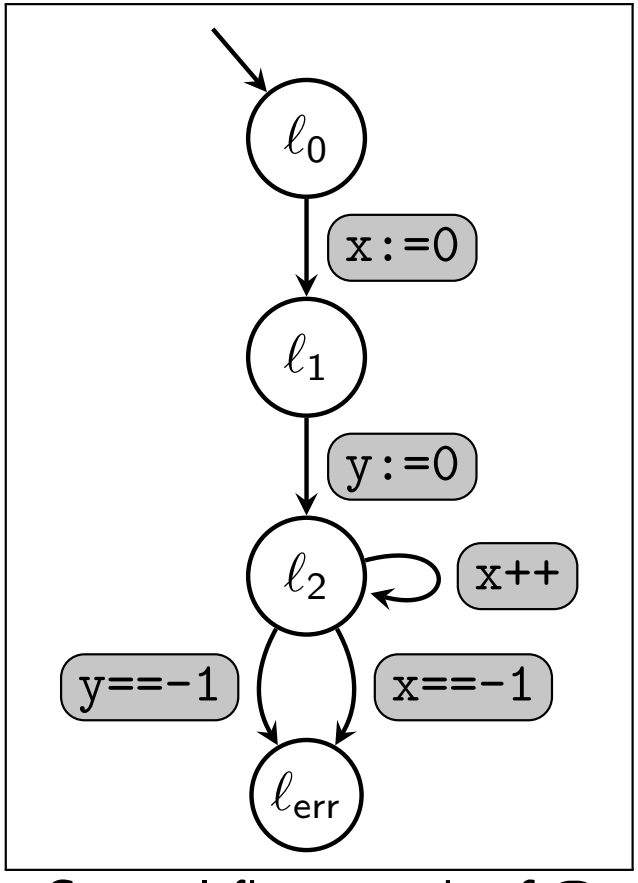
# Trace Abstraction

Monday, December 14, 2011

# Example – Our Model of a Verification Problem

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

Example program $\mathcal{P}$




Control flow graph of $\mathcal{P}$

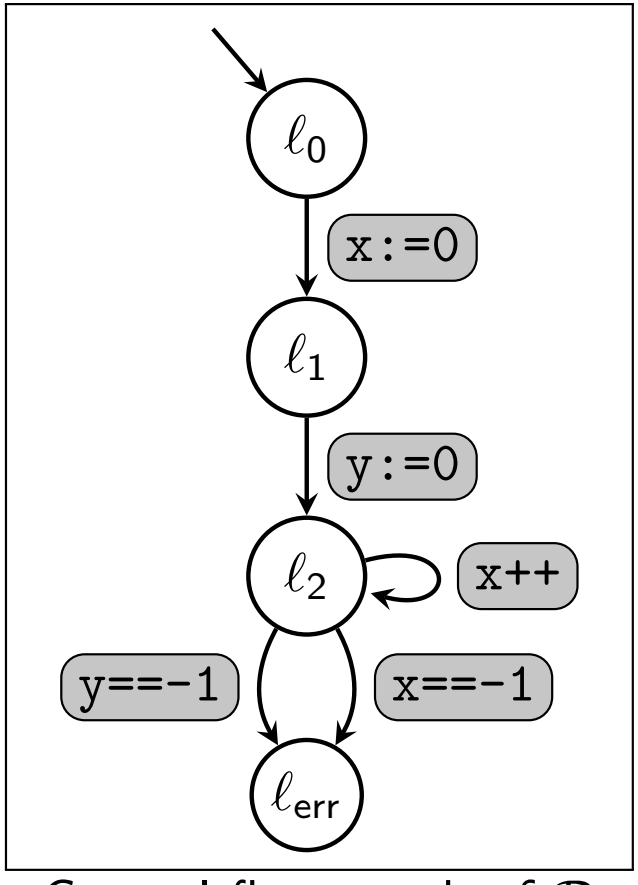
# Statements

## Statement

Letter of our alphabet. No further meaning.

In our example:

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




Control flow graph of $\mathcal{P}$

# Statements

## Statement

Letter of our alphabet. No further meaning.

In our example:

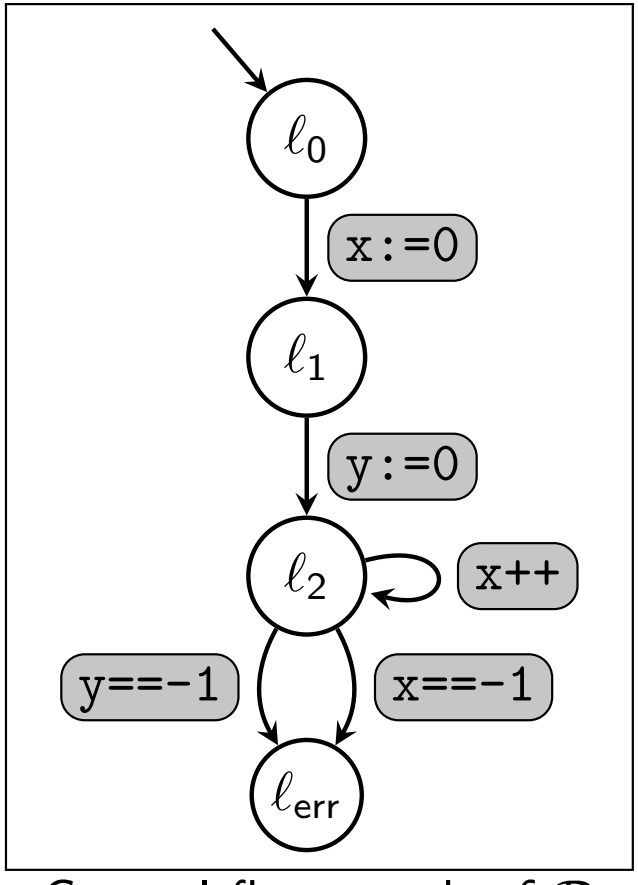
$\Sigma = \{$ `x:=0`, `y:=0`, `x++`, `x==-1`, `y==-1` $\}$

## Trace

Word over the alphabet of statements.

Example:

$\pi =$ `y==-1`.`x++`.`x++`.`x:=0`.`x==-1`




Control flow graph of $\mathcal{P}$

# Error Traces

## Control Automaton $\mathcal{A}_\mathcal{P}$

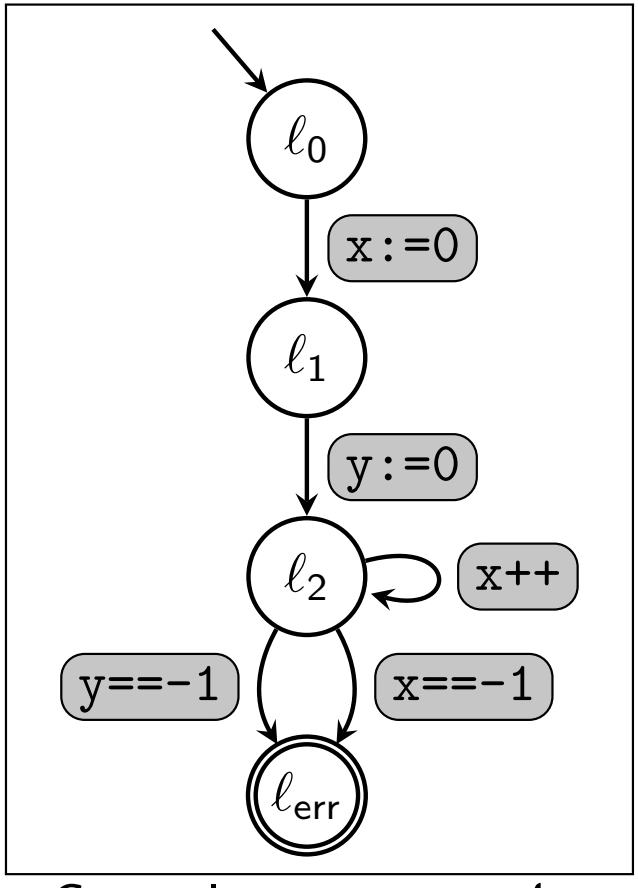
Automaton over the set of statements.
Encodes a verification problem.

$$\mathcal{A}_\mathcal{P} = \langle LOC, \delta, \{\ell_{\text{init}}\}, \{\ell_{\text{err}}\} \rangle$$

## Error Trace of $\mathcal{P}$

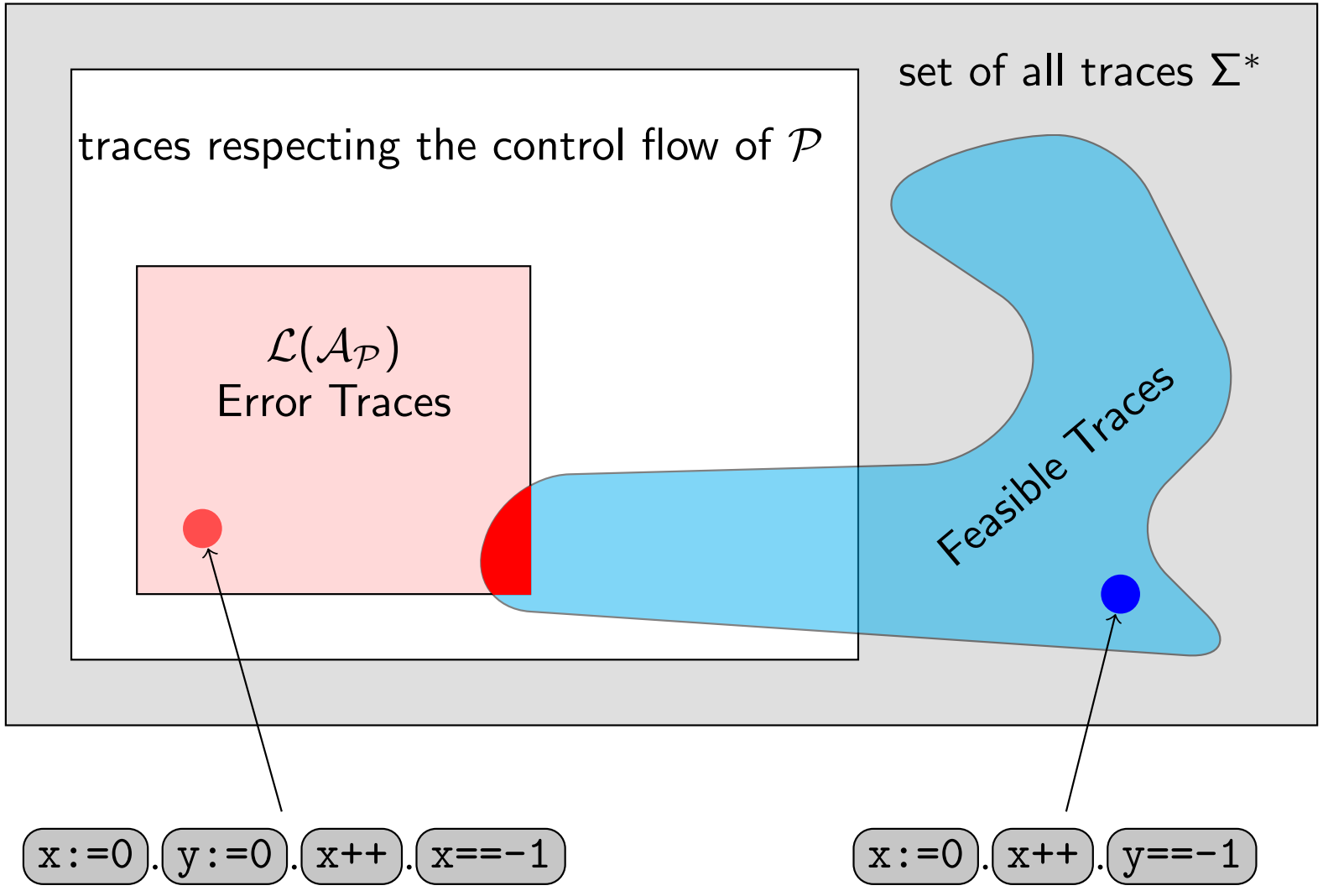
Trace accepted by $\mathcal{A}_\mathcal{P}$

In our example
$\pi =$ `x:=0` . `y:=0` . `x++` . `x==-1`
is an error trace.

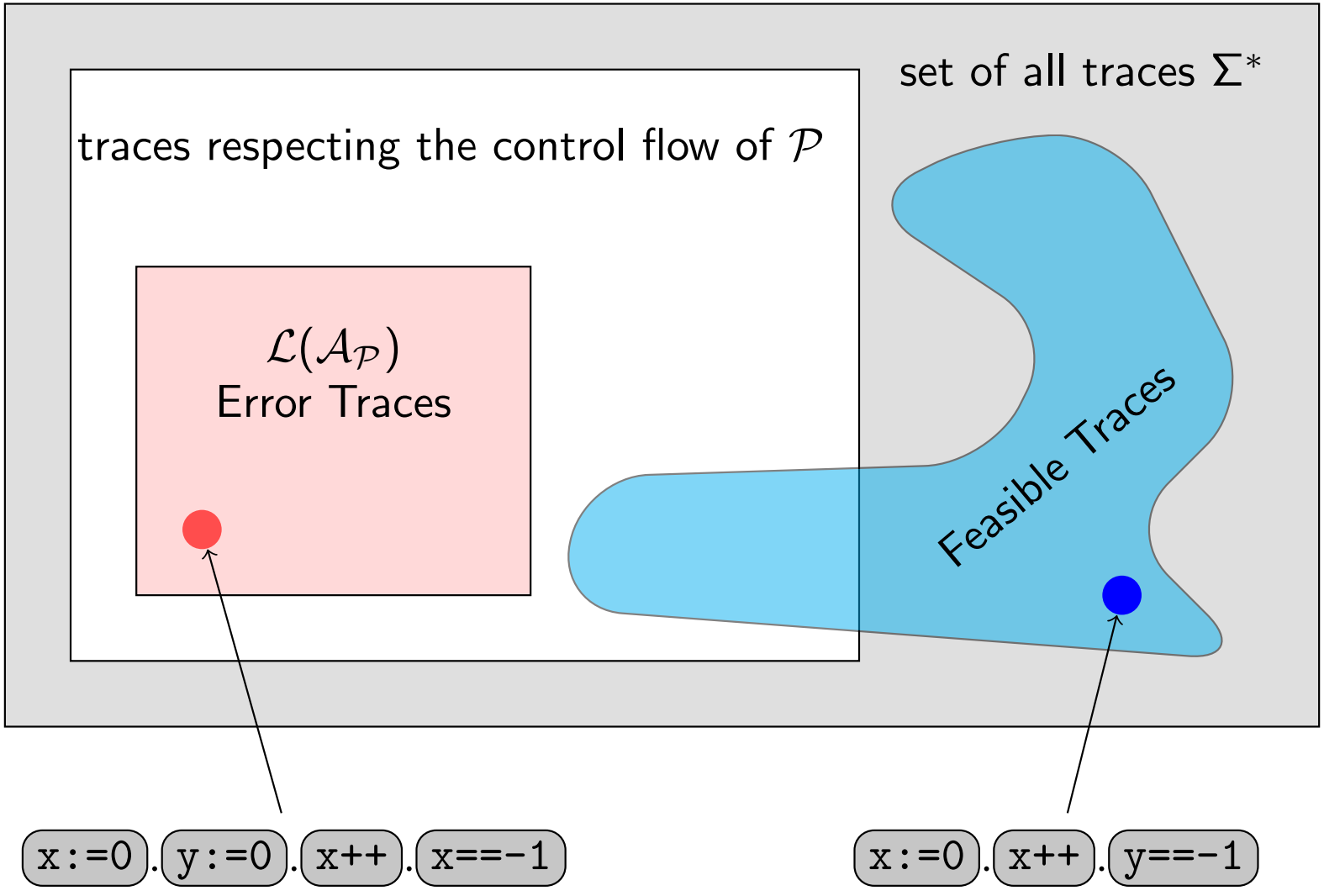



Control automaton $\mathcal{A}_\mathcal{P}$

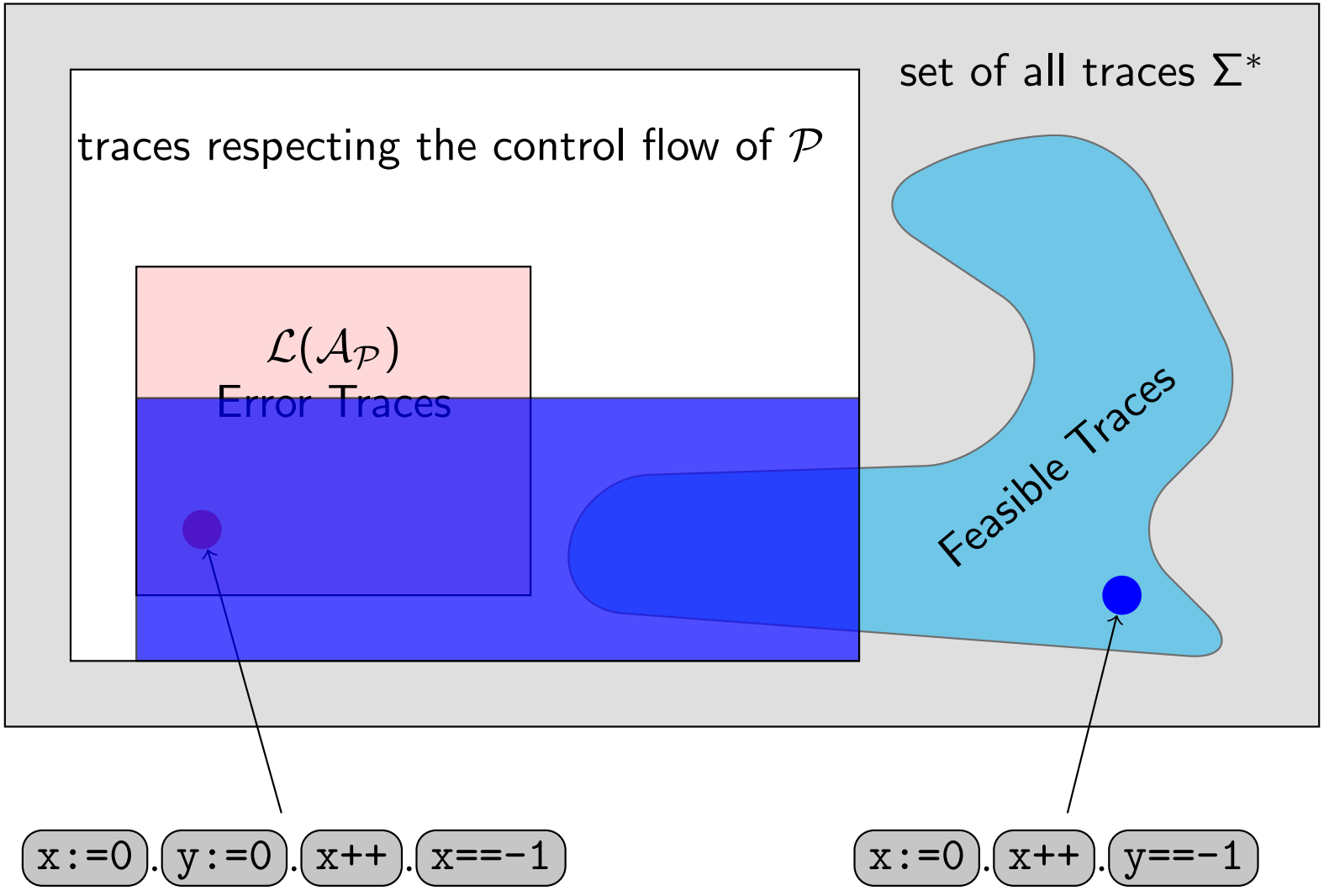
# Set Theoretic View of Trace Abstraction



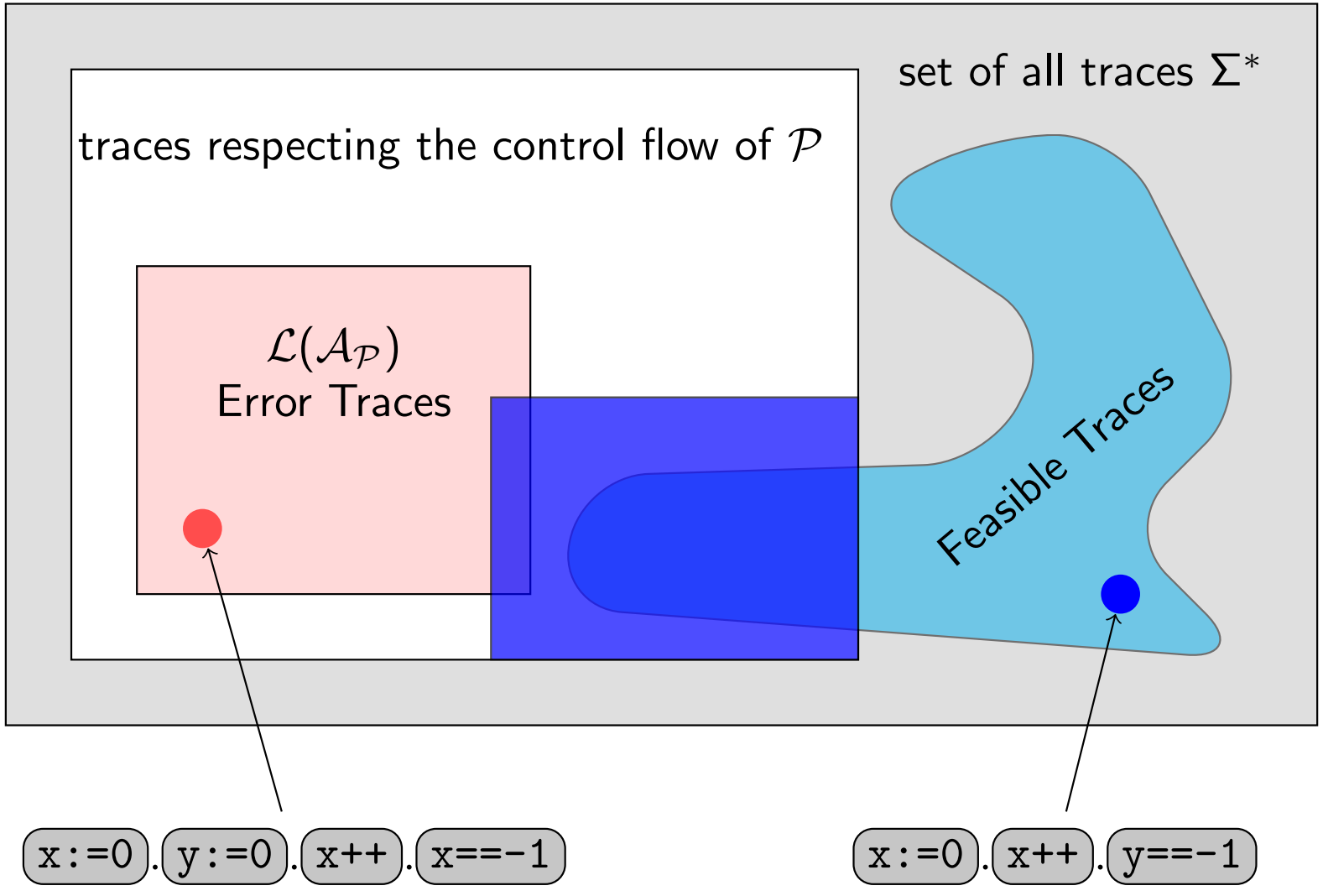
# Set Theoretic View of Trace Abstraction



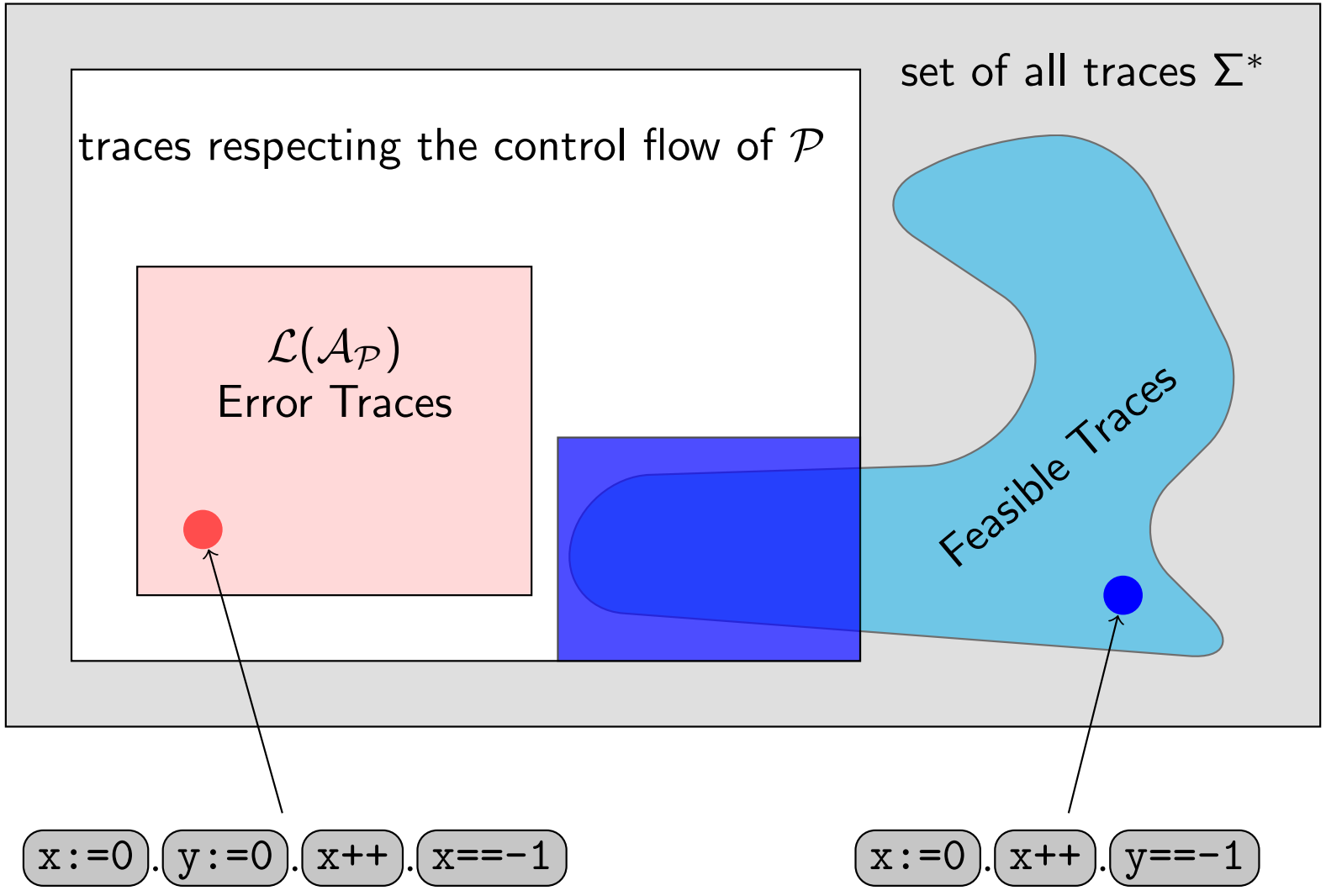
# Set Theoretic View of Trace Abstraction



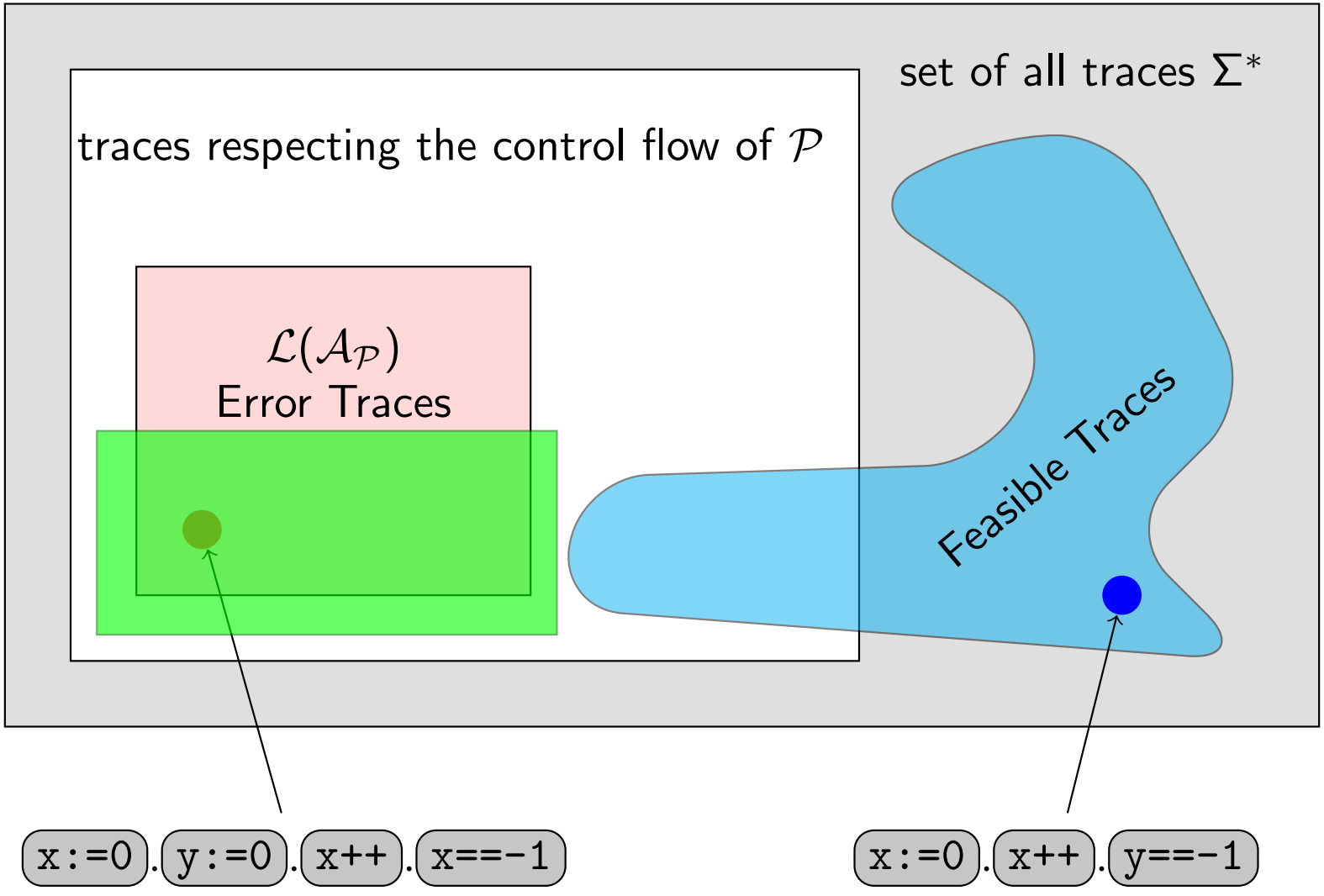
# Set Theoretic View of Trace Abstraction



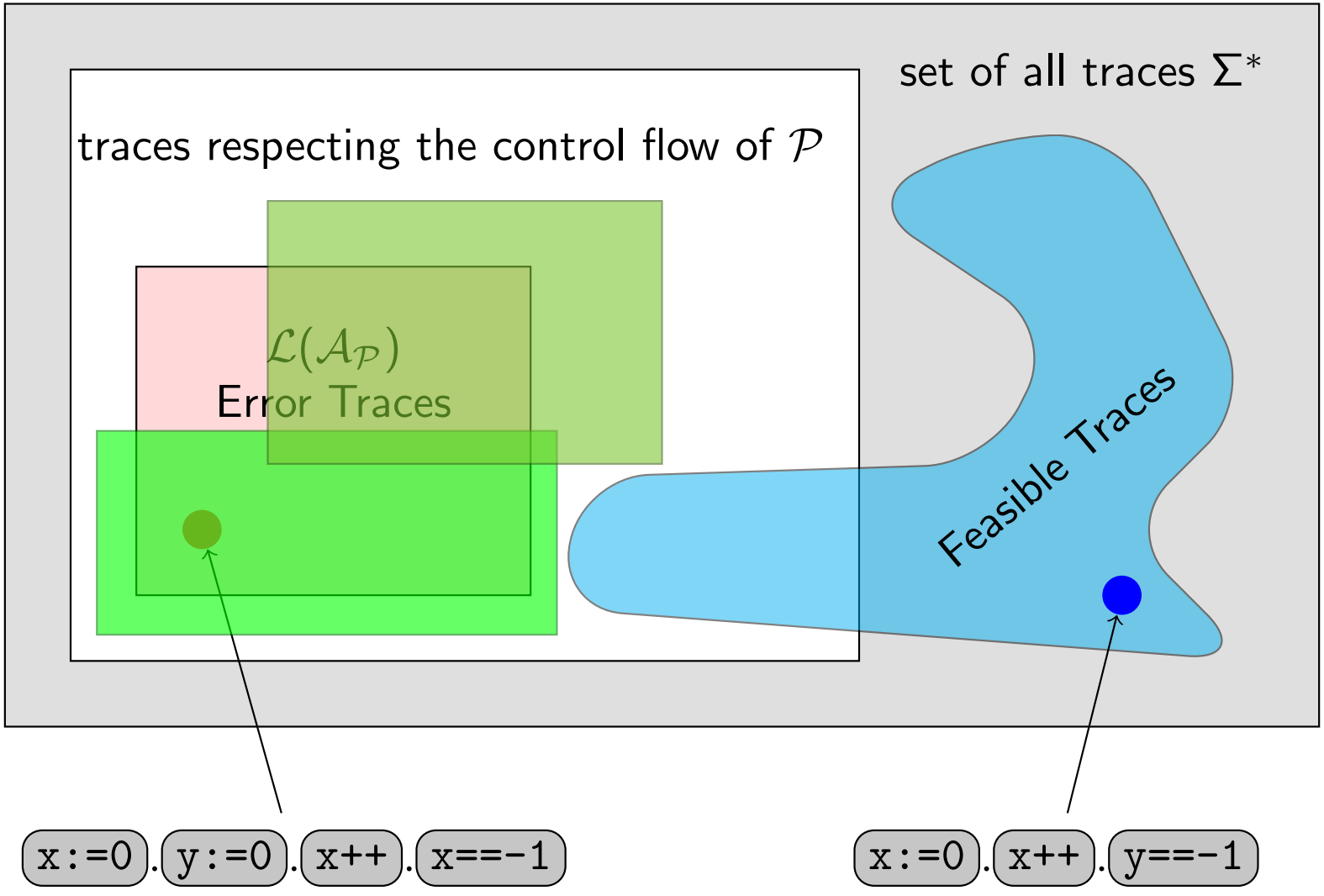
# Set Theoretic View of Trace Abstraction



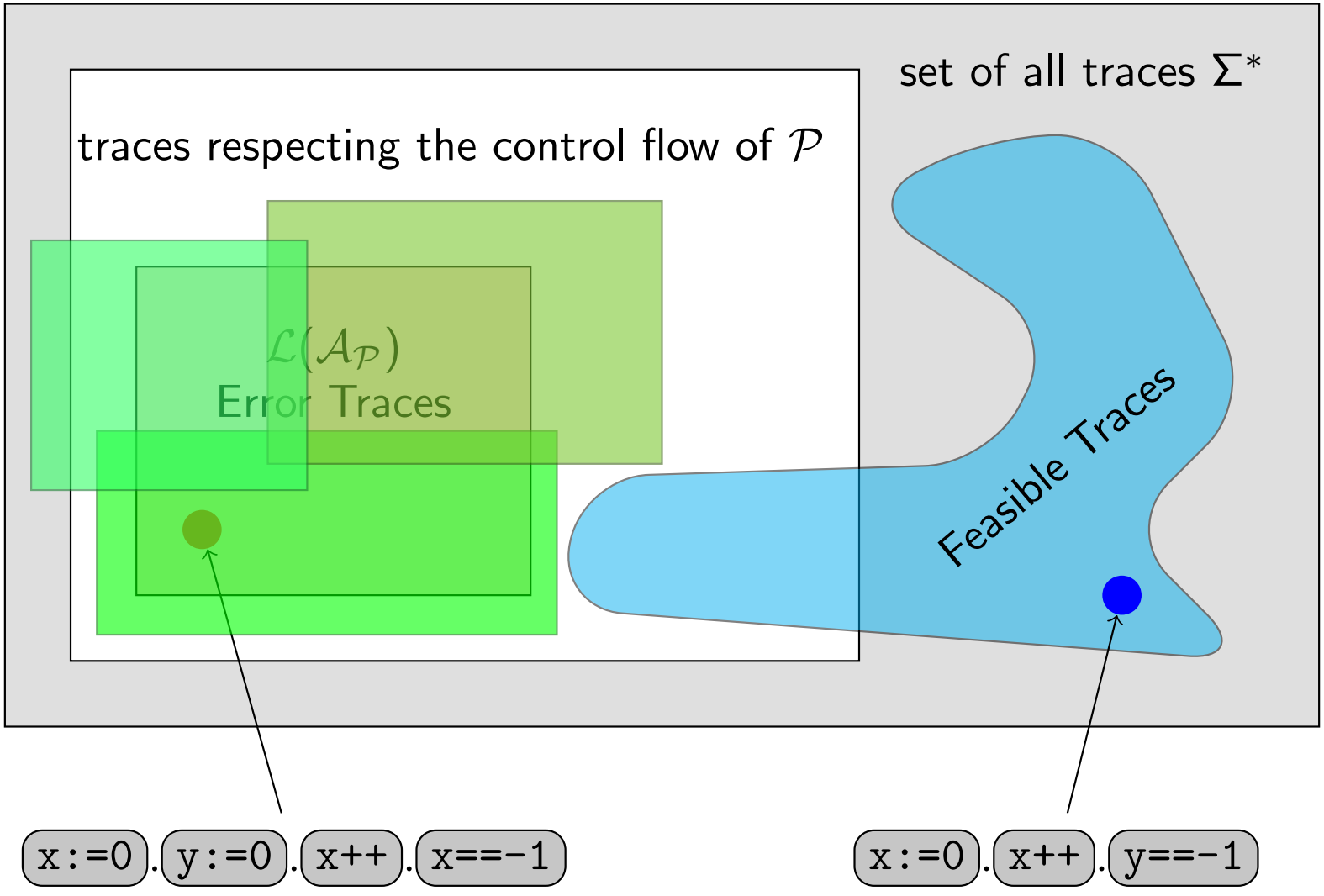
# Set Theoretic View of Trace Abstraction



# Set Theoretic View of Trace Abstraction



# Set Theoretic View of Trace Abstraction



# Trace Abstraction

## Definition (Trace Abstraction)

A *trace abstraction* is given by a tuple of automata $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ such that each $\mathcal{A}_i$ recognizes a subset of infeasible traces, for $i = 1, \ldots, n$.

We say that *the trace abstraction* $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ *does not admit an error trace* if $\mathcal{A}_\mathcal{P} \cap \overline{\mathcal{A}_1} \cap \ldots \cap \overline{\mathcal{A}_n}$ is empty.

# Trace Abstraction

## Definition (Trace Abstraction)

A *trace abstraction* is given by a tuple of automata $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ such that each $\mathcal{A}_i$ recognizes a subset of infeasible traces, for $i = 1, \ldots, n$.

We say that *the trace abstraction* $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ *does not admit an error trace* if $\mathcal{A}_\mathcal{P} \cap \overline{\mathcal{A}_1} \cap \ldots \cap \overline{\mathcal{A}_n}$ is empty.

## Theorem (Soundness)

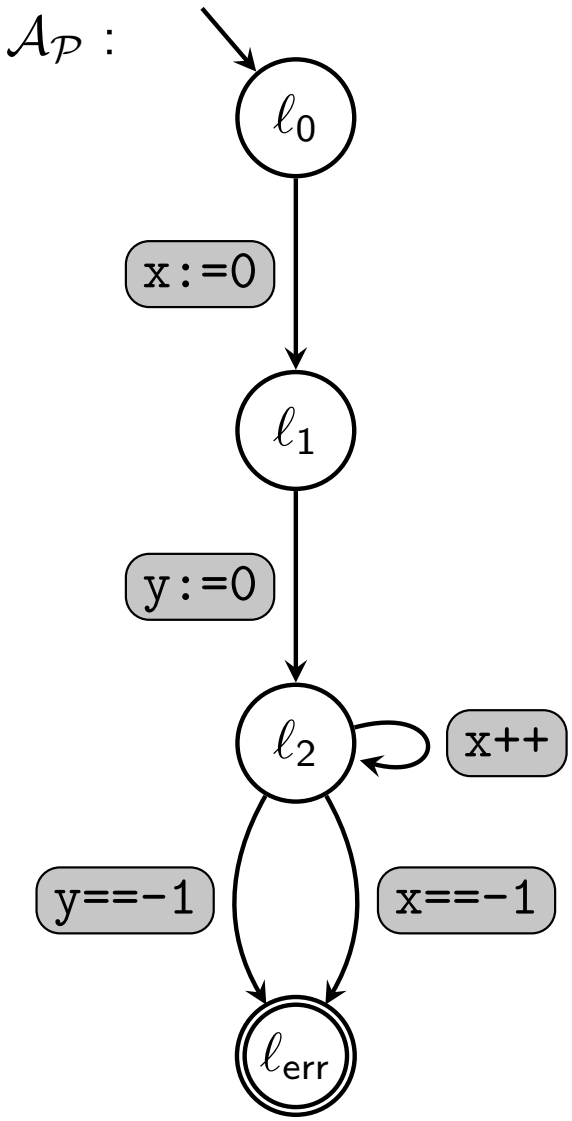
$$\mathcal{L}(\mathcal{A}_\mathcal{P} \cap \overline{\mathcal{A}_1} \cap \ldots \cap \overline{\mathcal{A}_n}) = \emptyset \quad \Rightarrow \quad \mathcal{P} \text{ is correct}$$

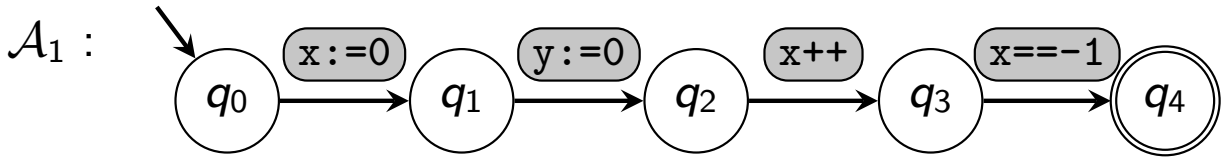
## Theorem (Completeness)

*If $\mathcal{P}$ is correct, there is a trace abstraction $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ such that*

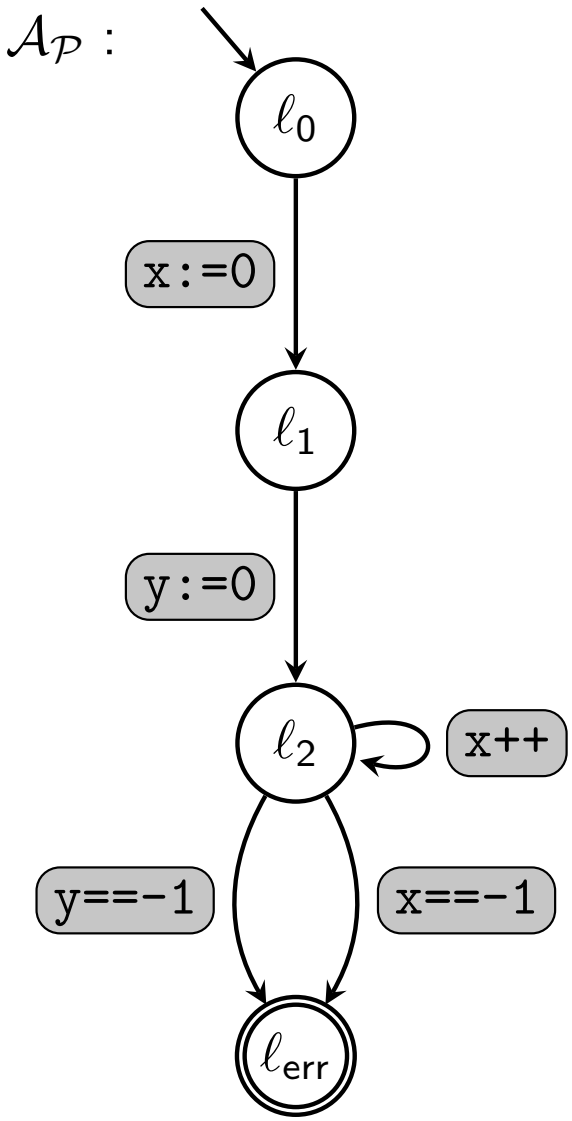
$$\mathcal{L}(\mathcal{A}_\mathcal{P} \cap \overline{\mathcal{A}_1} \cap \ldots \cap \overline{\mathcal{A}_n}) = \emptyset$$

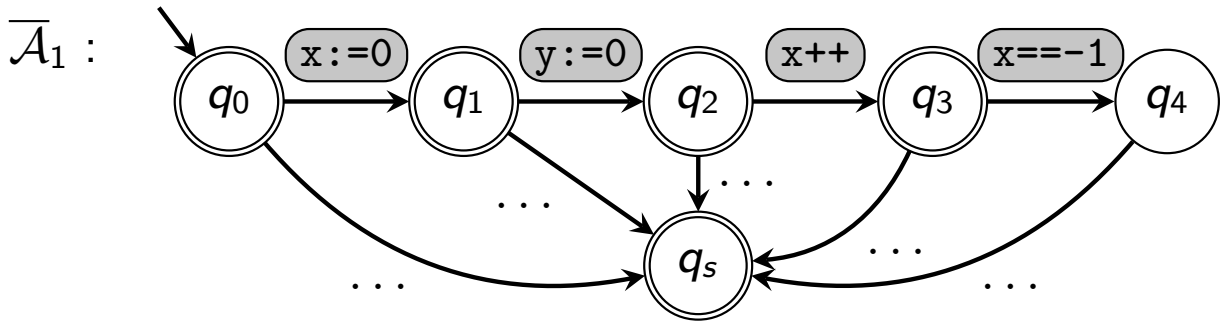
# Example – Exclude an Infeasible Trace



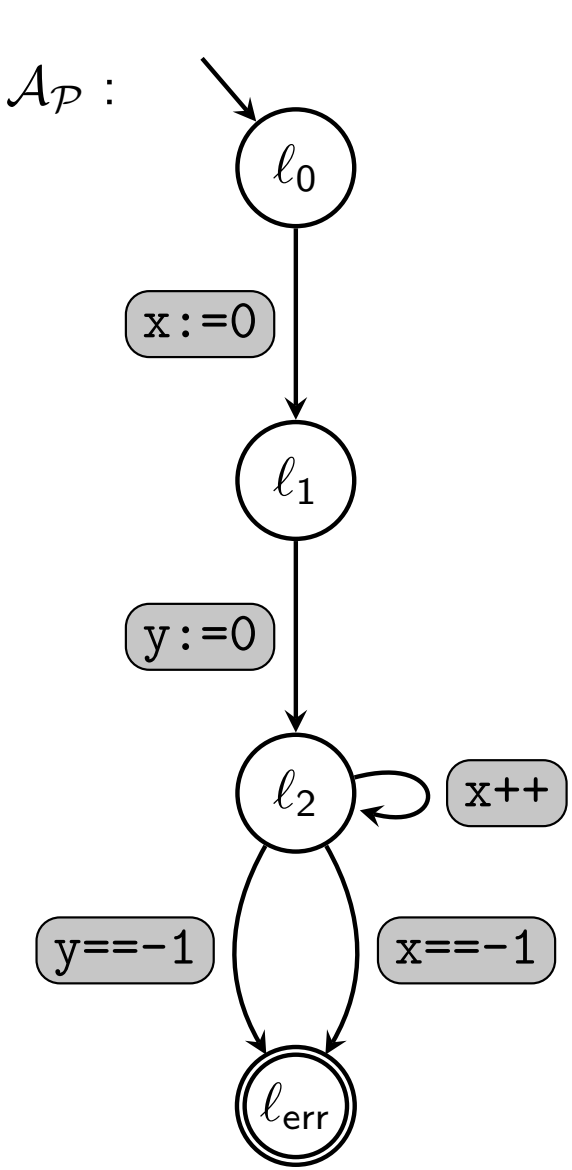



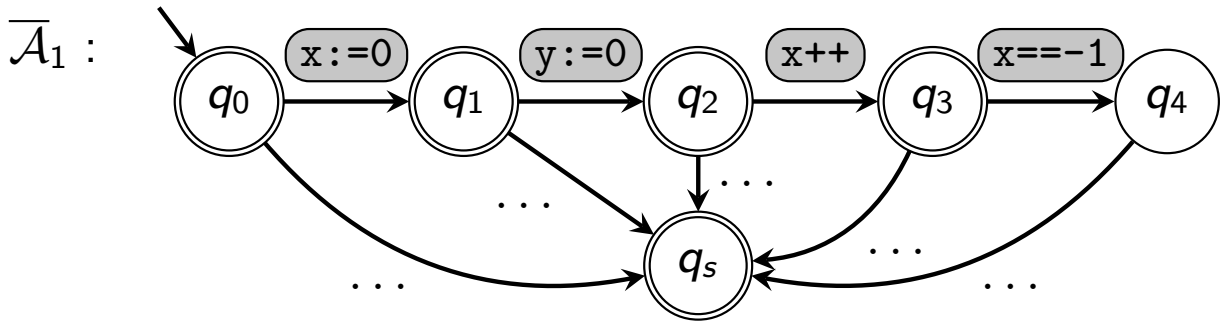
# Example – Exclude an Infeasible Trace

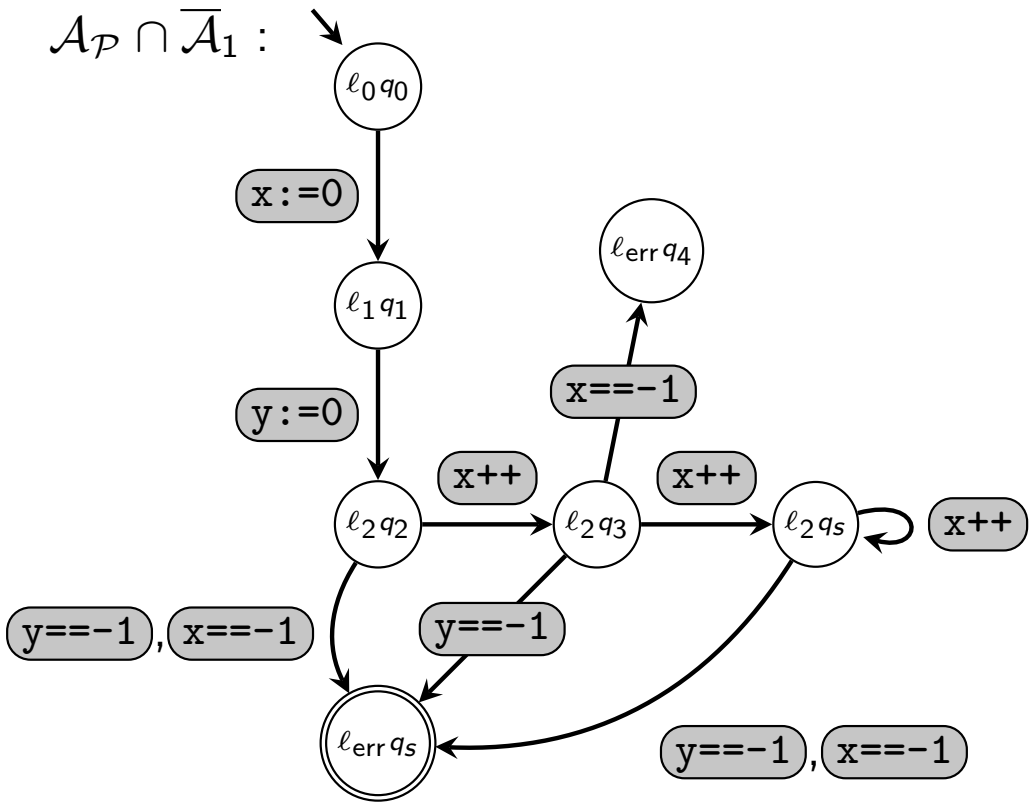


# Example – Exclude an Infeasible Trace



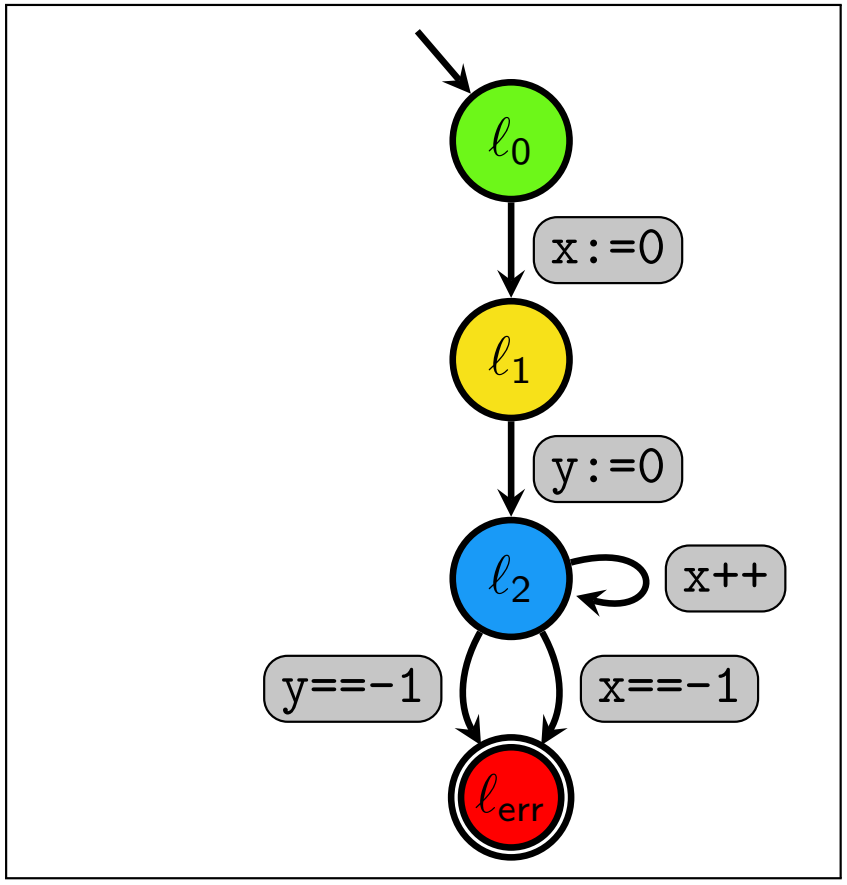
# Control flow as finite automaton

$\ell_0$: `x:=0`

$\ell_1$: `y:=0`

$\ell_2$: `while(nondet) {x++}`
`assert x!= -1`
`assert y!= -1`

Example program $\mathcal{P}$




Control flow graph of $\mathcal{P}$

# Floyd-Hoare proof as finite automaton

$\{\top\}$

$\ell_0$: `x:=0`

$\{x \geq 0\}$

$\ell_1$: `y:=0`

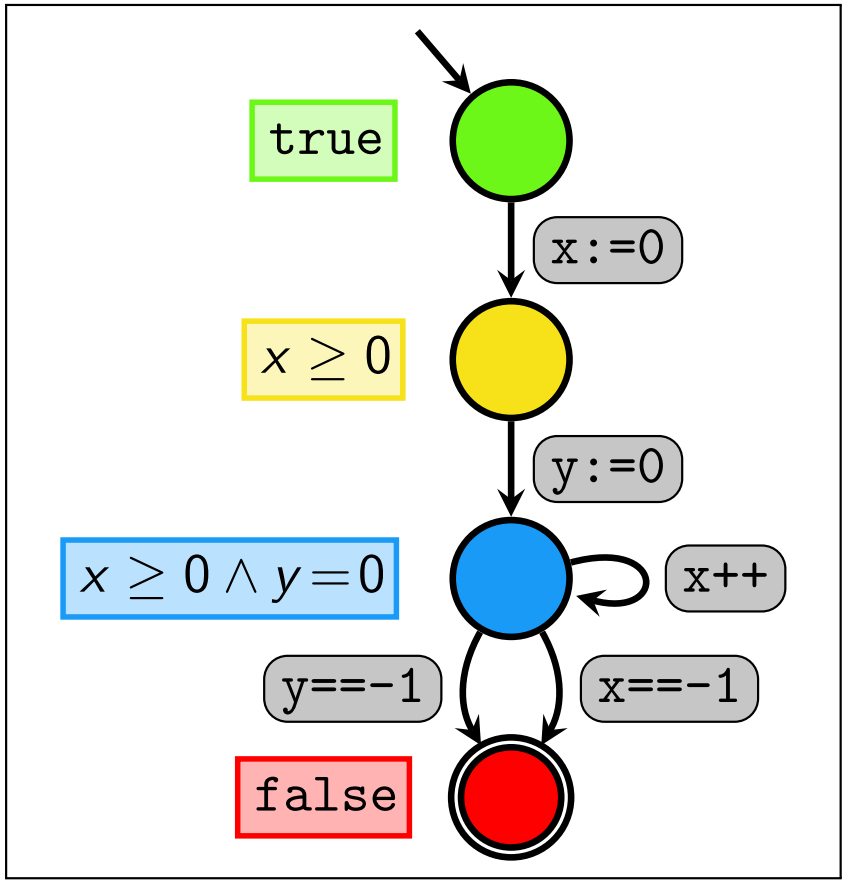
$\{x \geq 0 \wedge y = 0\}$

$\ell_2$: `while(nondet) {x++}`

`assert x!= -1`

`assert y!= -1`

Example program $\mathcal{P}$




Control flow graph of $\mathcal{P}$

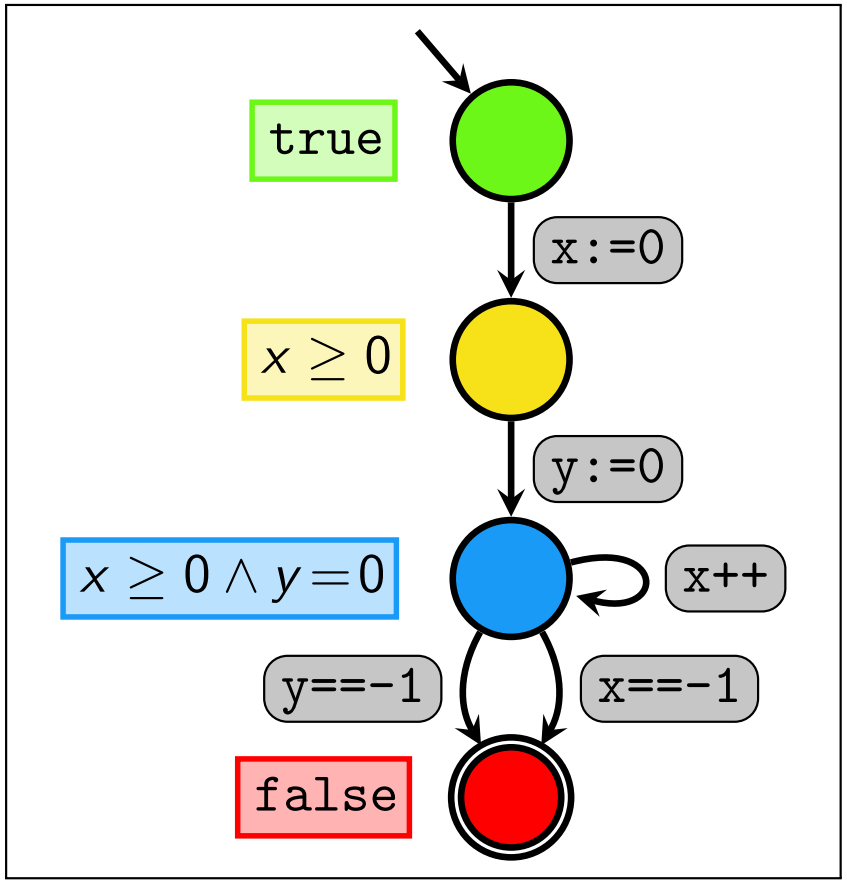
# Floyd-Hoare proof as finite automaton

```
ℓ0: x:=0

ℓ1: y:=0

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

Example program $\mathcal{P}$




Control flow graph of $\mathcal{P}$

Observation: Every transition is related to a Hoare triple!

e.g. ( ● , y:=0 , ● ) $\in \delta$ $\quad post($ x ≥ 0 , y:=0 $) \subseteq$ x ≥ 0 ∧ y = 0

# Interpolant Automata

Given: Sequence of predicates $\mathcal{I} = I_0, I_1, \ldots, I_n$

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

$$\mathcal{A}_\mathcal{I} = \langle Q_\mathcal{I}, \delta_\mathcal{I}, Q_\mathcal{I}^{\text{init}}, Q_\mathcal{I}^{\text{fin}} \rangle \qquad Q_\mathcal{I} = \{q_0, \ldots, q_n\}$$

$$(q_i, st, q_j) \in \delta_\mathcal{I} \quad \text{implies} \quad post(st, I_i) \subseteq I_j$$

$$q_i \in Q^{\text{init}} \quad \text{implies} \quad I_i = true$$

$$q_i \in Q^{\text{fin}} \quad \text{implies} \quad I_i = false$$

# Interpolant Automata

Given: Sequence of predicates $\mathcal{I} = I_0, I_1, \ldots, I_n$

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

$$\mathcal{A}_\mathcal{I} = \langle Q_\mathcal{I}, \delta_\mathcal{I}, Q_\mathcal{I}^{\text{init}}, Q_\mathcal{I}^{\text{fin}} \rangle \qquad Q_\mathcal{I} = \{q_0, \ldots, q_n\}$$

$$(q_i, st, q_j) \in \delta_\mathcal{I} \quad \text{implies} \quad post(st, I_i) \subseteq I_j$$

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

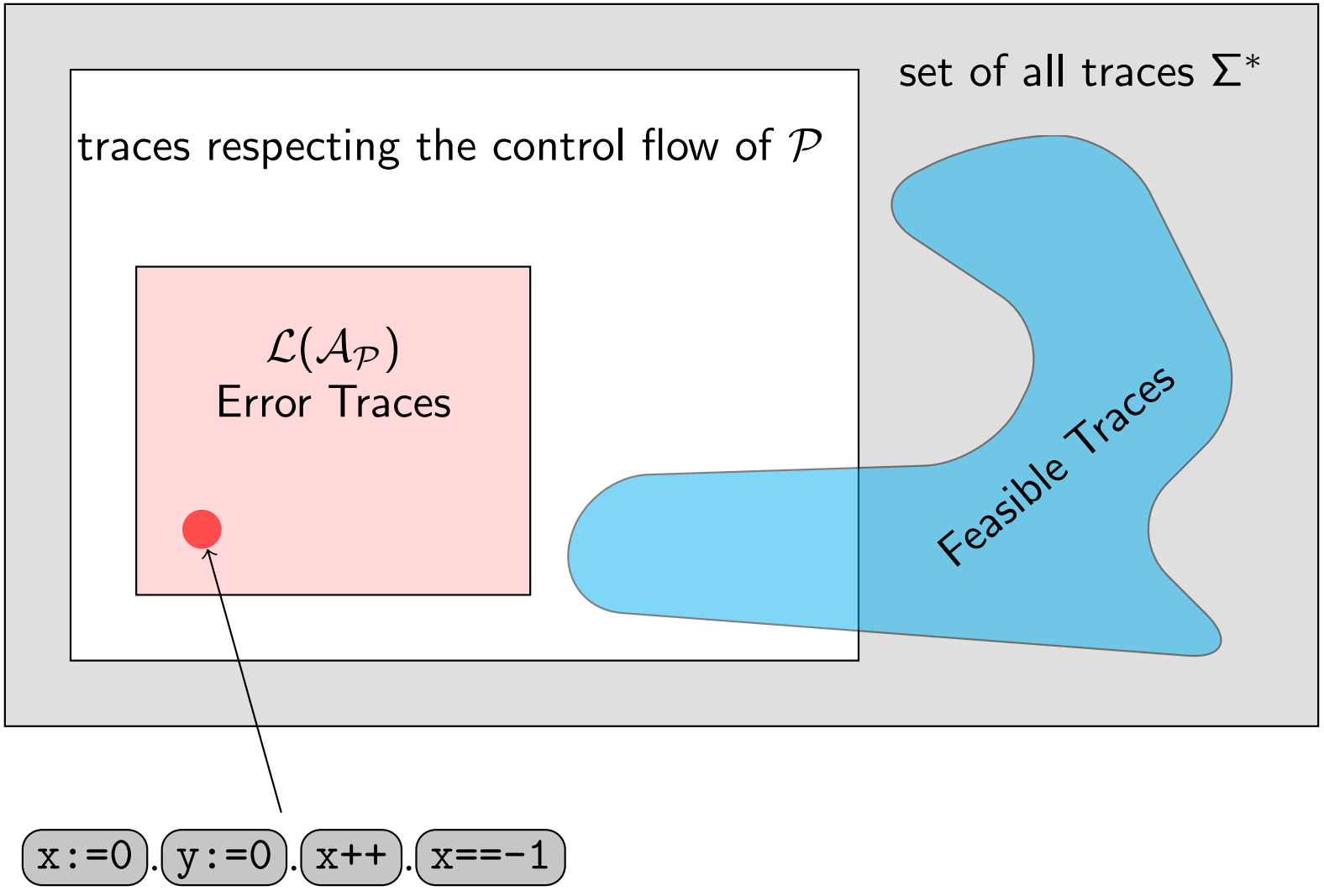
$$q_i \in Q^{\text{fin}} \quad \text{implies} \quad I_i = \mathit{false}$$

## Theorem

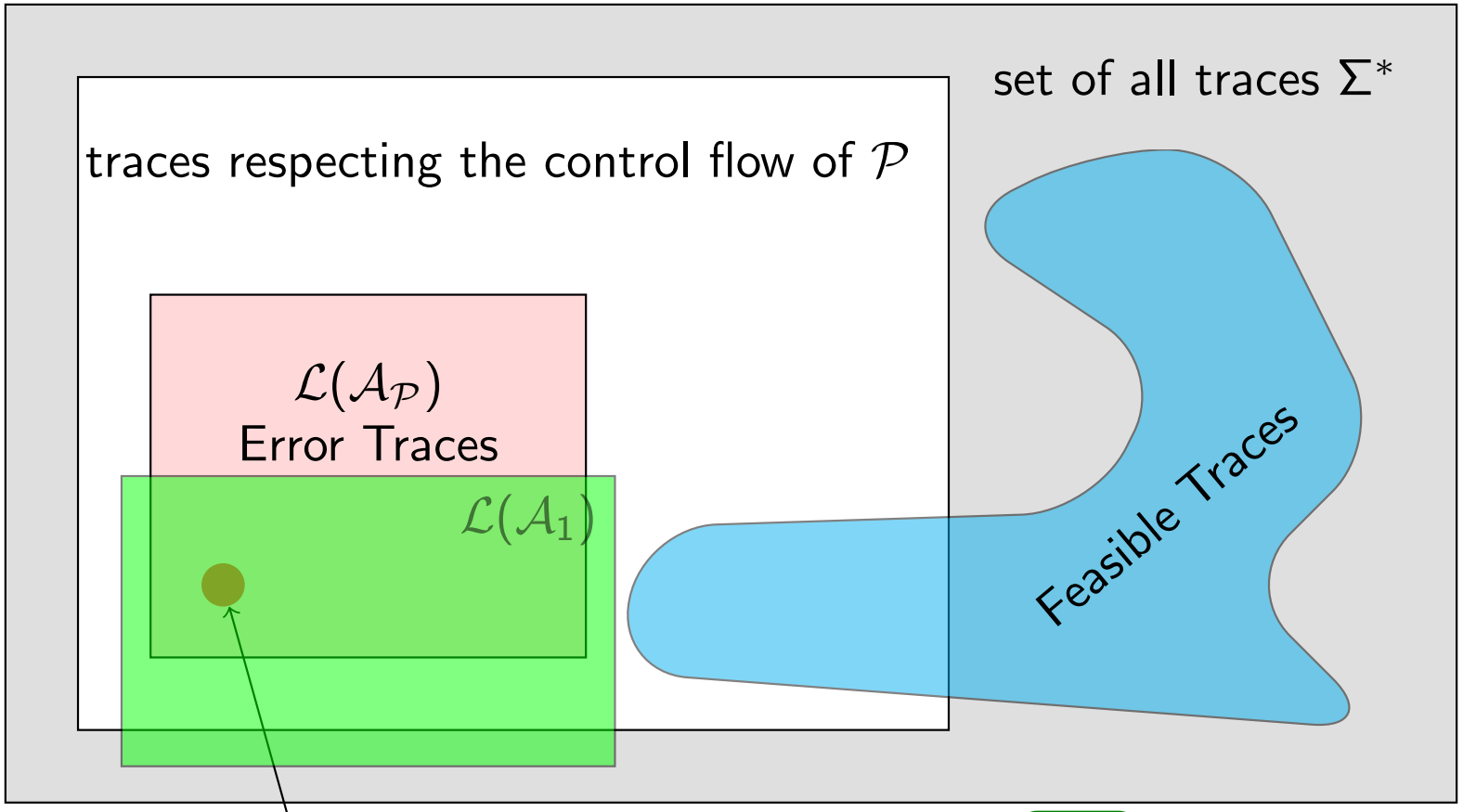
*An interpolant automaton $\mathcal{A}_\mathcal{I}$ recognizes a subset of infeasible traces.*

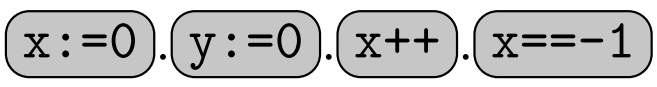
$$\mathcal{L}(\mathcal{A}_\mathcal{I}) \subseteq \mathit{Infeasible}$$

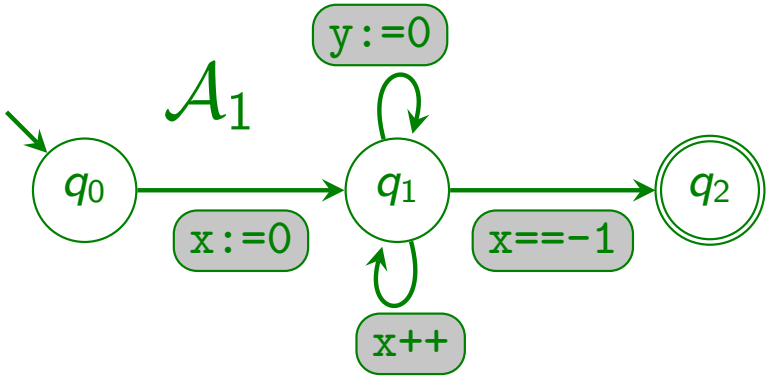
# Example – Refinement Using Interpolant Automata



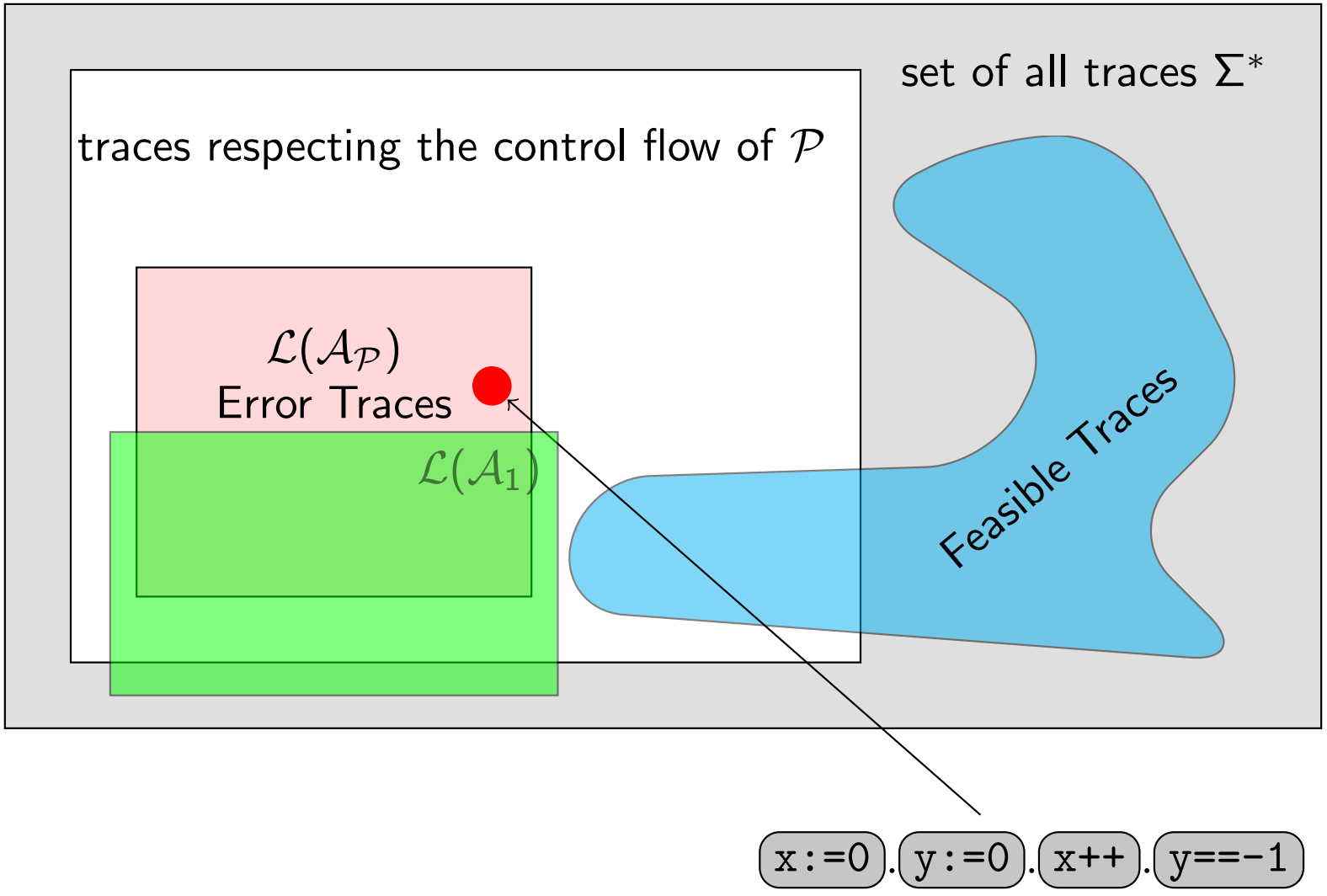
# Example – Refinement Using Interpolant Automata



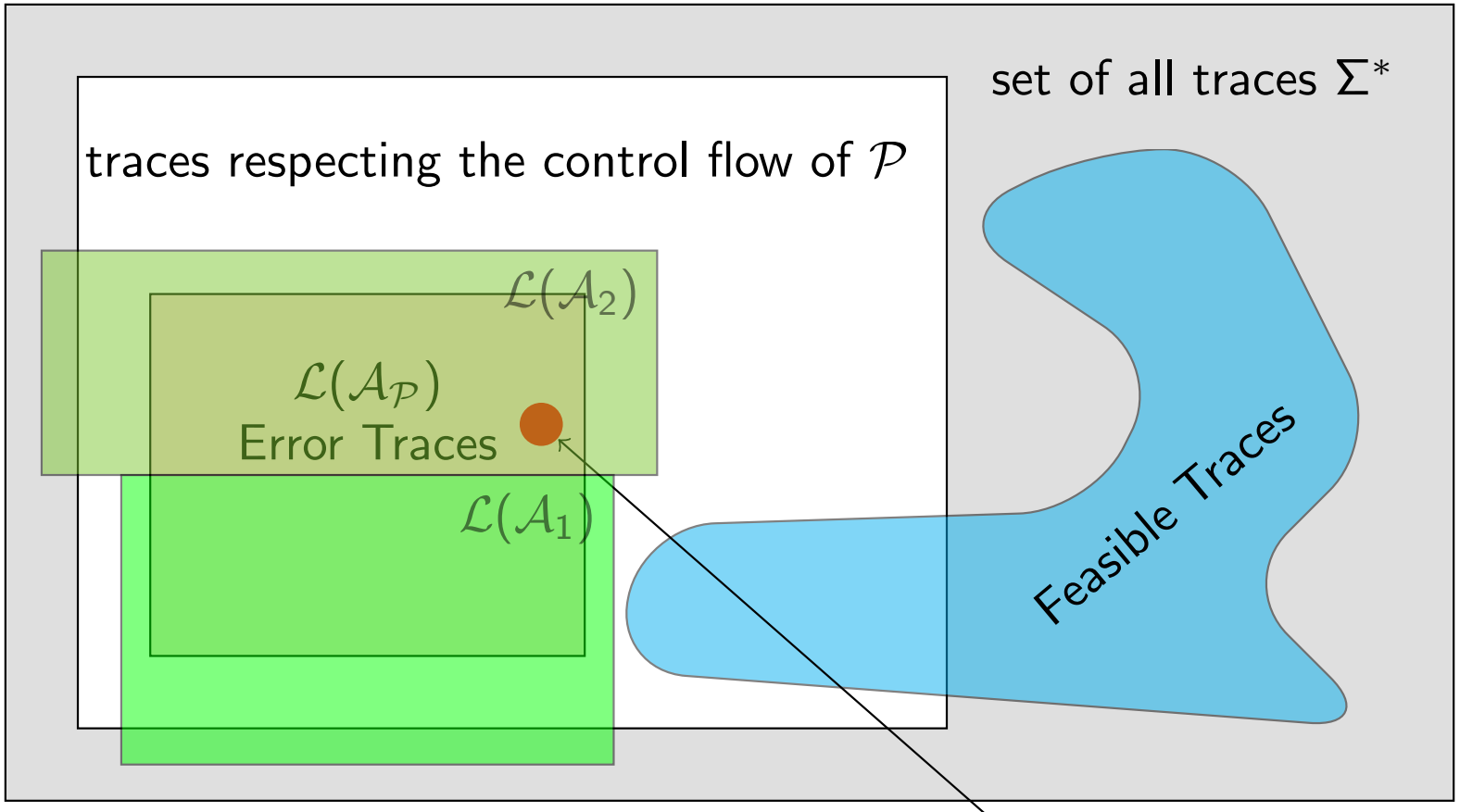



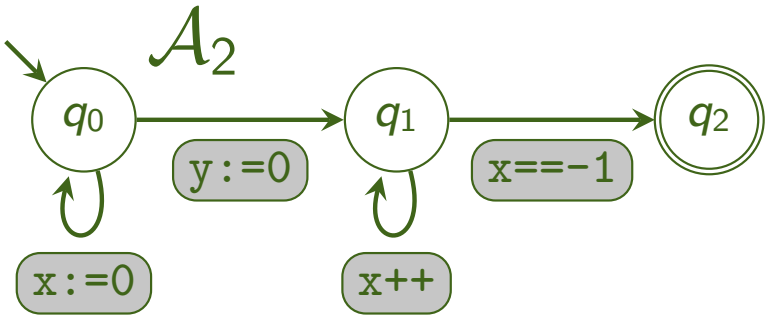



# Example – Refinement Using Interpolant Automata



# Example – Refinement Using Interpolant Automata




x:=0 . y:=0 . x++ . y==-1

# CEGAR for Trace Abstraction