# Trace Abstraction (Recap)

Andreas Podelski

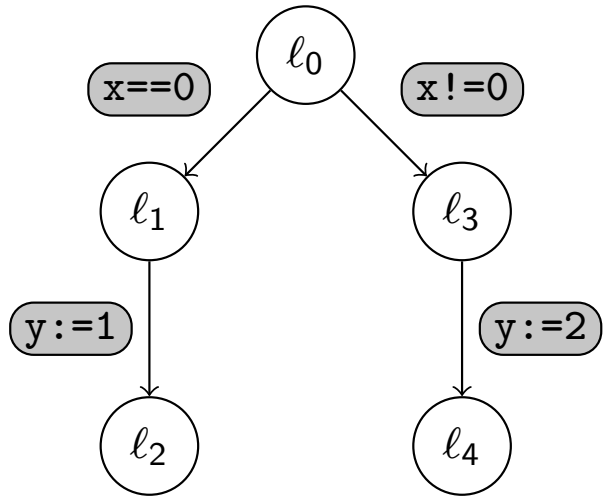
University of Freiburg, Germany

Tuesday, December 15, 2011

# Preliminaries: Programs

program = graph

```
ℓ0:  if (x==0)
ℓ1:     y:=1
ℓ2:
     else
ℓ3:     y:=2
ℓ4:
```

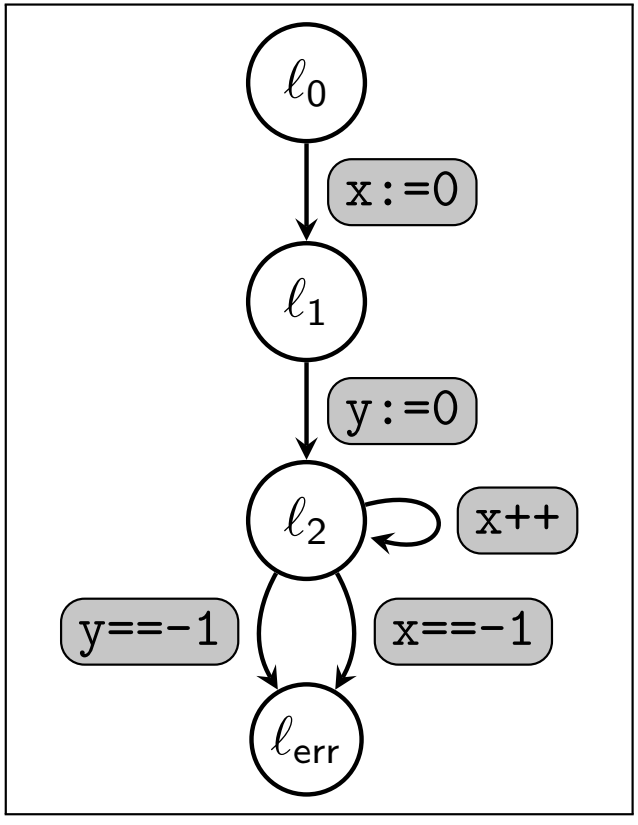



- nodes = control locations
  two special nodes:
  - initial location $\ell_0$
  - error location $\ell_{\text{err}}$
- edges labeled by statements
  only two kinds of statements:
  - update e.g., `y:=1`, `y:=2`
  - assume e.g., `x==0`, `x!=0`

# Running Example: Program $\mathcal{P}$

```
ℓ0:  x:=0
ℓ1:  y:=0
ℓ2:  while(nondet) {x++}
     assert x!= -1
     assert y!= -1
```

program $\mathcal{P}$




program $\mathcal{P}$

# Trace = Word over an Alphabet

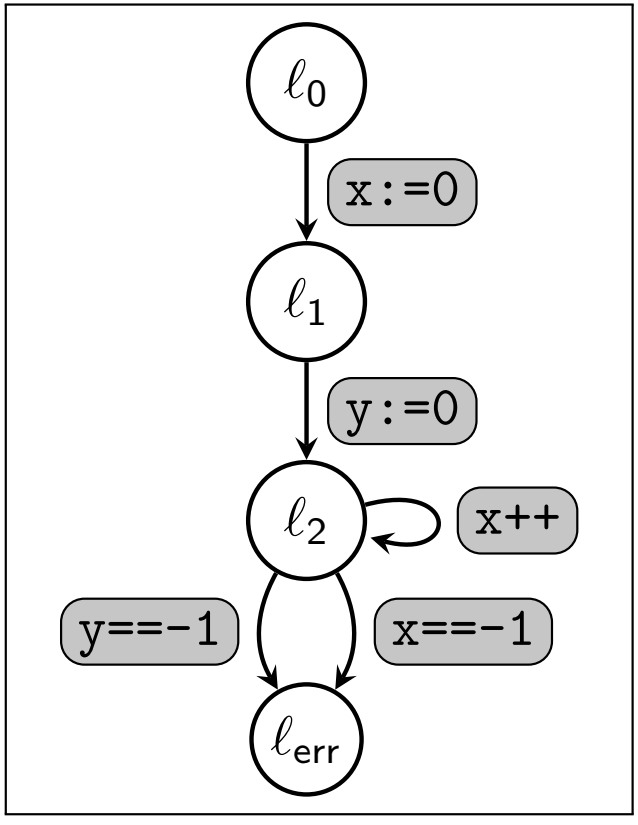
alphabet $\Sigma$ = set of statements

$\Sigma = \{$ `x:=0`, `y:=0`, `x++`, `x==-1`, `y==-1` $\}$

examples

$w_1 =$ `y==-1`.`x++`.`x++`.`x:=0`.`x==-1`

$w_2 =$ `x:=0`.`y:=0`.`x++`.`x++`.`y==-1`



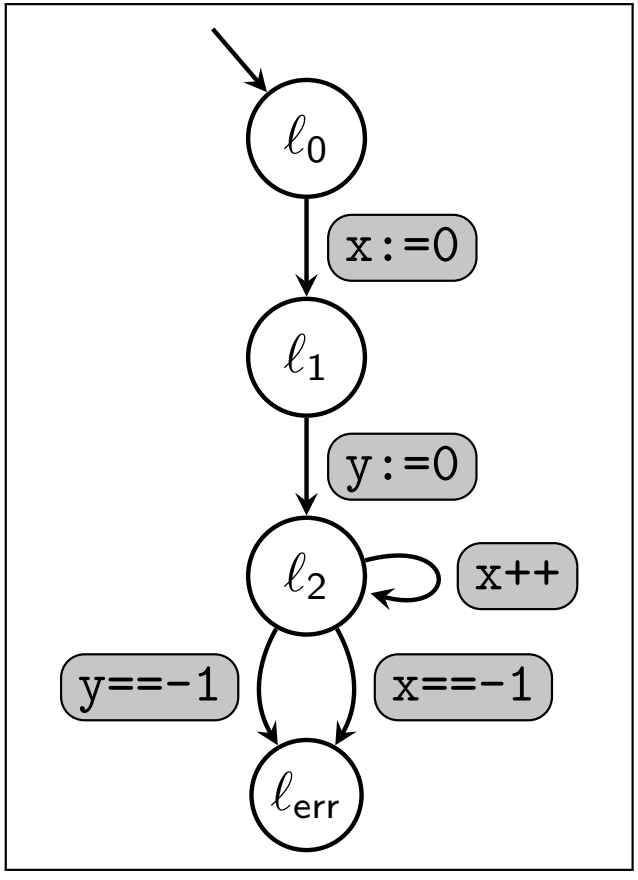

program $\mathcal{P}$

# Error Trace

error trace = word w along path from $\ell_0$ to $\ell_{\text{err}}$

$w_1 =$ `x:=0`.`y:=0`.`y==-1`

$w_2 =$ `x:=0`.`y:=0`.`x++`.`x++`.`y==-1`




program $\mathcal{P}$

# Feasible Trace

feasible trace = word $w$ formed by letter of an possible execution path

$w_1 =$ `x==-1`.`x:=0` feasible

$w_2 =$ `x:=0`.`x==-1` not feasible

$w_3 =$ `x>=0`.`x--`.`x--`. . . . .`x--`.`x--` feasible

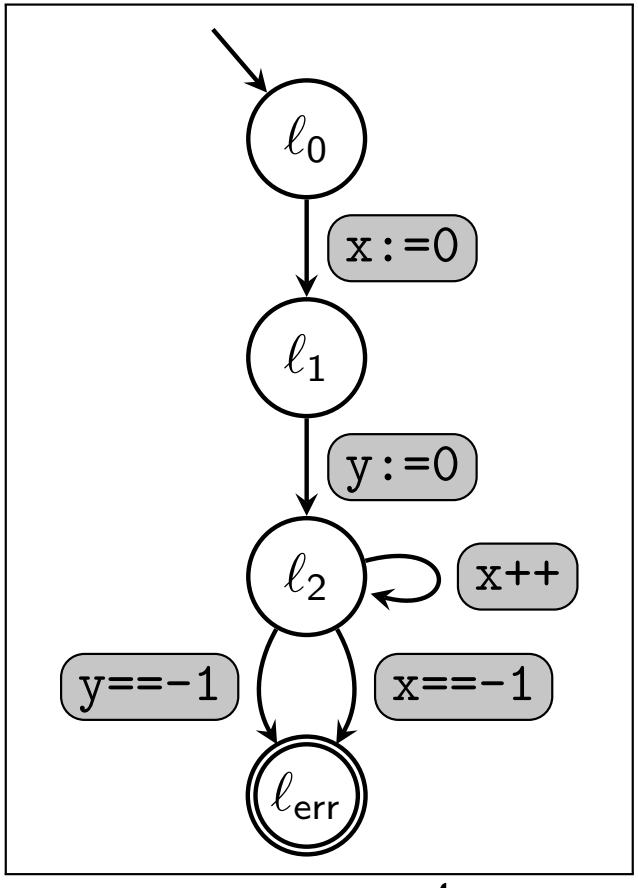
(`x--` . . . `x--`: finitely many)

$w_4 =$ `x>=0`.`x--`.`x--`.`x--`.`x--`. . . not feasible

(`x--` . . . : infinitely many)

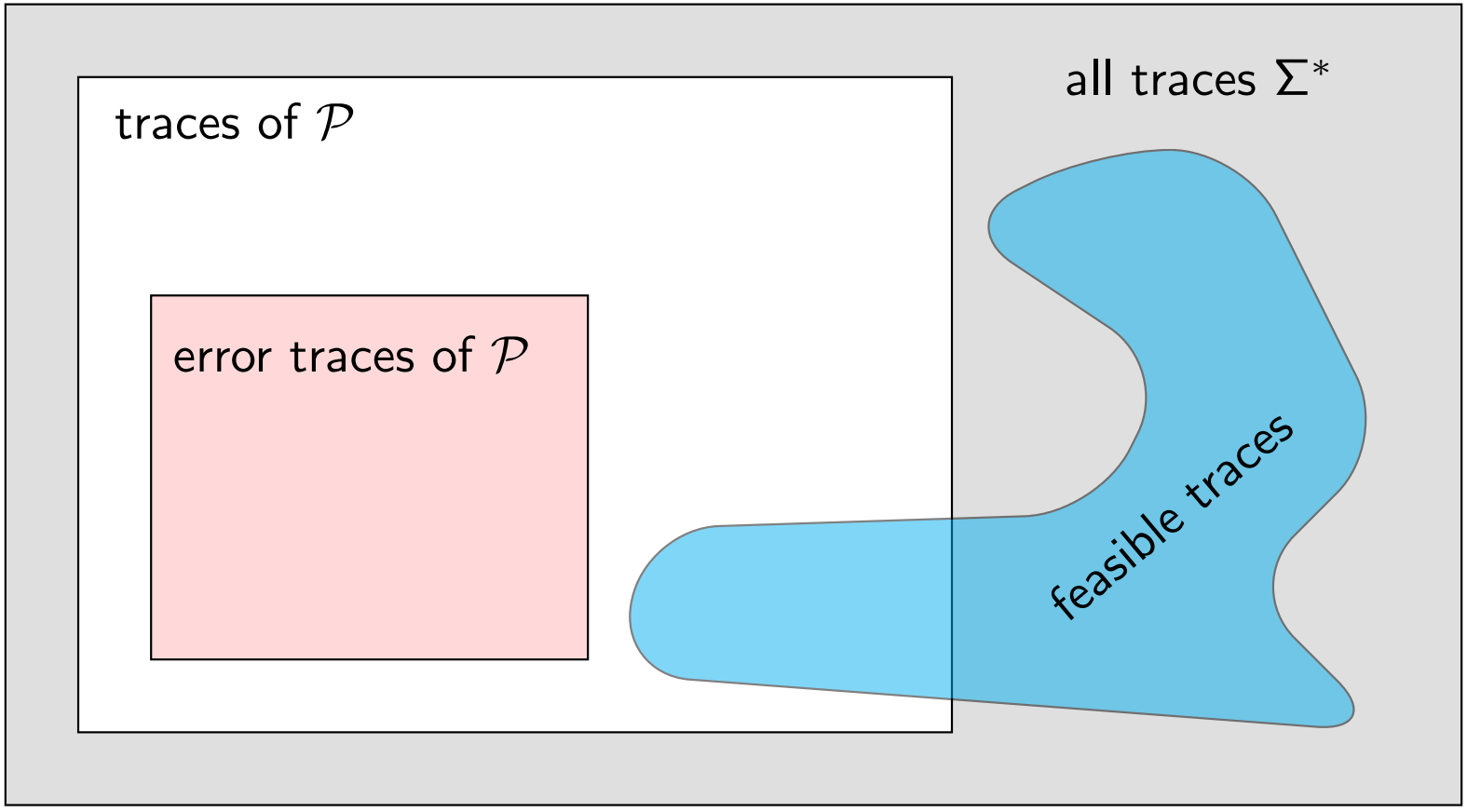
# Automata over Alphabet $\Sigma$ of Statements

- automaton $\mathcal{A}_\mathcal{P}$ defines set of error traces
- note: set of feasible traces can not be defined by automaton

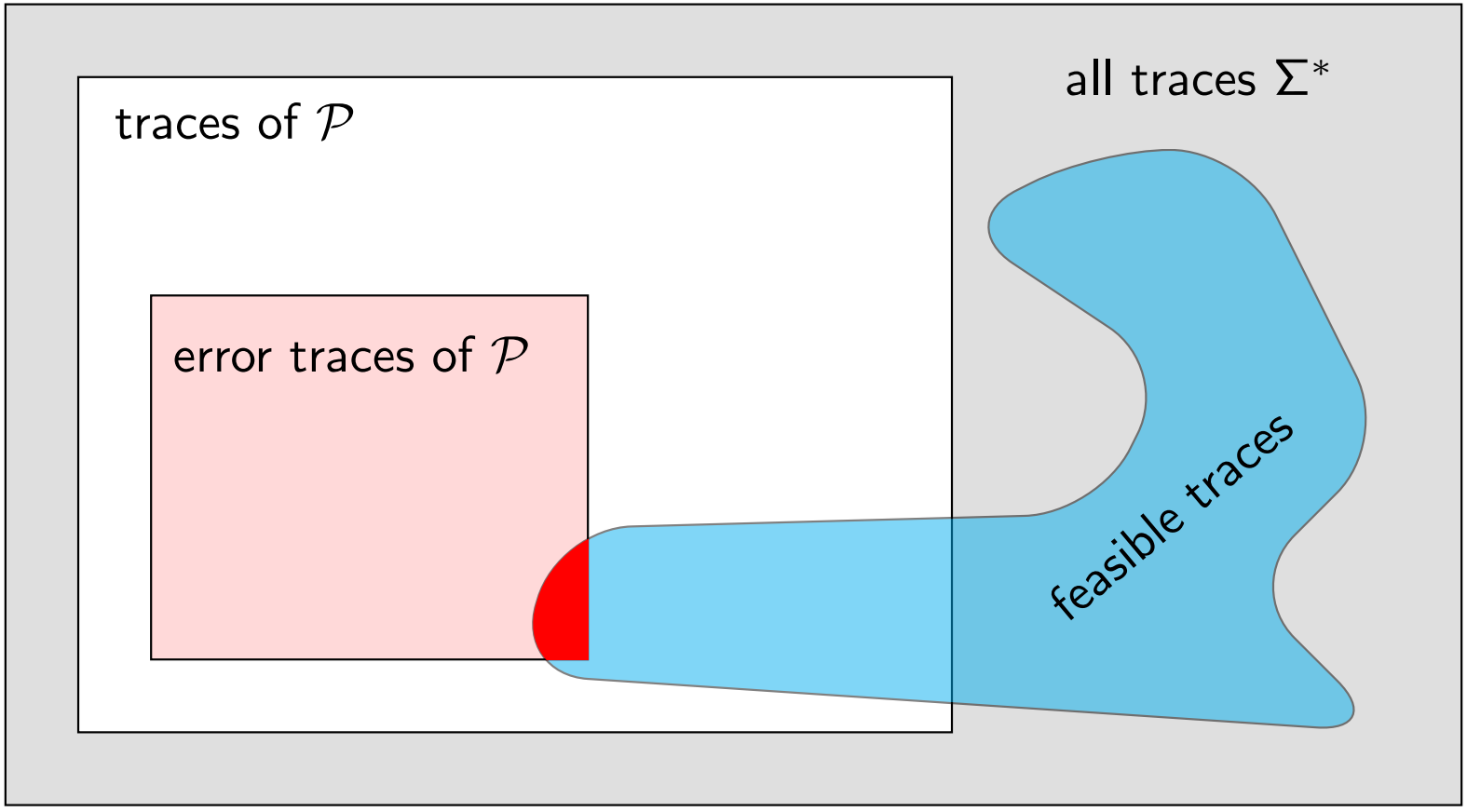



automaton $\mathcal{A}_\mathcal{P}$

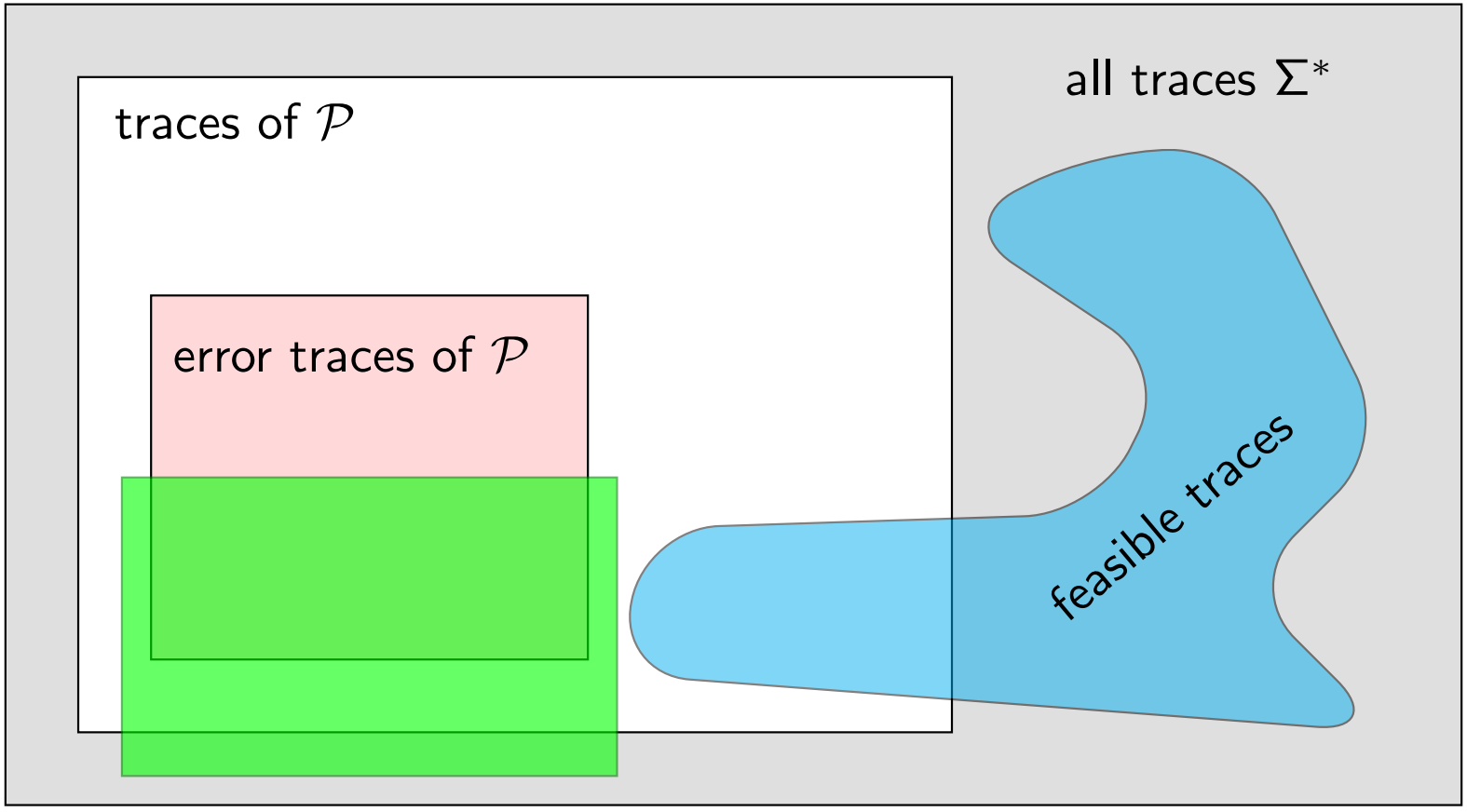
# Correctness of Program $\mathcal{P}$



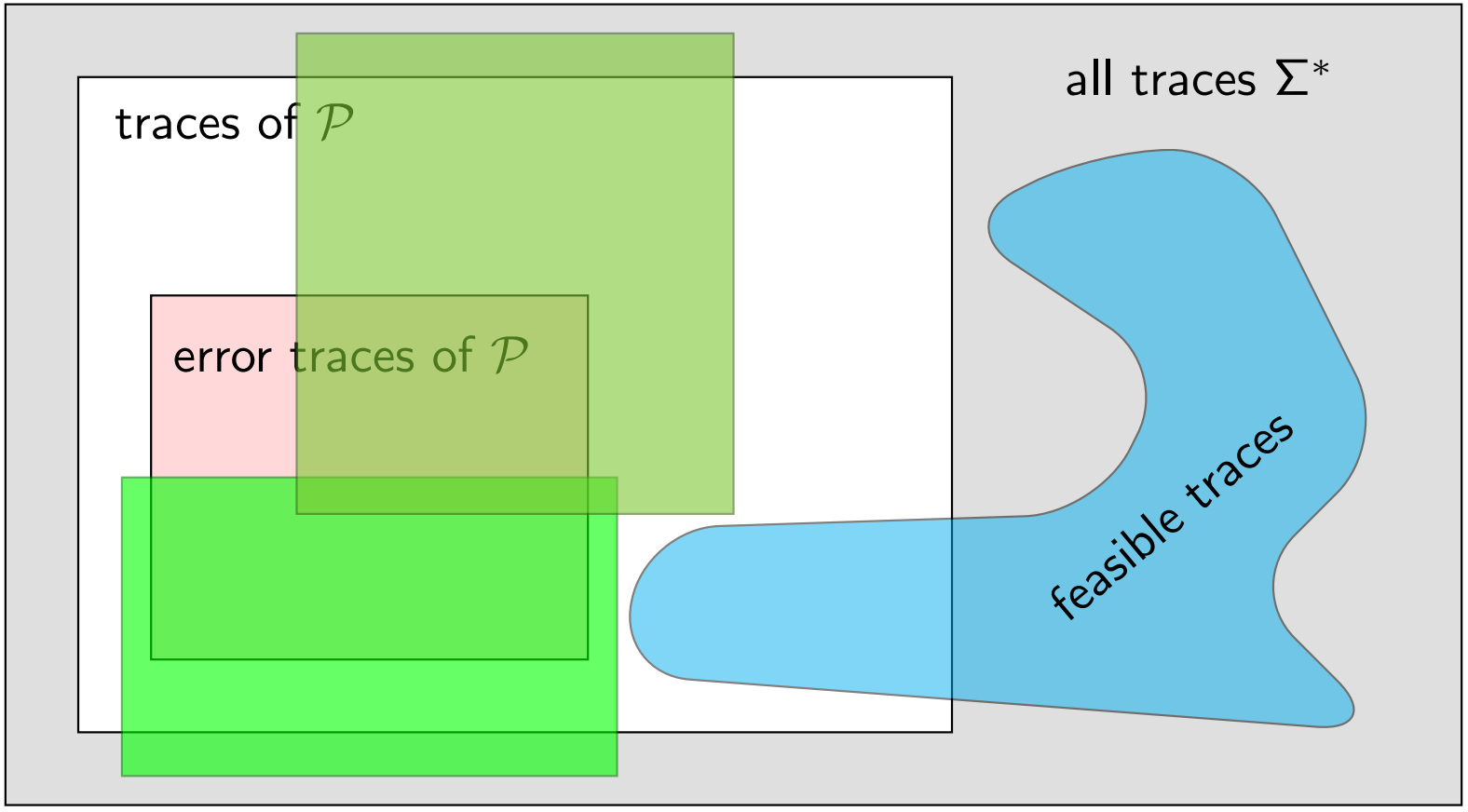
# Incorrectness of Program $\mathcal{P}$



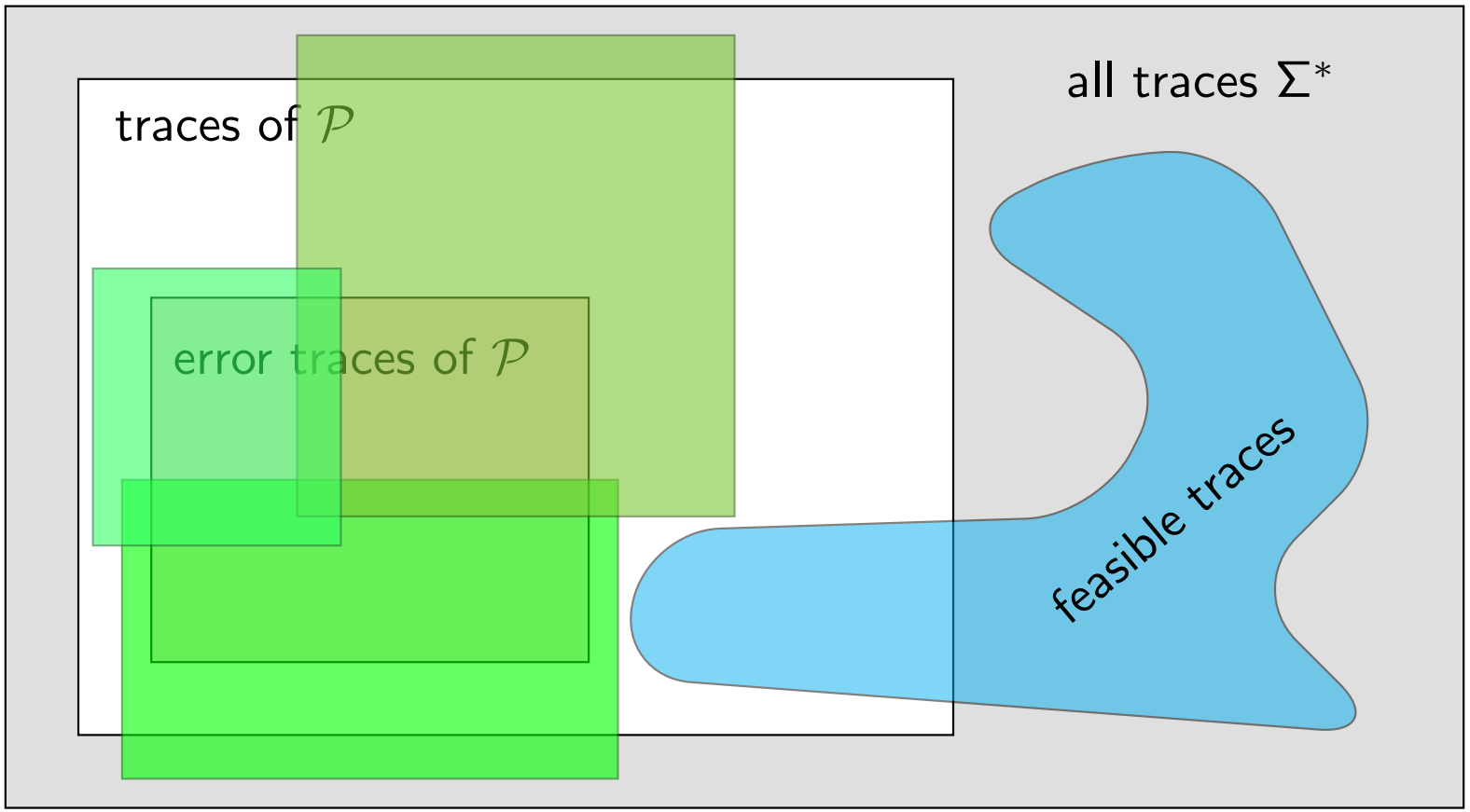
# Decomposition of Correctness Proof



# Decomposition of Correctness Proof



# Decomposition of Correctness Proof



program $\mathcal{P}$

alphabet $\Sigma$ = set of statements

automaton $\mathcal{A}_\mathcal{P}$ = set of error traces for program $\mathcal{P}$

**Proof Rule**

$$\begin{array}{rcl} \mathcal{A}_\mathcal{P} & \subseteq & \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n \\ \mathcal{A}_1, \ldots, \mathcal{A}_n & \subseteq & \Sigma^* \backslash \textsc{Feasible} \end{array} \quad \Rightarrow \quad \text{program } \mathcal{P} \text{ is correct}$$

"$\mathcal{A}_1, \ldots, \mathcal{A}_n$ are a decomposition of a correctness proof for program $\mathcal{P}$"

# How?

how do we obtain a decomposition $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of a correctness proof for program $\mathcal{P}$?

next:

1. specific algorithm à la CEGAR
2. generalization

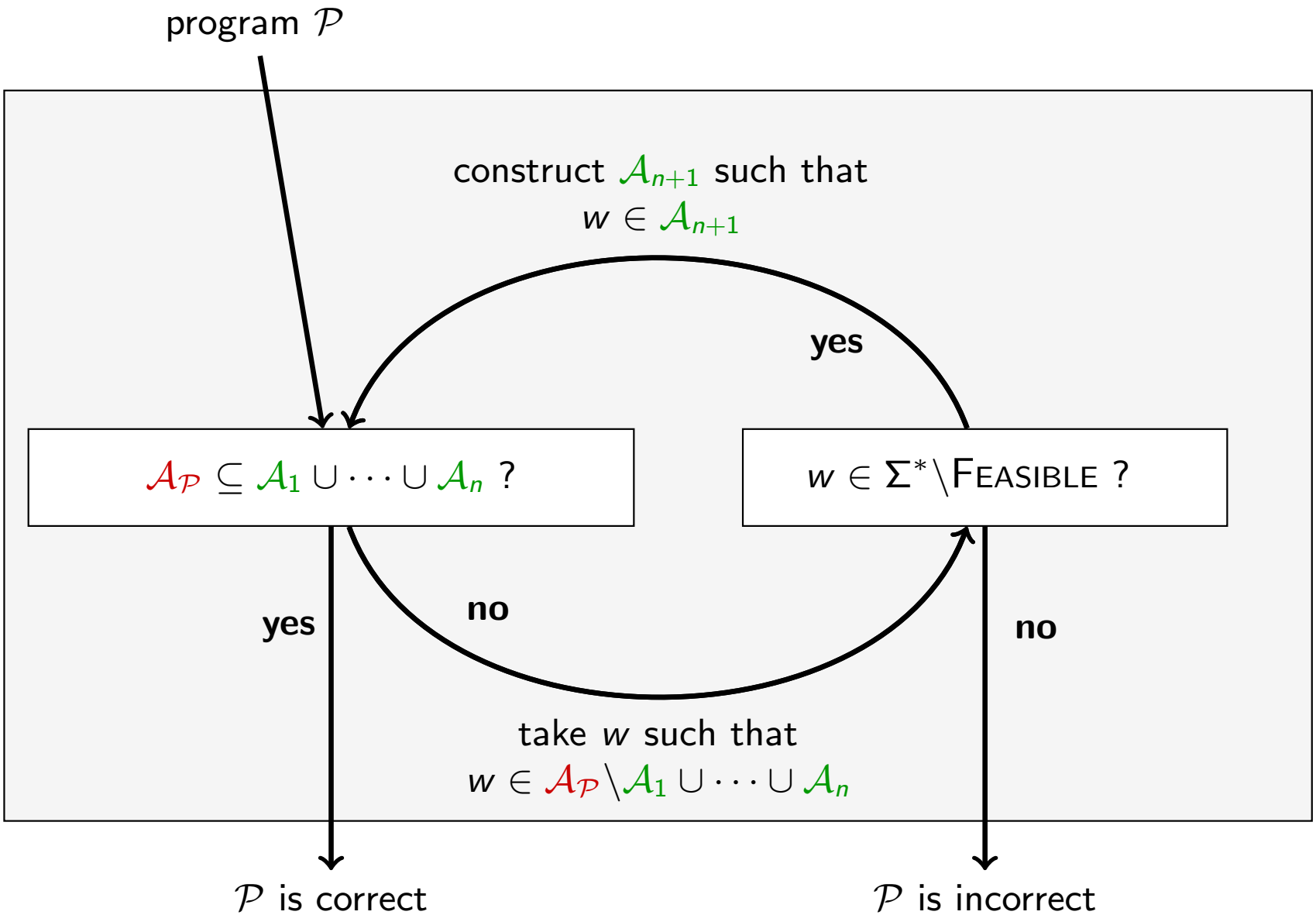
# How?

how do we obtain a decomposition $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of a correctness proof for program $\mathcal{P}$?

next:

1. specific algorithm à la CEGAR
2. generalization

# Compute Decomposition of Correctness Proof à la CEGAR



next:

generalization of counterexamples given trace w such that

1. $w \in \mathcal{A}_{\mathcal{P}} \backslash \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$ (" w is counterexample")
2. $w \in \Sigma^* \backslash \textsc{Feasible}$ ("w is infeasible")

next:

generalization of counterexamples given trace w such that

1. $w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$ (" w is counterexample")
2. $w \in \Sigma^* \setminus \textsc{Feasible}$ ("w is infeasible")

construct automaton $\mathcal{A}_{n+1}$ such that

1. $w \in \mathcal{A}_{n+1}$
2. $w \in \Sigma^* \setminus \textsc{Feasible}$

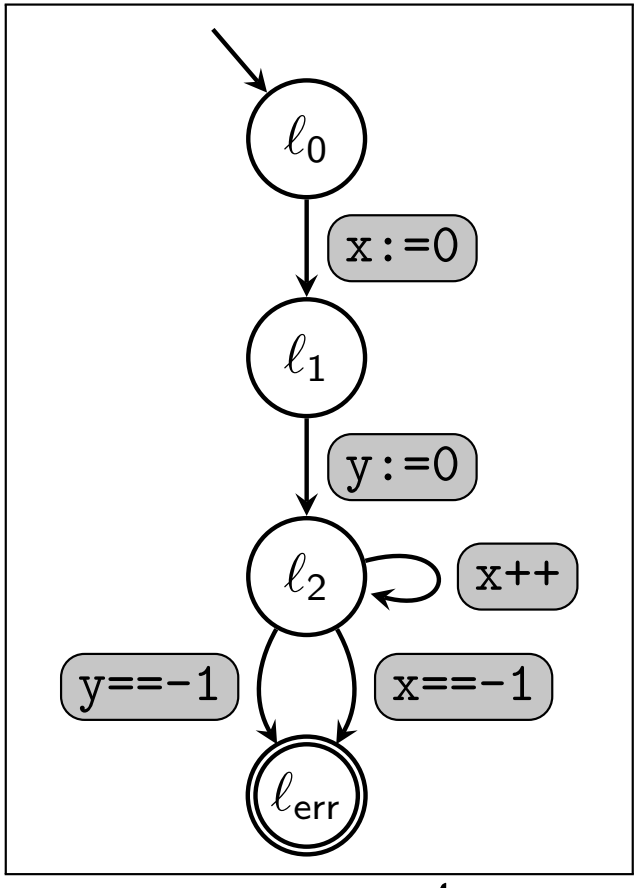
# First Iteration of Algorithm for Example Program $\mathcal{P}$

n=0, i.e. $\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n = \emptyset$

take $w_1 \in \mathcal{A}_\mathcal{P} \setminus \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$

$w_1 =$ `x:=0` . `y:=0` . `y==-1`

$w_1$ is not feasible




automaton $\mathcal{A}_\mathcal{P}$

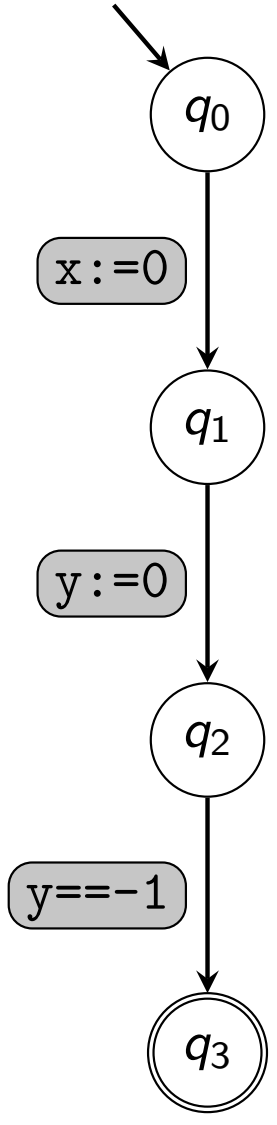
# First Iteration of Algorithm for Example Program $\mathcal{P}$

$w_1 =$ `x:=0`.`y:=0`.`y==-1`

construct automaton $\mathcal{A}_1$ such that

1. $w \in \mathcal{A}_1$
2. $w \in \Sigma^* \setminus \text{FEASIBLE}$

trivial solution: $\mathcal{A}_1 = \{w_1\}$



# First Iteration of Algorithm for Example Program $\mathcal{P}$

$w_1 =$ `x:=0`.`y:=0`.`y==-1`

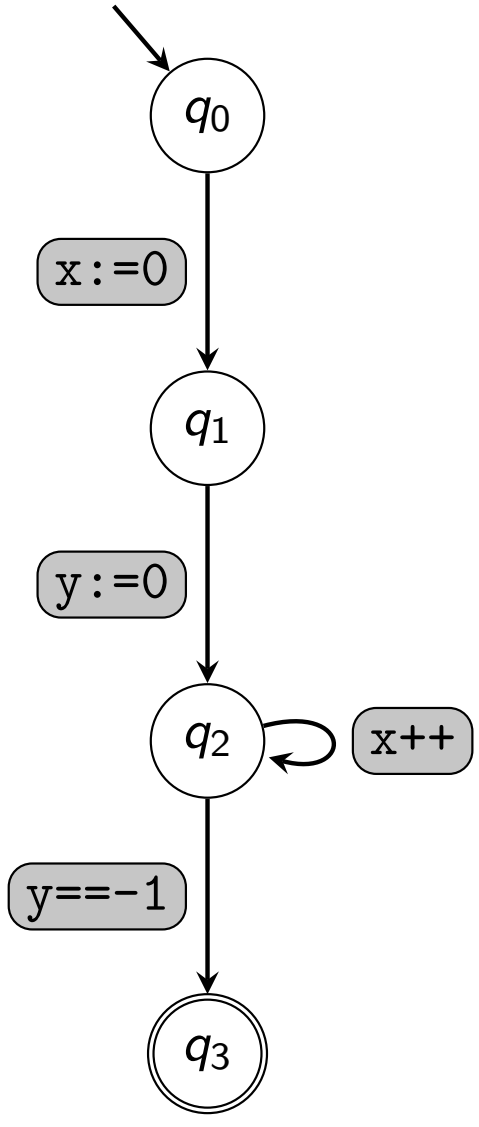
construct automaton $\mathcal{A}_1$ such that

1. $w \in \mathcal{A}_1$
2. $w \in \Sigma^* \setminus \text{FEASIBLE}$

trivial solution: $\mathcal{A}_1 = \{w_1\}$

observe:
statement `x:++` does not affect variable y

therefore
`x:=0`.`y:=0`.`x:++`.`y==-1`
is also infeasible



# Second Iteration of Algorithm for Example Program $\mathcal{P}$

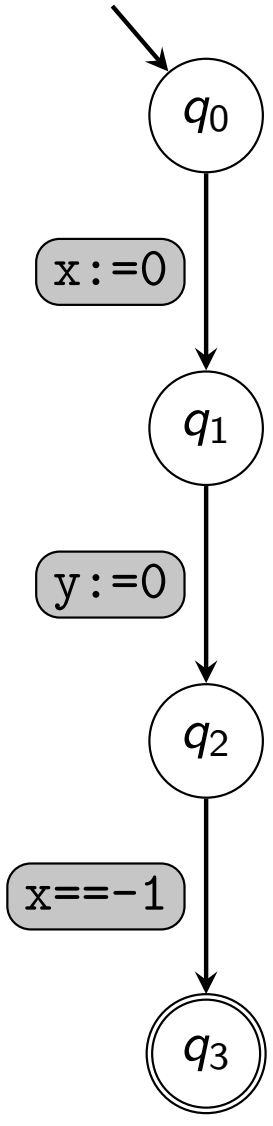
n=1 $\quad w_2 \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1$

$w_2 =$ `x:=0`.`y:=0`.`x==-1`

construct automaton $\mathcal{A}_2$ such that

1. $w \in \mathcal{A}_2$
2. $w \in \Sigma^* \setminus \text{FEASIBLE}$

trivial solution: $\mathcal{A}_2 = \{w_2\}$



observe:

proof of infeasiblity of

$$w_2 = \texttt{x:=0}.\texttt{y:=0}.\texttt{x==-1}$$

is correctness proof of Hoare triple

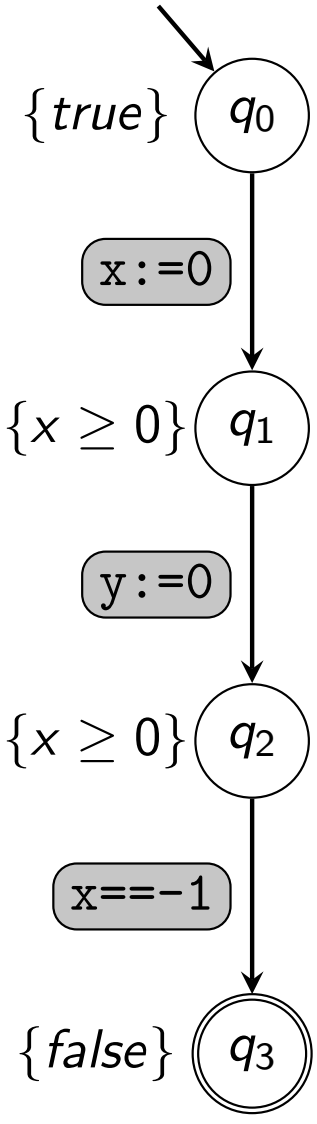
$$\{\mathit{true}\}\ w_2\ \{\mathit{false}\}$$

# Second Iteration of Algorithm for Example Program $\mathcal{P}$

$\{true\}$ `x:=0` $\{x \geq 0\}$

$\{x \geq 0\}$ `y:=0` $\{x \geq 0\}$

$\{x \geq 0\}$ `x==-1` $\{false\}$



# Second Iteration of Algorithm for Example Program $\mathcal{P}$

$\{true\}$ `x:=0` $\{x \geq 0\}$

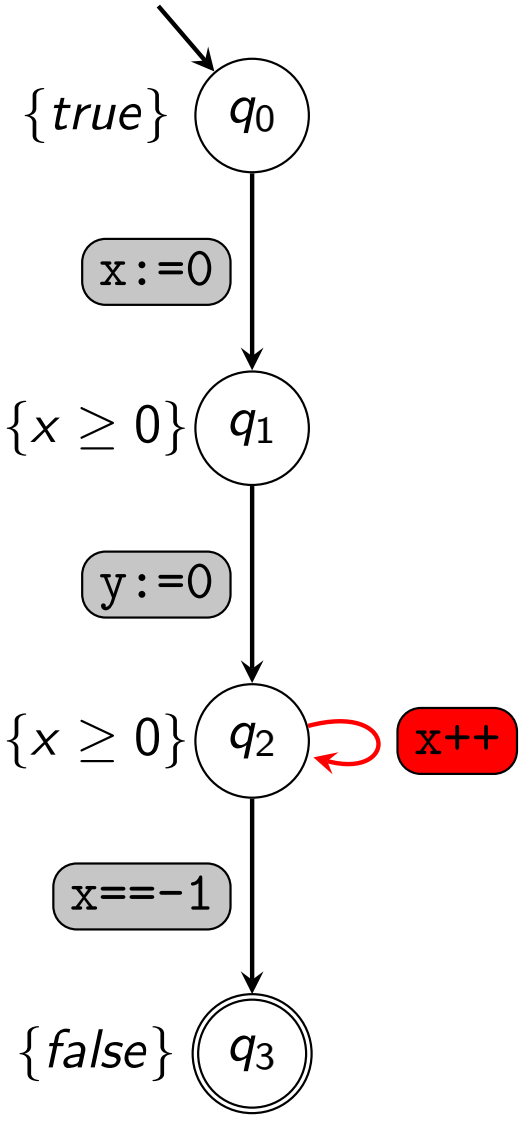
$\{x \geq 0\}$ `y:=0` $\{x \geq 0\}$

$\{x \geq 0\}$ `x==-1` $\{false\}$

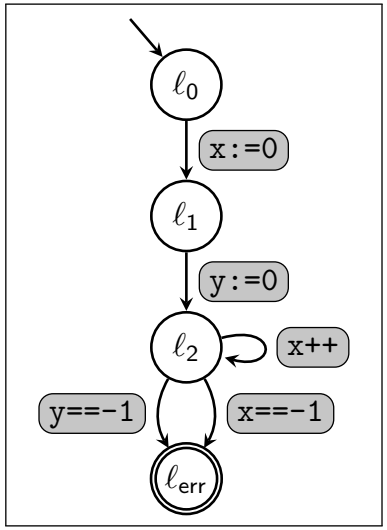
observe:

$\{x \geq 0\}$ `x++` $\{x \geq 0\}$

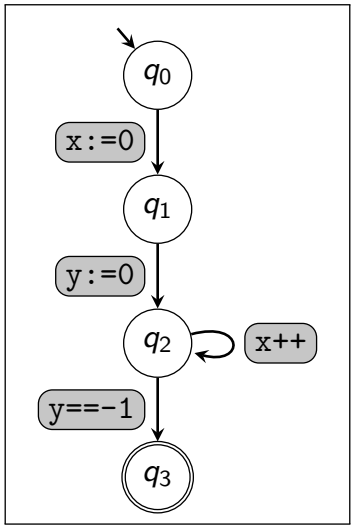
is a valid Hoare triple



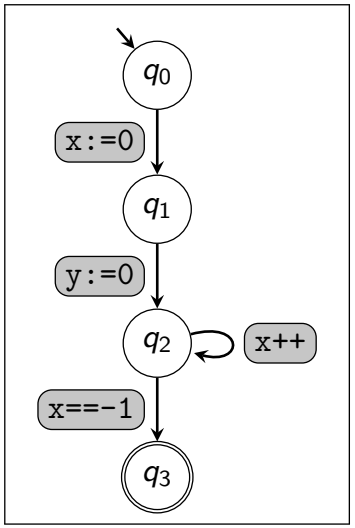
# $\mathcal{A}_1$ and $\mathcal{A}_2$ are Decomposition of a Correctness Proof for Program $\mathcal{P}$



automaton $\mathcal{A}_\mathcal{P}$



automaton $\mathcal{A}_1$



automaton $\mathcal{A}_2$

$\mathcal{A}_\mathcal{P} \subseteq \mathcal{A}_1 \cup \mathcal{A}_2$

$\mathcal{A}_1 \subseteq \Sigma^* \backslash \textsc{Feasible}$

$\mathcal{A}_2 \subseteq \Sigma^* \backslash \textsc{Feasible}$

**Proof Rule**

$$\begin{array}{rcl} \mathcal{A}_\mathcal{P} & \subseteq & \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n \\ \mathcal{A}_1, \ldots, \mathcal{A}_n & \subseteq & \Sigma^* \backslash \textsc{Feasible} \end{array} \quad \Rightarrow \quad \text{program } \mathcal{P} \text{ is correct}$$

# Interpolant Automaton $\mathcal{A}_{\mathcal{I}}$

## Definition (Interpolant Automaton $\mathcal{A}_{\mathcal{I}}$)

$\mathcal{I} = I_0, I_1, \ldots, I_n$ sequence of state predicates ("Interpolants")

$$\mathcal{A}_{\mathcal{I}} = \langle Q_{\mathcal{I}}, \delta_{\mathcal{I}}, Q^{\text{init}}_{\mathcal{I}}, Q^{\text{fin}}_{\mathcal{I}} \rangle$$

$$Q_{\mathcal{I}} = \{q_0, \ldots, q_n\}$$

$$(q_i, st, q_j) \in \delta_{\mathcal{I}} \quad \text{only if} \quad \{I_i\}\ st\ \{I_j\}$$

$$q_i \in Q^{\text{init}} \quad \text{only if} \quad I_i = \mathit{true}$$

$$q_i \in Q^{\text{fin}} \quad \text{only if} \quad I_i = \mathit{false}$$

# Interpolant Automaton $\mathcal{A}_\mathcal{I}$

## Definition (Interpolant Automaton $\mathcal{A}_\mathcal{I}$)

$\mathcal{I} = I_0, I_1, \ldots, I_n$ sequence of state predicates ("Interpolants")

$$\mathcal{A}_\mathcal{I} = \langle Q_\mathcal{I}, \delta_\mathcal{I}, Q_\mathcal{I}^{\text{init}}, Q_\mathcal{I}^{\text{fin}} \rangle$$

$$Q_\mathcal{I} = \{q_0, \ldots, q_n\}$$

$$(q_i, st, q_j) \in \delta_\mathcal{I} \quad \text{only if} \quad \{I_i\}\ st\ \{I_j\}$$

$$q_i \in Q^{\text{init}} \quad \text{only if} \quad I_i = \mathit{true}$$

$$q_i \in Q^{\text{fin}} \quad \text{only if} \quad I_i = \mathit{false}$$

## Theorem

*An interpolant automaton $\mathcal{A}_\mathcal{I}$ recognizes a subset of infeasible traces.*

$$\mathcal{L}(\mathcal{A}_\mathcal{I}) \subseteq \mathit{Infeasible}$$

# CEGAR with Database of Interpolant Automata