous execution of  $r$ .



$$[[\overleftarrow{r}]Q \subseteq P$$

It is an over-approximation!

# The taxonomy

	Forward	Backward
Over	{HL} $\llbracket r \rrbracket P \subseteq Q$	(NC) $\llbracket \overleftarrow{r} \rrbracket Q \subseteq P$
Under	[IL] $\llbracket r \rrbracket P \supseteq Q$	<del>⟨SIL⟩?</del> $\llbracket \overleftarrow{r} \rrbracket Q \supseteq P$

# **Sufficient incorrectness logic (SIL)**

# Ongoing work

## Sufficient Incorrectness Logic: SIL and Separation SIL

FLAVIO ASCARI, Università di Pisa, Italy

ROBERTO BRUNI, Università di Pisa, Italy

ROBERTA GORI, Università di Pisa, Italy

FRANCESCO LOGOZZO, Meta Platforms, Inc., USA

Sound over-approximation methods have been proved effective for guaranteeing the absence of errors, but inevitably they produce false alarms that can hamper the programmers. Conversely, under-approximation methods are aimed at bug finding and are free from false alarms. We introduce Sufficient Incorrectness Logic (SIL), a new under-approximating, triple-based program logic to reason about program errors. SIL is designed to set apart the initial states leading to errors. We prove that SIL is correct and complete for a minimal set of rules, and we study additional rules that can facilitate program analyses. We formally compare SIL to existing triple-based program logics. Incorrectness Logic and SIL both perform under-approximations, but while the former exposes only true errors, the latter locates the set of initial states that lead to such errors. Hoare Logic performs over-approximations and as such cannot capture the set of initial states leading to errors in *non-deterministic* programs – for deterministic and terminating programs, Hoare Logic and SIL coincide. Finally, we instantiate SIL with Separation Logic formulae (Separation SIL) to handle pointers and dynamic allocation and we prove its correctness and, for loop-free programs, also its completeness. We argue that in some cases Separation SIL can yield more succinct postconditions and provide stronger guarantees than Incorrectness Separation Logic and can support effective backward reasoning.

CCS Concepts: • **Theory of computation** → **Logic and verification**; *Proof theory*; *Hoare logic*; **Separation logic**; *Programming logic*.

Additional Key Words and Phrases: Sufficient Incorrectness Logic, Incorrectness Logic, Necessary Conditions, Outcome Logic

### 1 INTRODUCTION

Formal methods aim to provide tools for reasoning and establishing program guarantees. Historically, research in formal reasoning progressed from manual correctness proofs to effective, automatic methods that improve program reliability and security. In the late 60s, Floyd [1967] and Hoare [1969] independently introduced formal systems to reason about programs. In the 70s/early 80s, the focus was on mechanization, with the introduction of numerous techniques such as predicate transformers [Dijkstra 1975], Abstract Interpretation [Cousot and Cousot 1977], model checking [Clarke and Emerson 1981], type inference [Damas and Milner 1982] and mechanized program proofs [Coquand and Huet 1985]. Those seminal works, in conjunction with the development of automatic and semi-automatic theorem provers (e.g., [de Moura 2007]) brought impressive wins in proving program correctness of real-world applications. For instance, the Astree abstract interpreter automatically proves the absence of runtime errors in millions of lines of safety-critical C [Blanchet et al. 2003], the SLAM model checker was used to check Windows drivers [Ball and Rajamani 2001], CompCert is a certified C compiler developed in Coq [Leroy 2009], and VCC uses the calculus of weakest precondition to verify safety properties of annotated Concurrent C programs [Cohen et al. 2009].

Despite the aforementioned successes, effective program correctness methods struggle to reach mainstream adoption. As program correctness is undecidable, all those methods *over-approximate* programs behaviours. Over-approximation guarantees soundness (if the program is proved to be

Authors' addresses: Flavio Ascari, Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127, Pisa, Italy, flavio.ascari@phd.unipi.it; Roberto Bruni, Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127, Pisa, Italy, roberto.bruni@unipi.it; Roberta Gori, Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127, Pisa, Italy, roberta.gori@unipi.it; Francesco Logozzo, Meta Platforms, Inc., USA, logozzo@meta.com.

“SIL can characterises the source of errors”



# Sufficient Incorrectness Logic (SIL)

Given  $Q$  a specification of the possible errors

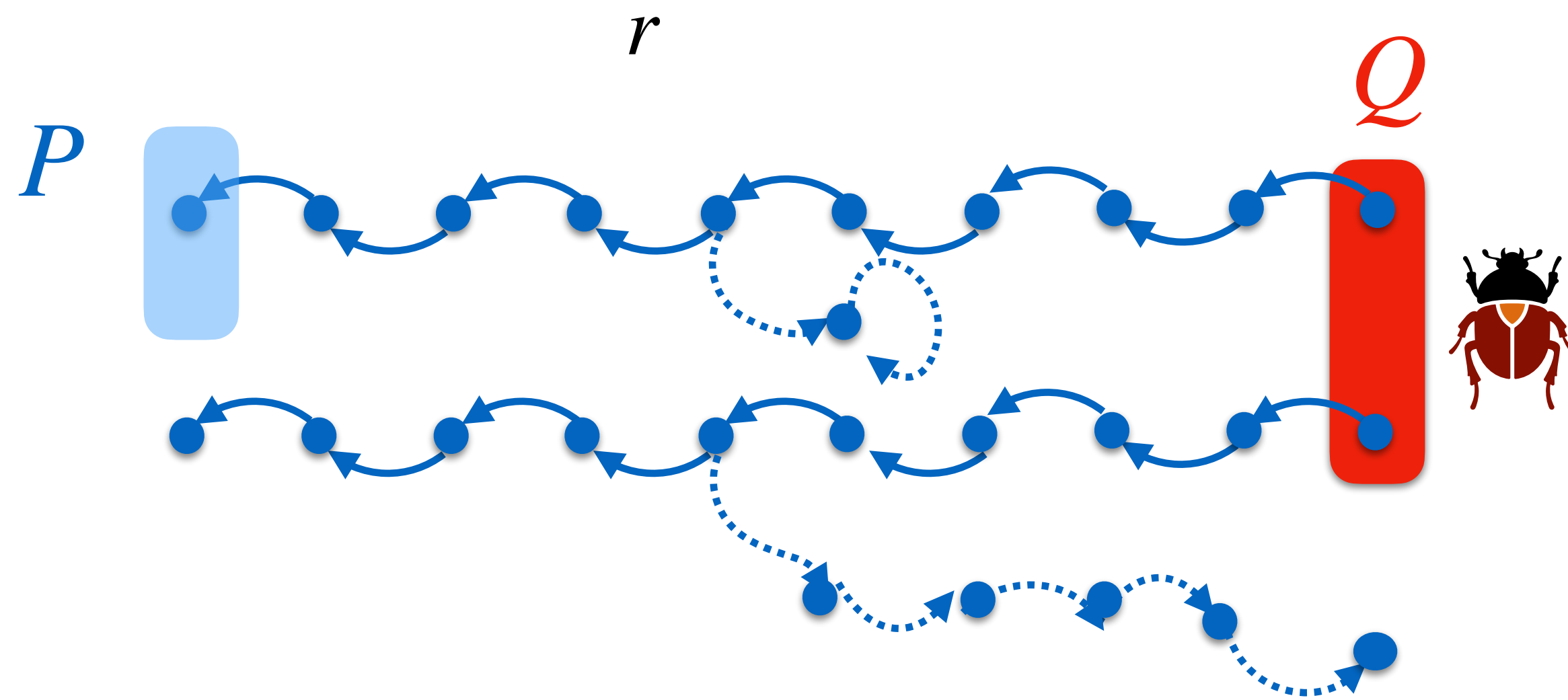
$\llbracket P \rrbracket c \llbracket Q \rrbracket$  is valid when

It is an under-approximation!

$$\llbracket \overleftarrow{r} \rrbracket Q \supseteq P$$

means

$$\forall \sigma \in P \exists \sigma' \in Q . \sigma' \in \llbracket r \rrbracket \sigma$$



An **under-approximating** logic designed to devise the initial states leading to errors

# Bug reporting

Which errors should a tool report to programmers?

We do not want false positives but for the others?

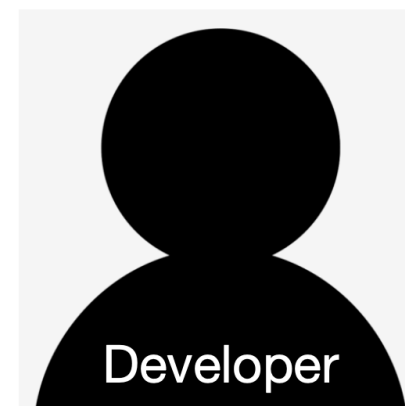
Should the tool report all of them?

```
int foo ( int * x)  
{ *x=32 }
```

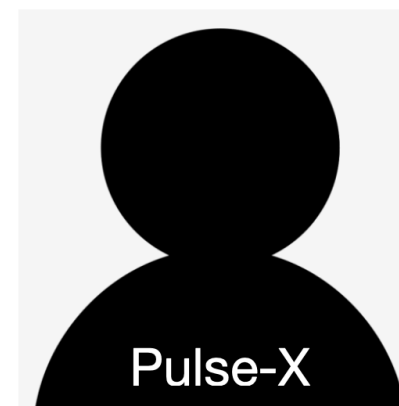
Pulse (based on IL) would find

```
[x=null] foo(x) [er: x=null]
```

## Should the tool report this?



“But I never call foo with null!”



“Which bugs shall I report then?”

# Solution: manifest errors

An error is **manifest** if it occurs **independently of the context** and is therefore particularly interesting to point out to programmers

Manifest errors cannot be characterised with IL

But they can be easily characterised with SIL

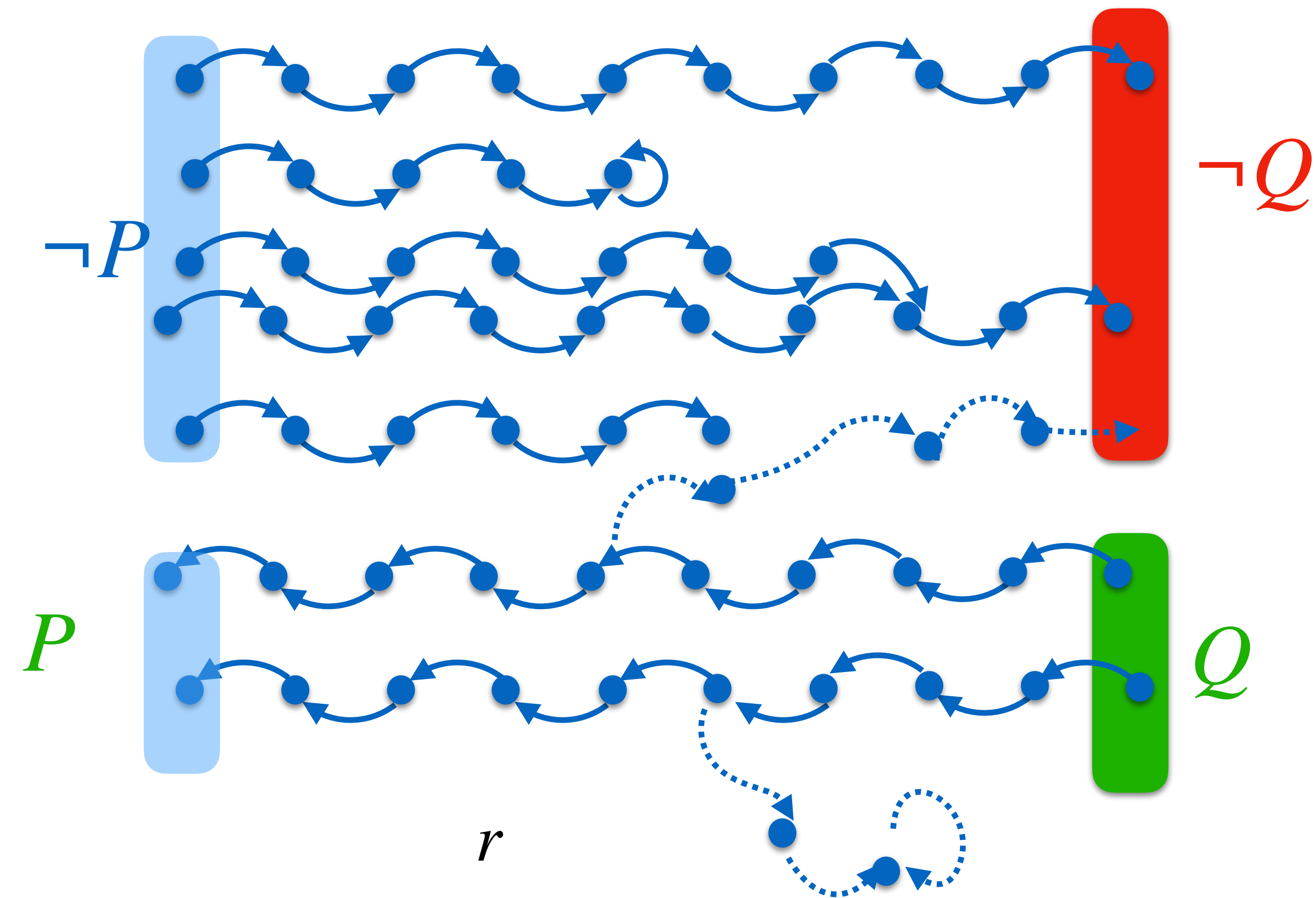
$\langle\langle true \rangle\rangle r \langle\langle Q \rangle\rangle$  is valid  $\Leftrightarrow Q$  is a **manifest error**



# Compare logics along the approximation axis

	Forward	Backward
Over	$\{\text{HL}\} \quad \llbracket r \rrbracket P \subseteq Q$	$(\text{NC}) \quad \llbracket \overleftarrow{r} \rrbracket Q \subseteq P$
Under	$\llbracket \text{IL} \rrbracket \quad \llbracket r \rrbracket P \supseteq Q$	$\langle\langle \text{SIL} \rangle\rangle \quad \llbracket \overleftarrow{r} \rrbracket Q \supseteq P$

# NC vs HL



$$\{\neg P\} r \{\neg Q\} \iff (P) r (Q)$$

$$[[r]]\neg P \subseteq \neg Q \iff [[\overleftarrow{r}]]Q \subseteq P$$

# Compare logics along the approximation axis

	Forward	Backward
Over	{HL} $\llbracket r \rrbracket P \subseteq Q$	(NC) $\llbracket \overleftarrow{r} \rrbracket Q \subseteq P$
Under	$\llbracket r \rrbracket P \supseteq Q$	$\llbracket \overleftarrow{r} \rrbracket Q \supseteq P$

# SIL vs IL

$c_{42}$ :

```
if even (x) {  
    if odd(y) { z := 42; }  
}
```

Safe  $z \neq 42$   
E.g.,  $x := 1 / (42 - z)$

No relations!

Given a specification of the possible errors

$Q \triangleq \{ z = 42 \}$

With **IL** one can prove

$[z=11] c_{42} [z=42 \wedge \text{odd}(y) \wedge \text{even}(x)]$

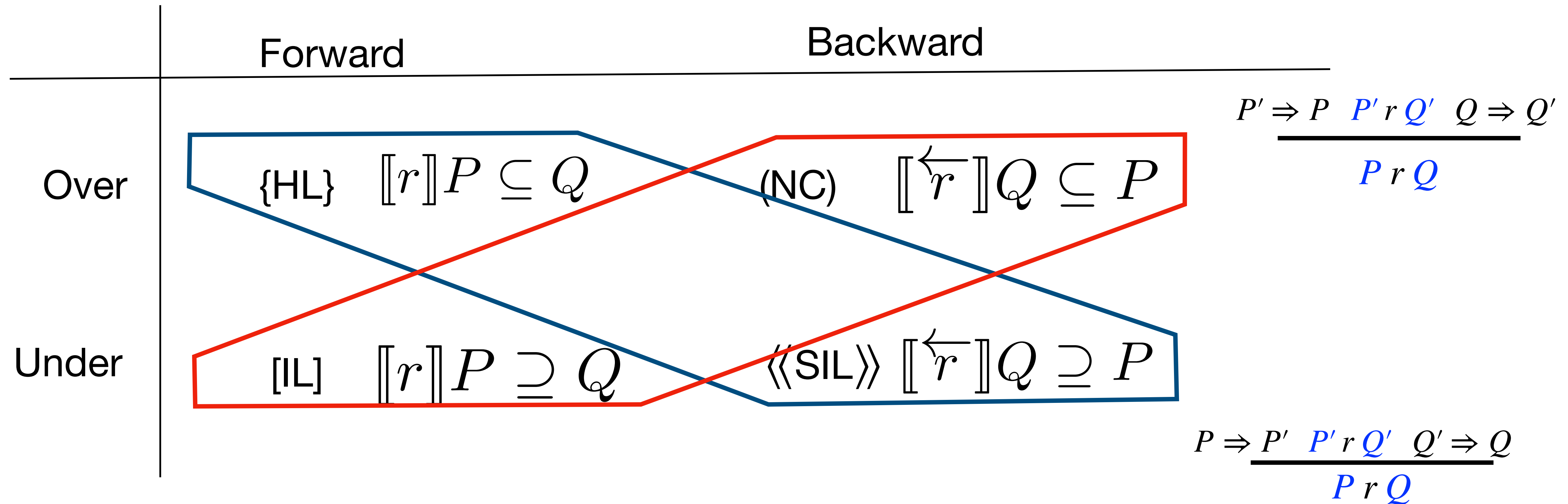
Expressing that the postcondition is reachable

With **SIL** one can prove

$\langle\langle z=11 \wedge \text{odd}(y) \wedge \text{even}(x) \rangle\rangle c_{42} \langle\langle z=42 \rangle\rangle$

Expressing a precondition that leads to error states

# Compare logics according to the consequence rule



Consequence rules follows the diagonal of the schema, so they suggest relations between HL-SIL and IL-NC

# Relations following the diagonals

NC-IL: no relation

HL-SIL: loosely related,

$r$  **deterministic** and **terminating**: **SIL** equivalent to **HL**

$$\langle\langle P \rangle\rangle r \langle\langle Q \rangle\rangle \Leftrightarrow \{P\} r \{Q\}$$

# SIL vs HL

$c_{42}$  :

$x := \text{nondet};$

if *even* (x) {

    if *odd*(y) {  $z := 42;$  }

}

Safe  $z \neq 42$

E.g.,  $x := 1 / (42 - z)$

Given a specification of the possible errors

$Q \triangleq \{ z = 42 \}$

With **SIL** one can prove

$\langle\langle \text{odd}(y) \rangle\rangle c_{42} \langle\langle z=42 \rangle\rangle$

Expressing a precondition that leads to error states

With **HL** one can prove

$\{ z=42 \} c_{42} \{ z=42 \}$

# SIL Rules

The proof system favours backward analysis starting from the (error) postconditions

Hoare's axiom for assignment

$$\frac{}{\langle\langle Q[a/x] \rangle\rangle x := a \langle\langle Q \rangle\rangle} \langle\langle atom - a \rangle\rangle$$

$$\langle\langle y > 0 \rangle\rangle x := y - 1 \quad \langle\langle x \geq 0 \rangle\rangle$$

$$\langle\langle y \neq 43 \rangle\rangle x := y - 1 \quad \langle\langle x \neq 42 \rangle\rangle$$



# SIL Rules

The proof system favours backward analysis starting from the (error) postconditions

$$\frac{}{\langle\langle Q \cap b \rangle\rangle b? \langle\langle Q \rangle\rangle} \langle\langle atom - g \rangle\rangle$$
$$\begin{array}{ccc} \langle\langle \emptyset \rangle\rangle & (x > 0)? & \langle\langle x = -42 \rangle\rangle \\ \langle\langle x = 42 \rangle\rangle & (x > 0)? & \langle\langle x = 42 \rangle\rangle \end{array}$$

# SIL Rules

The proof system favours backward analysis starting from the (error) postconditions

Same conditions for both branches

$$\frac{\langle\langle P_1 \rangle\rangle r_1 \langle\langle Q \rangle\rangle \quad \langle\langle P_2 \rangle\rangle r_2 \langle\langle Q \rangle\rangle}{\langle\langle P_1 \cup P_2 \rangle\rangle r_1 + r_2 \langle\langle Q \rangle\rangle} \langle\langle choice \rangle\rangle$$

$$\begin{array}{lll} \langle\langle y = 43 \vee y = 42 \rangle\rangle & (x := y - 1) + (x = y) & \langle\langle x = 42 \rangle\rangle \\ \langle\langle true \rangle\rangle = \langle\langle y \neq 43 \vee y \neq 42 \rangle\rangle & (x := y - 1) + (x := y) & \langle\langle x \neq 42 \rangle\rangle \\ \langle\langle y \neq 43 \rangle\rangle & (x := y - 1) + (x := 42) & \langle\langle x \neq 42 \rangle\rangle \end{array}$$

# SIL Rules

The proof system favours backward analysis starting from the (error) postconditions

Backward iteration starting from final state  $Q_0$

$$\frac{\forall n \geq 0. \langle\langle Q_{n+1} \rangle\rangle r \langle\langle Q_n \rangle\rangle}{\langle\langle \bigcup_{n \geq 0} Q_n \rangle\rangle r^* \langle\langle Q_0 \rangle\rangle} \quad \langle\langle iter \rangle\rangle$$

$$\langle\langle x \leq 42 \rangle\rangle = \langle\langle \dots \vee x = 41 \vee x = 42 \rangle\rangle (x := x + 1)^* \quad \langle\langle x = 42 \rangle\rangle$$

# SIL Rules

The proof system favours backward analysis starting from the (error) postconditions

**SIL** can drop disjunction going backward:

$$\frac{}{\langle\langle\emptyset\rangle\rangle r \langle\langle Q\rangle\rangle} \quad \langle\langle empty\rangle\rangle \qquad \frac{\langle\langle P \cup P'\rangle\rangle r \langle\langle Q\rangle\rangle}{\langle\langle P\rangle\rangle r \langle\langle Q\rangle\rangle} \quad \langle\langle cons'\rangle\rangle$$

$$\langle\langle x = 41 \vee x = 42\rangle\rangle \quad (x := x + 1)^* \quad \langle\langle x = 42\rangle\rangle$$

# **Validity, soundness and completeness**

# A proof system for SIL

Core rules

$$\frac{}{\langle\langle [\bar{c}]Q \rangle\rangle c \langle\langle Q \rangle\rangle} \langle\langle \text{atom} \rangle\rangle$$

$$\frac{P \subseteq P' \quad \langle\langle P' \rangle\rangle r \langle\langle Q' \rangle\rangle \quad Q' \subseteq Q}{\langle\langle P \rangle\rangle r \langle\langle Q \rangle\rangle} \langle\langle \text{cons} \rangle\rangle$$

$$\frac{\langle\langle P_1 \rangle\rangle r_1 \langle\langle Q \rangle\rangle \quad \langle\langle P_2 \rangle\rangle r_2 \langle\langle Q \rangle\rangle}{\langle\langle P_1 \cup P_2 \rangle\rangle r_1 + r_2 \langle\langle Q \rangle\rangle} \langle\langle \text{choice} \rangle\rangle$$

$$\frac{\langle\langle P \rangle\rangle r_1 \langle\langle R \rangle\rangle \quad \langle\langle R \rangle\rangle r_2 \langle\langle Q \rangle\rangle}{\langle\langle P \rangle\rangle r_1; r_2 \langle\langle Q \rangle\rangle} \langle\langle \text{seq} \rangle\rangle$$

$$\frac{\forall n \geq 0. \langle\langle Q_{n+1} \rangle\rangle r \langle\langle Q_n \rangle\rangle}{\langle\langle \bigcup_{n \geq 0} Q_n \rangle\rangle r^* \langle\langle Q_0 \rangle\rangle} \langle\langle \text{iter} \rangle\rangle$$

Additional rules

$$\frac{}{\langle\langle \emptyset \rangle\rangle r \langle\langle Q \rangle\rangle} \langle\langle \text{empty} \rangle\rangle$$

$$\frac{\langle\langle P_1 \rangle\rangle r \langle\langle Q_1 \rangle\rangle \quad \langle\langle P_2 \rangle\rangle r \langle\langle Q_2 \rangle\rangle}{\langle\langle P_1 \cup P_2 \rangle\rangle r \langle\langle Q_1 \cup Q_2 \rangle\rangle} \langle\langle \text{disj} \rangle\rangle$$

$$\frac{}{\langle\langle Q \rangle\rangle r^* \langle\langle Q \rangle\rangle} \langle\langle \text{iter0} \rangle\rangle$$

$$\frac{\langle\langle P \rangle\rangle r^*; r \langle\langle Q \rangle\rangle}{\langle\langle P \rangle\rangle r^* \langle\langle Q \rangle\rangle} \langle\langle \text{unroll} \rangle\rangle$$

$$\frac{\langle\langle P \rangle\rangle r^*; r \langle\langle Q_1 \rangle\rangle}{\langle\langle P \cup Q_2 \rangle\rangle r^* \langle\langle Q_1 \cup Q_2 \rangle\rangle} \langle\langle \text{unroll-split} \rangle\rangle$$

# Soundness and completeness

**SIL** validity of a triple :  $\llbracket \overleftarrow{r} \rrbracket Q \supseteq P$

**Th.** [*Soundness*]

All provable triples (including additional rules) are valid

**Th.** [*Completeness*]

All valid triples are provable (using the core rules)

# Questions



# Question 1

Which SIL triples are valid for any  $r$  and  $P$  ?

$\langle\langle \text{false} \rangle\rangle r \langle\langle P \rangle\rangle$



$\langle\langle \text{true} \rangle\rangle r \langle\langle \text{true} \rangle\rangle$



$\langle\langle P \rangle\rangle r^* \langle\langle P \vee x = 0 \rangle\rangle$



$\langle\langle wlp(r, P) \rangle\rangle r \langle\langle P \rangle\rangle$



# Question 2

Prove that rule [conj] is **unsound** for SIL

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

Consider  $\langle\langle x = 0 \rangle\rangle x := \text{nondet}() \langle\langle x = 0 \rangle\rangle$

and  $\langle\langle x = 0 \rangle\rangle x := \text{nondet}() \langle\langle x = 1 \rangle\rangle$

By rule [conj] we could derive  $\langle\langle x = 0 \rangle\rangle x := 1 \langle\langle \text{false} \rangle\rangle$

which is not sound!

# \* Exam 11

Find a derivation for the SIL triple

$\langle\langle \text{true} \rangle\rangle$  if  $x \geq y$  then  $z := x$  else  $z := y$   $\langle\langle z = \max(x, y) \rangle\rangle$

# \* Exam 12

Prove or disprove the validity of the following axiom in SIL

---

$$\langle\langle P \rangle\rangle (b)? \quad \langle\langle P \wedge b \rangle\rangle$$