

Search Modulo Theory

Andreas Podelski
University of Freiburg

```
l0: assume p != 0;
```

```
l1: while(n >= 0)
```

```
{
```

```
l2:
```

```
    if(n == 0)
```

```
    {
```

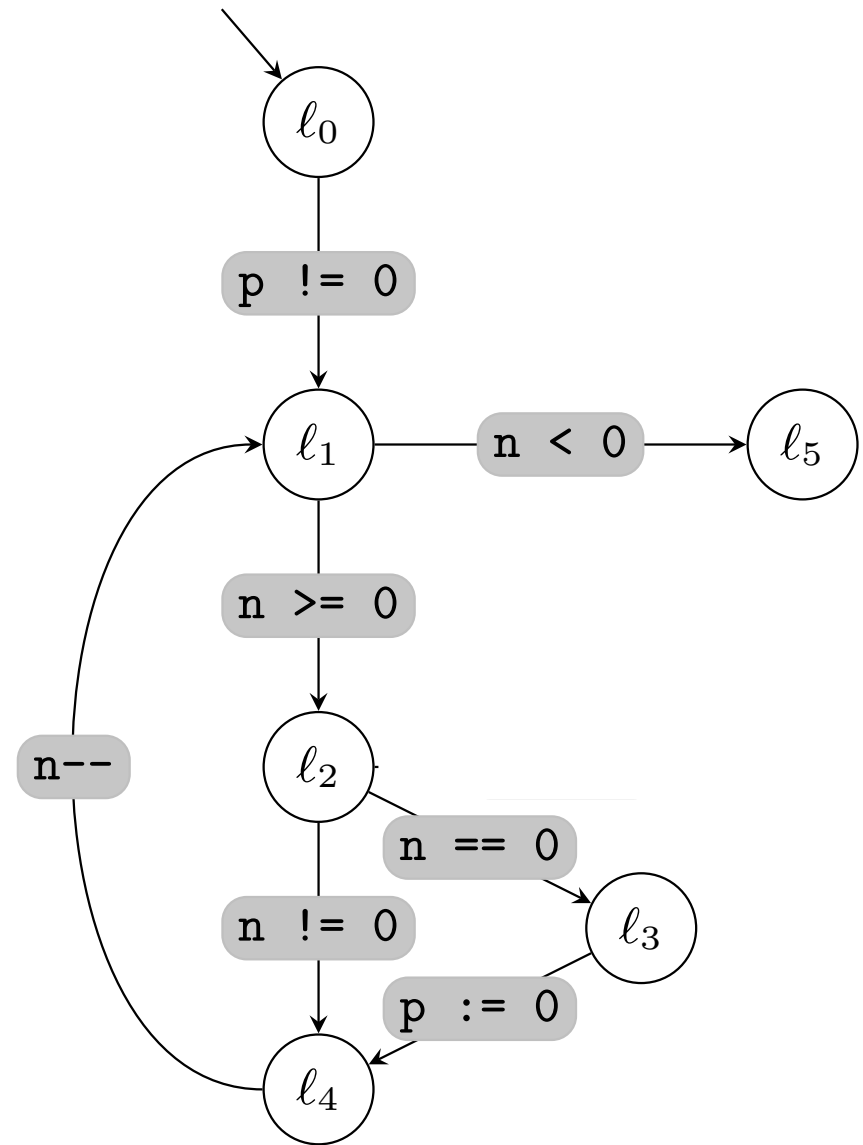
```
l3:        p := 0;
```

```
    }
```

```
l4:    n--;
```

```
}
```

```
l5:
```



```
l0: assume p != 0;
```

```
l1: while(n >= 0)
```

```
{
```

```
l2:   assert p != 0;
```

```
      if(n == 0)
```

```
      {
```

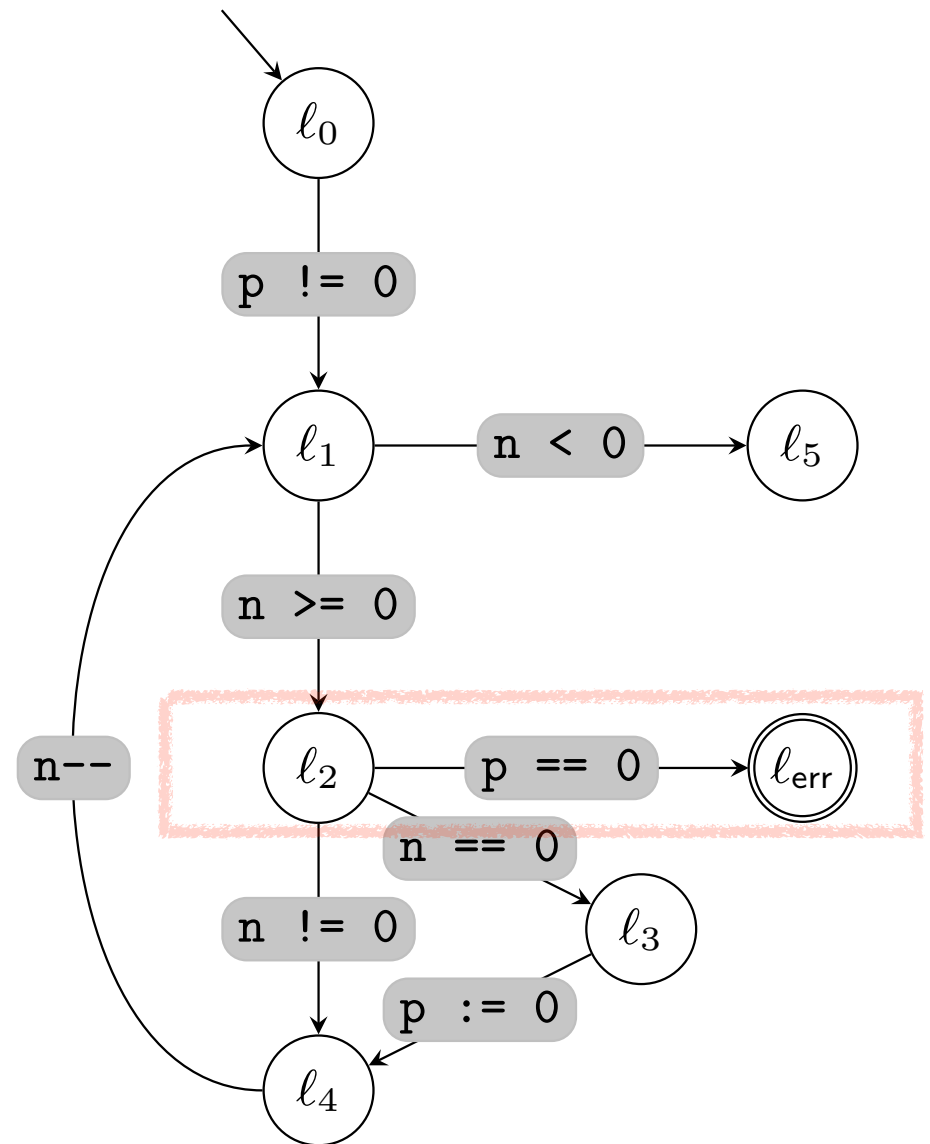
```
l3:   p := 0;
```

```
      }
```

```
l4:   n--;
```

```
}
```

```
l5:
```

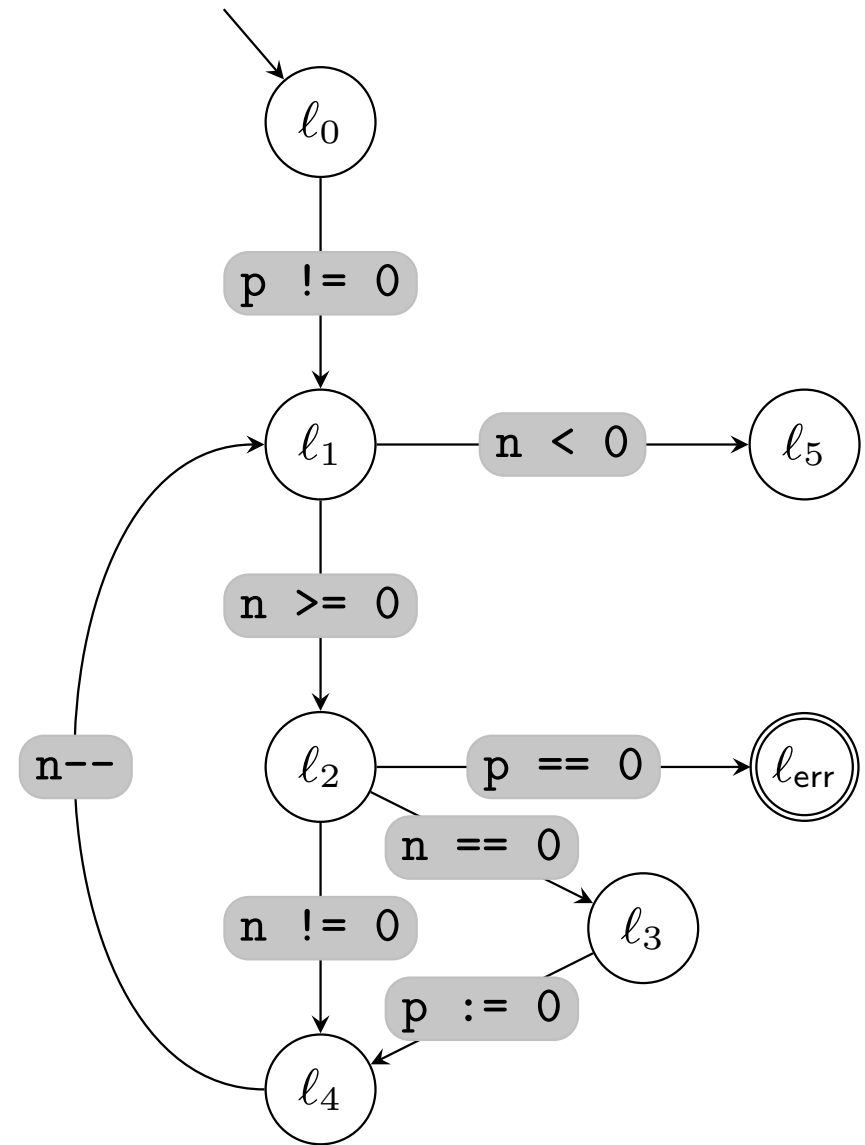


```

l0: assume p != 0;
l1: while(n >= 0)
  {
l2:   assert p != 0;

      if(n == 0)
      {
l3:     p := 0;
      }
l4:   n--;
  }
l5:

```



no execution violates assertion = no execution reaches error location

path in infinite state space of program:
feasible path in finite control flow graph

infinite search space:
symbolically by finite graph
(edges labeled by constraints and updates)

path in infinite search space:
feasible path in finite graph

feasibility = Satisfiability Modulo Theory (SMT)

infeasible trace

unsatisfiable formula

$x == 1 ; x == -1 ;$

$x = 1 \wedge x = -1$

infeasible/unsatisfiable ... *Modulo Theory*

infeasible trace

$x == 1 ; x == -1 ;$

$x := 1 ; x == -1 ;$

unsatisfiable *Modulo Theory*

$x = 1 \wedge x = -1$

$x' = 1 \wedge x' = -1$

Automated Program Verification

Andreas Podelski
University of Freiburg

joint work with

Matthias Heizmann and **Jochen Hoenicke**
University of Freiburg

Azadeh Farzan and **Zachary Kincaid**
University of Toronto

automaton

|ô'tämətən|

automation

|,ô'tə'māSHən|

program \mathcal{P}

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

1. $w \in \mathcal{A}_{n+1}$
2. $\mathcal{A}_{n+1} \subseteq \{ \text{infeasible traces} \}$

$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n ?$

w infeasible?

yes

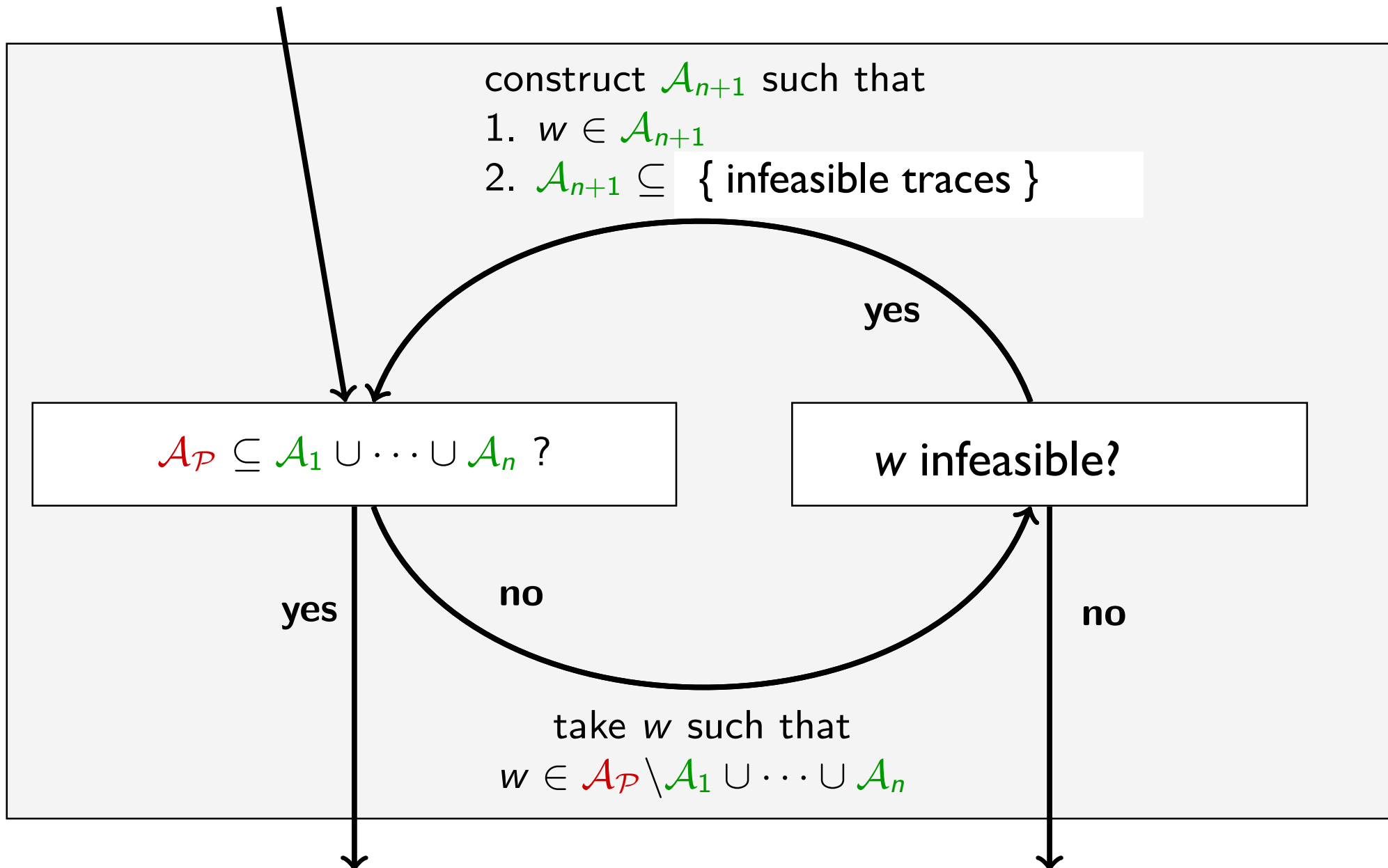
no

no

take w such that
 $w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n$

\mathcal{P} is correct

\mathcal{P} is incorrect



Ultimate Automizer

The screenshot shows the Ultimate Automizer web interface in a browser window. The browser title is "Uni-Freiburg : SWT - Ultimate - rekonq" and the address bar shows the URL "https://monteverdi.informatik.uni-freiburg.de/tomcat//Website/?task=VerifyC#". The main heading is "ULTIMATE WEB-INTERFACE".

On the left side, there are configuration options:

- Task:** Verify C
- Sample:** McCarthy91.c
- Tool:** Trace Abstraction

Below these is a "SETTINGS" section and a large blue "EXECUTE" button with a right-pointing arrow.

The central code editor displays the following code:

```
12 /*@ requires \true;
i 13  @ ensures x > 101 || \result == 91;
14  @*/
i 15  int f91(int x);
16
i 17  int f91(int x) {
18    if (x > 100)
19      return x -10;
20  else {
i 21    return f91(f91(x+11));
22  }
23 }
24
25
26
```

At the bottom left, there is a "Show editor fullscreen" link and a "Choose File" button with the text "No file selected".

At the bottom right, there is a table with the following data:

	Line	Ultimate Result
	21	procedure precondition always holds
	21	procedure precondition always holds
	13	procedure postcondition always holds

program = automaton

constructed from proof

proof by *SMT solver*

search *Modulo Theory*

add lemmas to prune the search space
lemmas inferred from proofs of *SMT* solver

SMT: Satisfiability Modulo Theory

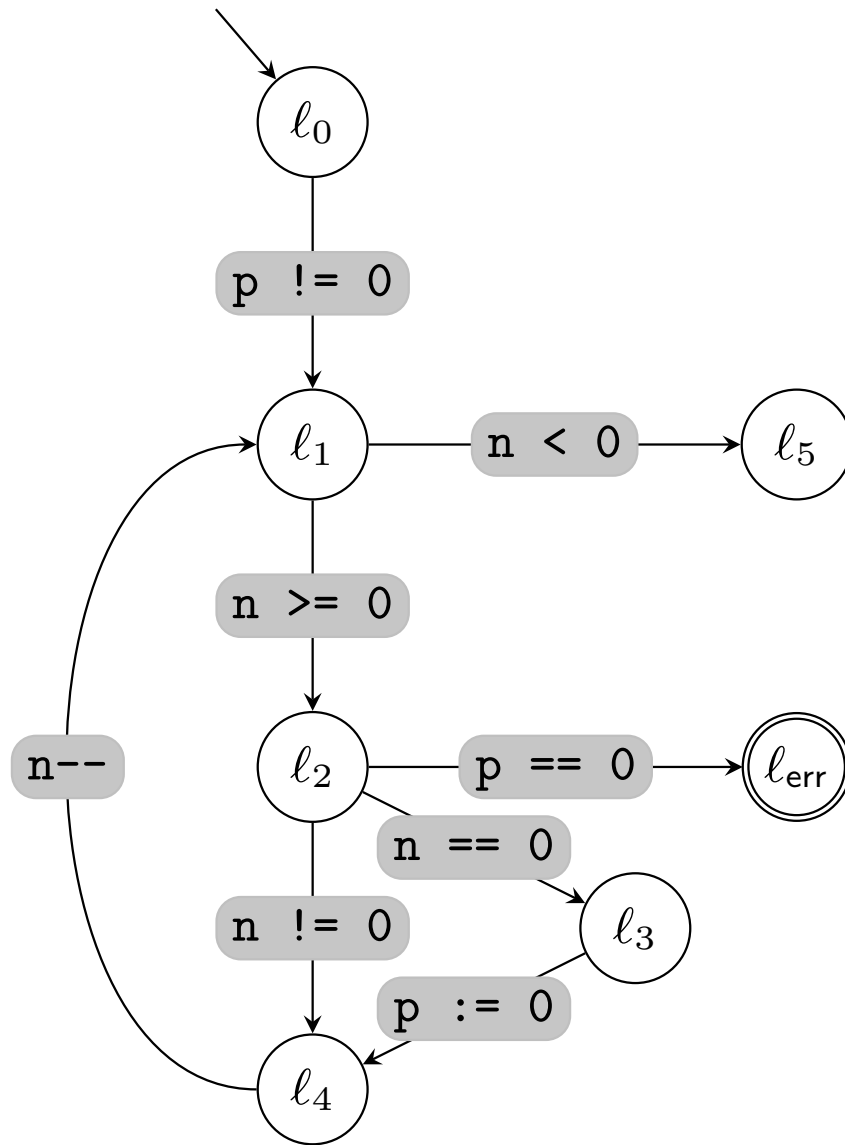
search *Modulo Theory*

add lemmas to prune the search space
lemmas inferred from proofs of *SMT* solver

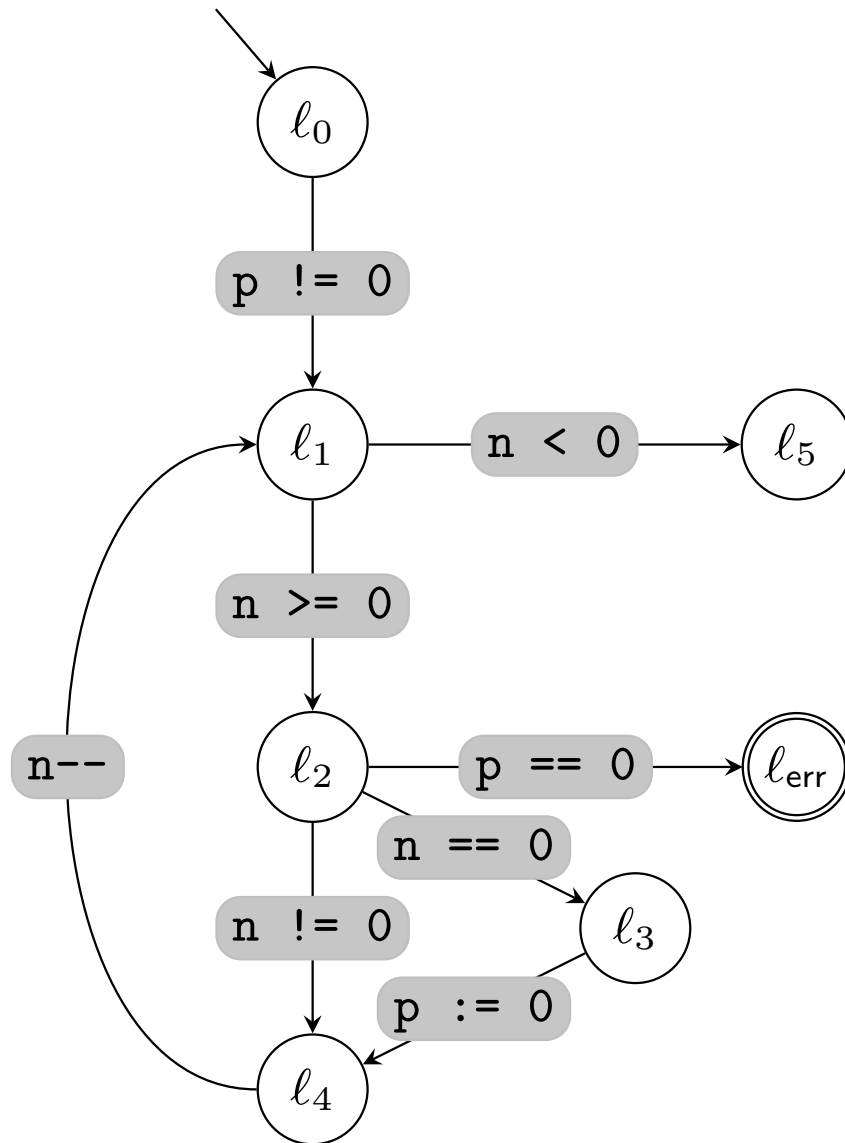
lemmas are automata (sets of paths)
automata constructed from proofs of *SMT* solver

SMT: Satisfiability Modulo Theory

error trace: sequence of statements along path to error location

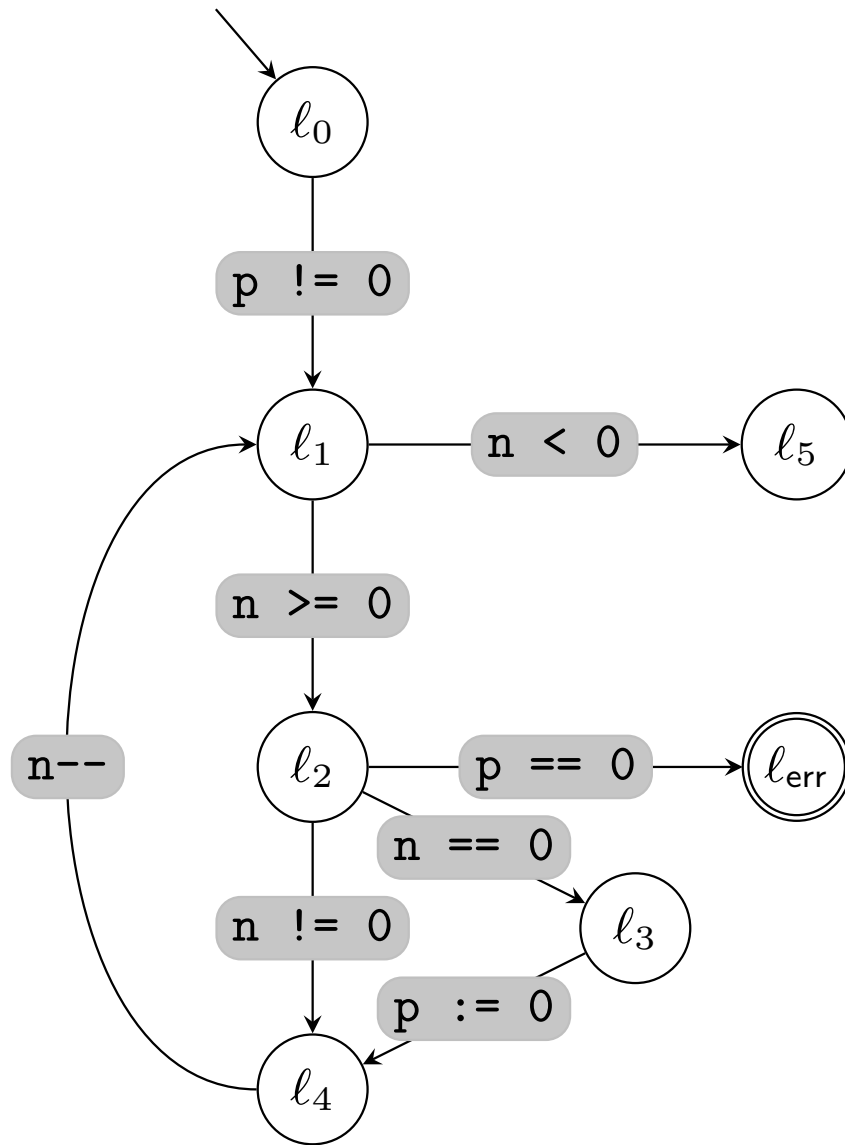


error trace: sequence of statements along path to error location



error trace: word accepted by **program automaton**

program correct = no execution reaches error location



program correct = no **feasible** error trace

program \mathcal{P}

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

1. $w \in \mathcal{A}_{n+1}$
2. $\mathcal{A}_{n+1} \subseteq \{ \text{infeasible traces} \}$

$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n ?$

w infeasible?

yes

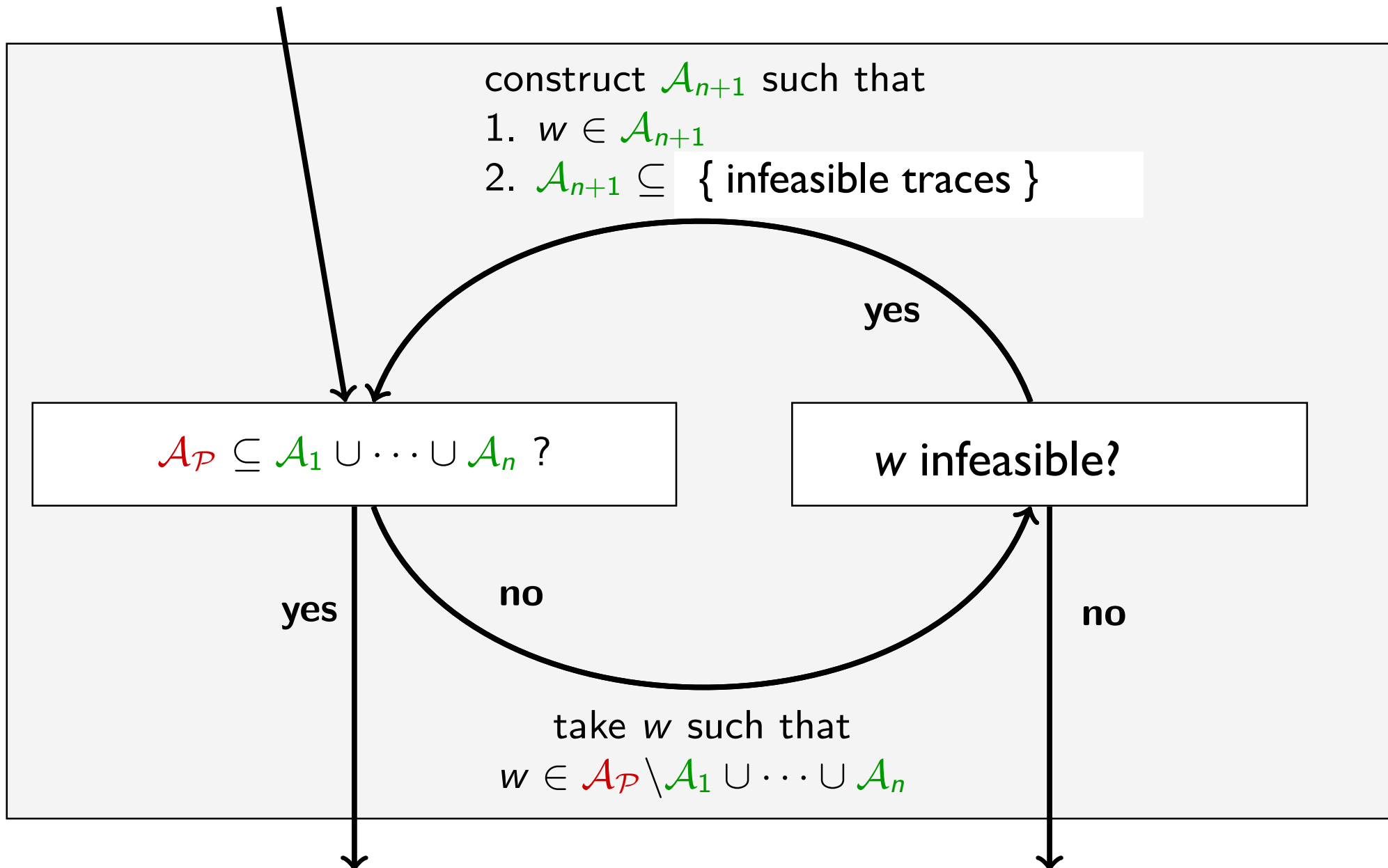
no

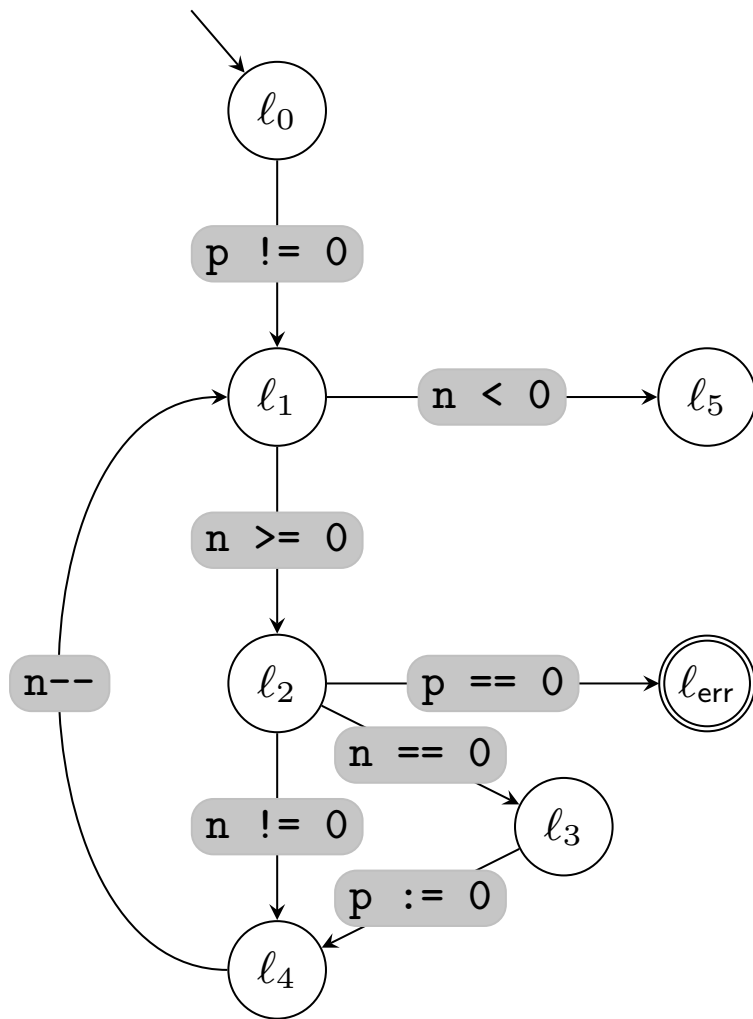
no

take w such that
 $w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n$

\mathcal{P} is correct

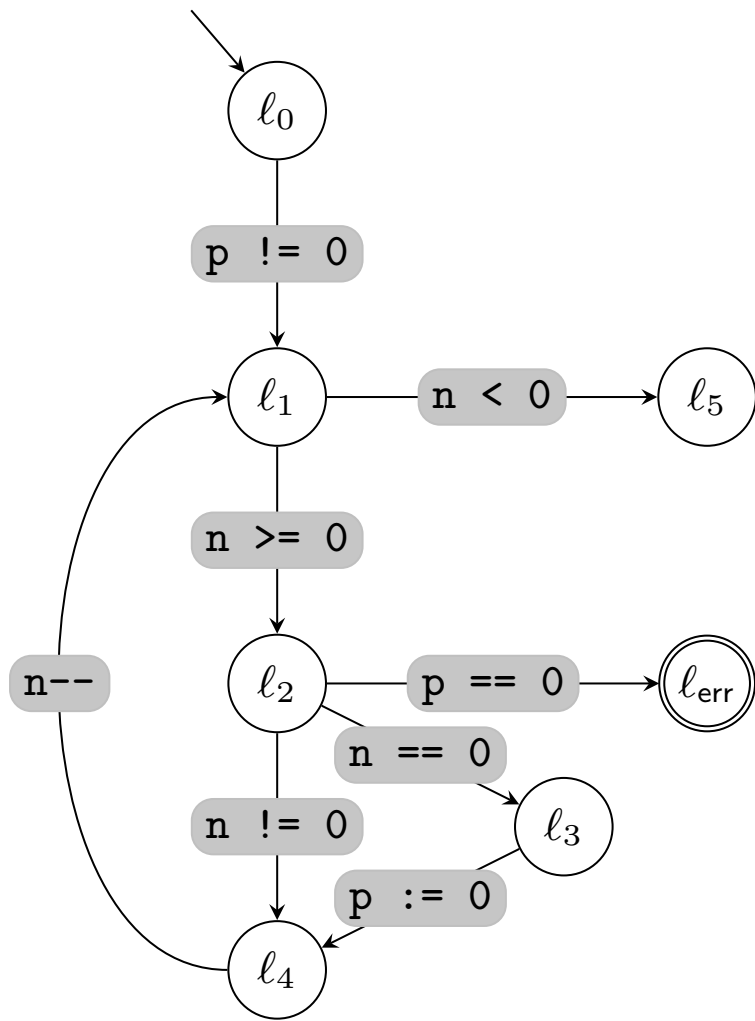
\mathcal{P} is incorrect



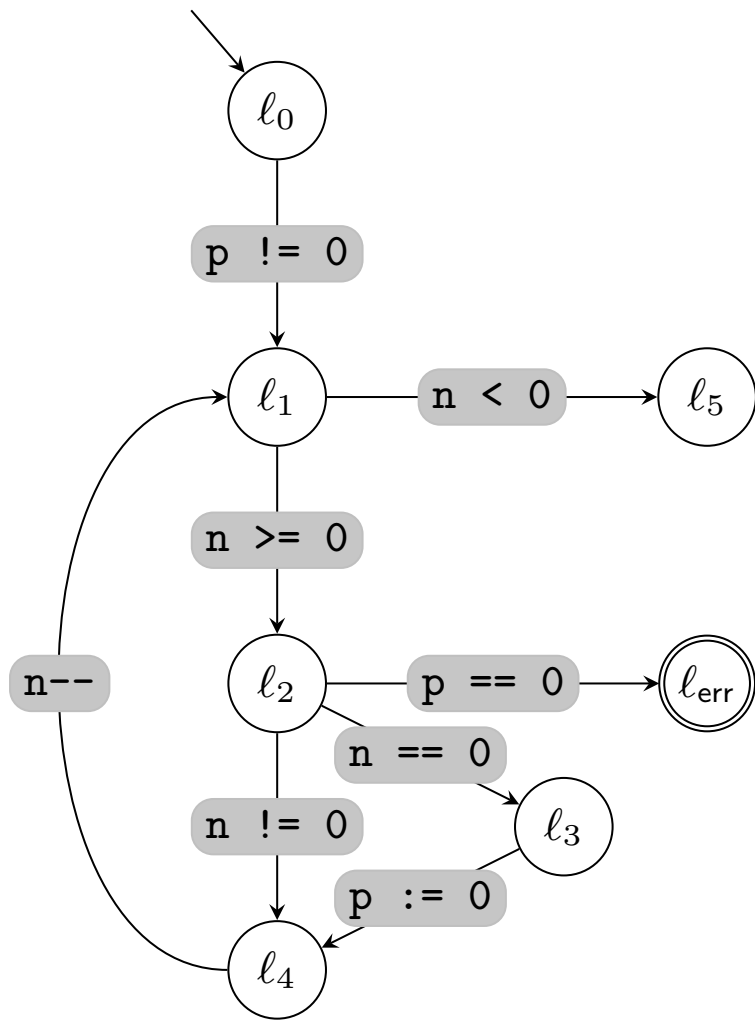


Does there exist an execution that leads to error state?

Does there exist a **feasible** error trace?

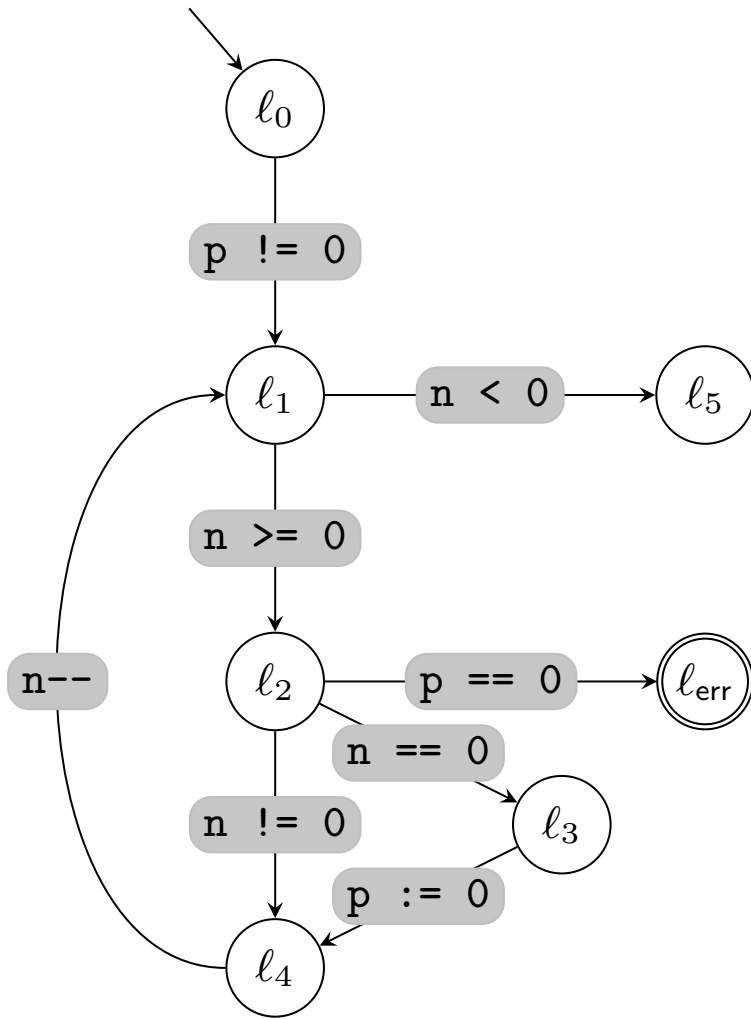


complex control? - just ignore it!



$(p \neq 0)$
 $(n \geq 0)$
 $(p = 0)$

error trace: sequence of statements along an error path



$(p \neq 0)$
 $(n \geq 0)$
 $(p = 0)$

$(p \neq 0)$
 $(p = 0)$

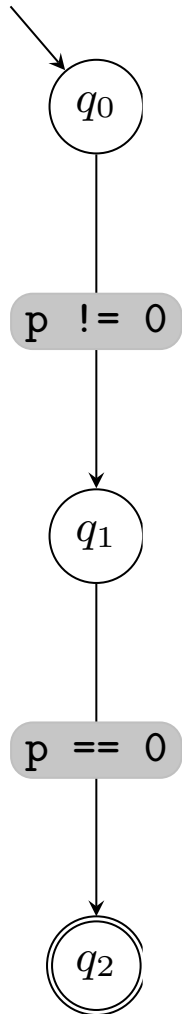
trace **infeasible**, formula **unsatisfiable**,

take **unsatisfiable core**

$(p \neq 0)$

$(p == 0)$

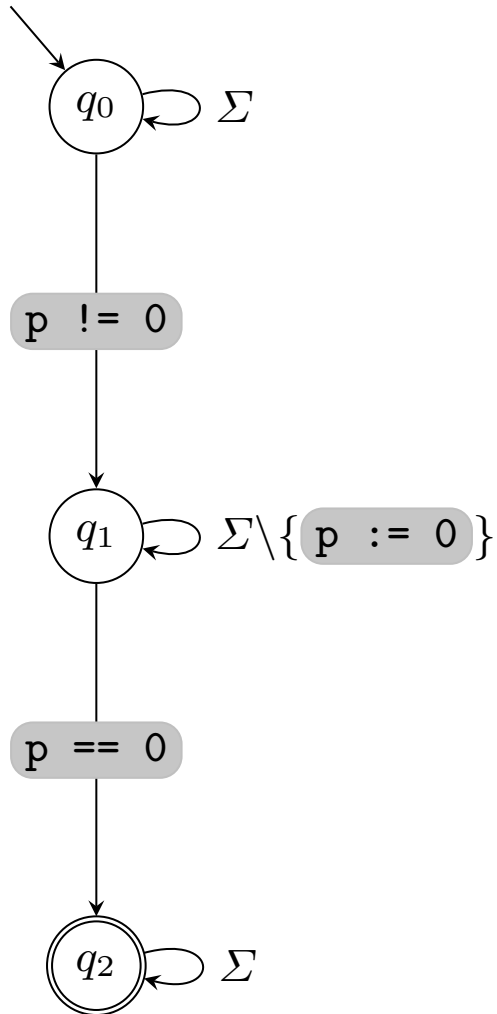
unsatisfiable core



$(p \neq 0)$

$(p == 0)$

construct automaton from unsatisfiable core (step I)

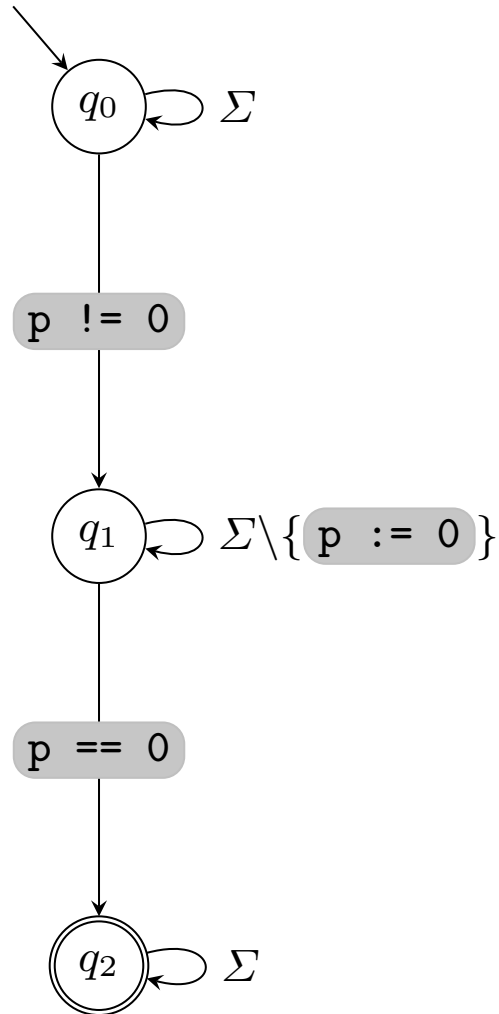


$(p \neq 0)$

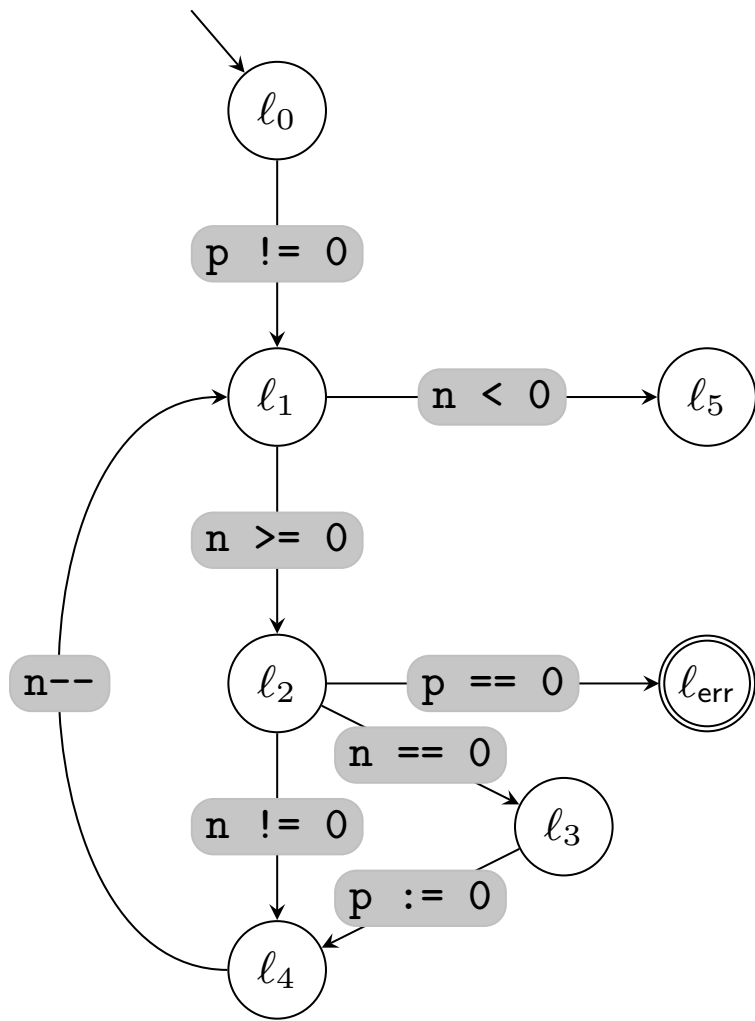
$(p == 0)$

construct automaton from unsatisfiable core (step 2)

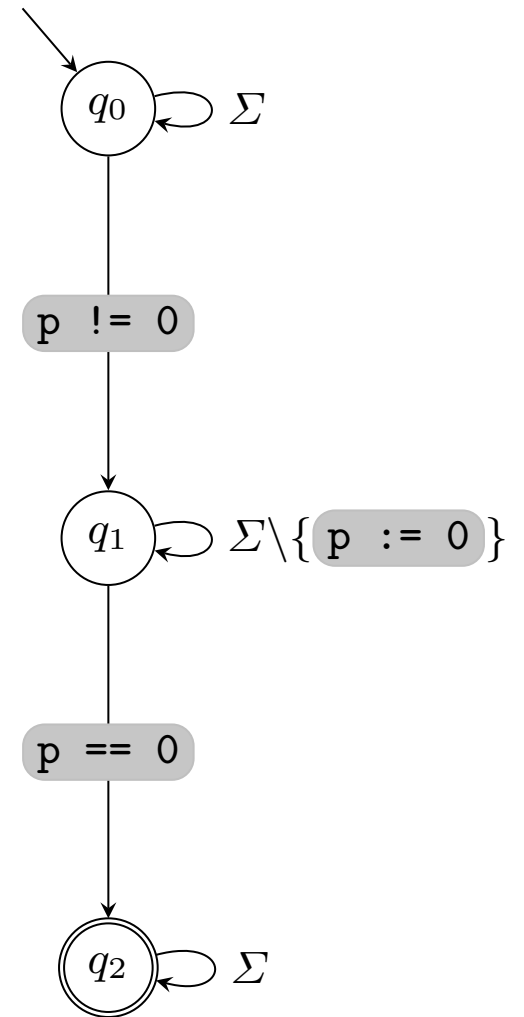
automaton constructed from unsatisfiability proof



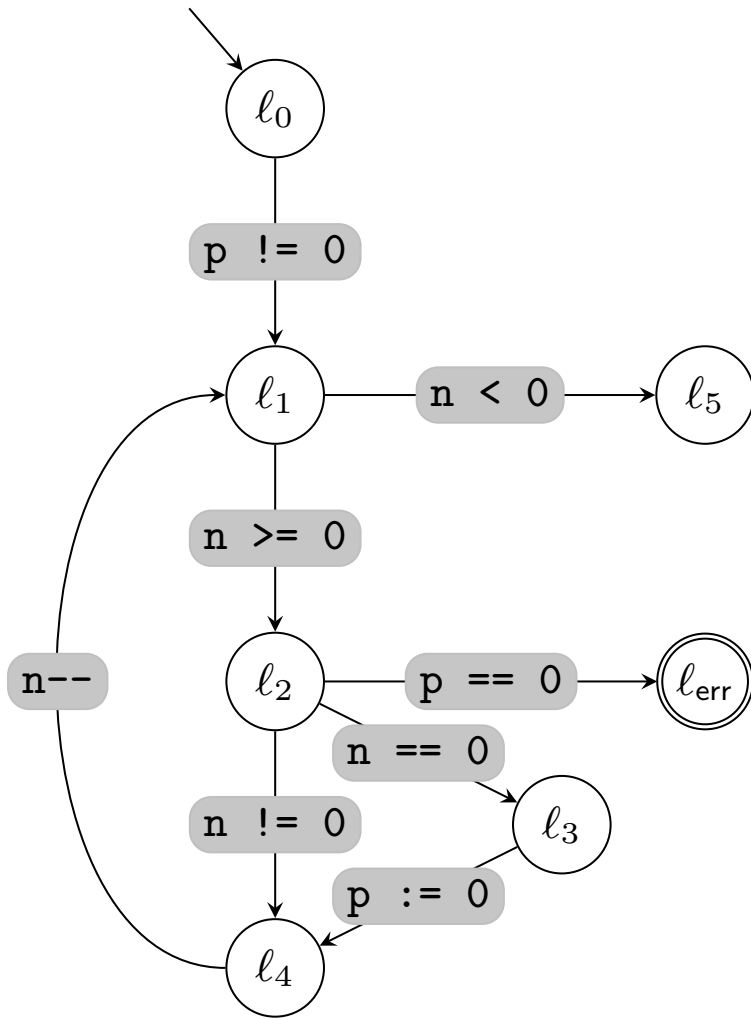
“data automaton” (ignores control of program)



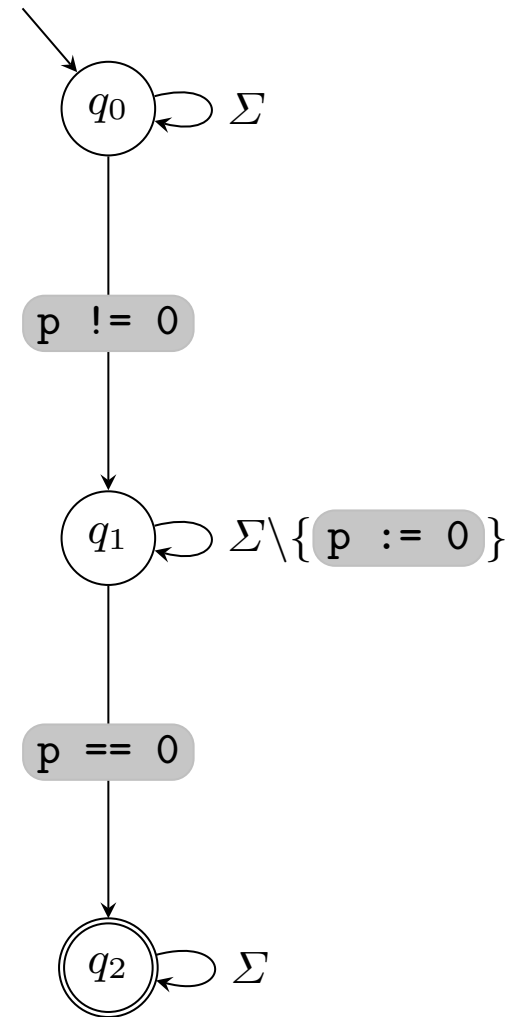
subset?



program automaton subset of data automaton ?

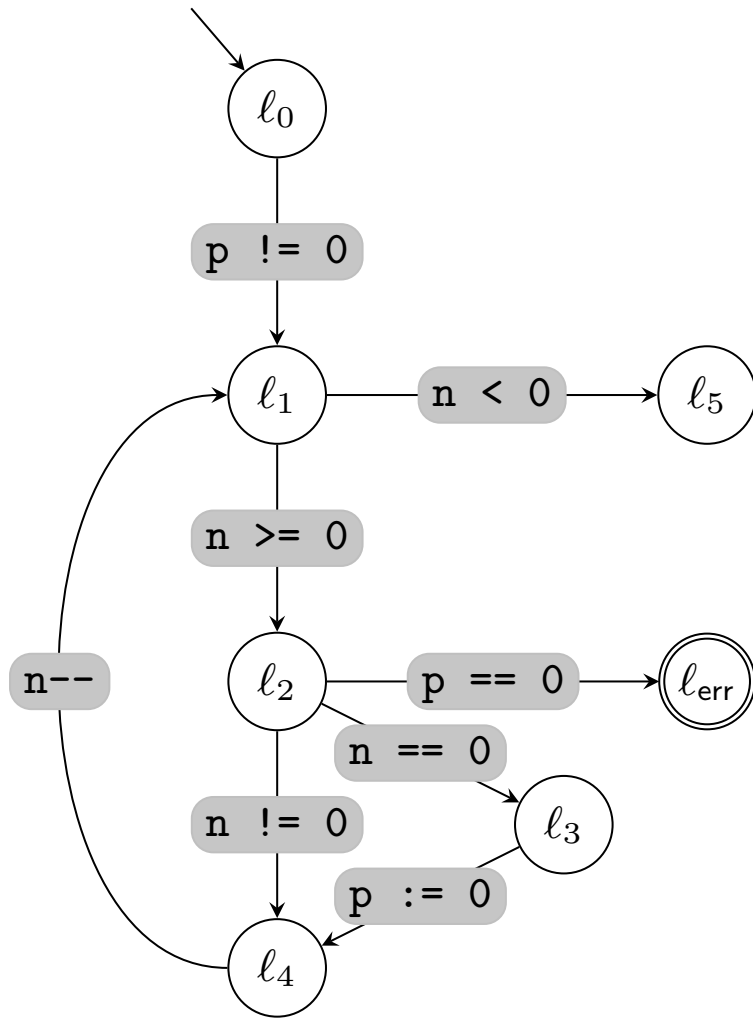


subset?

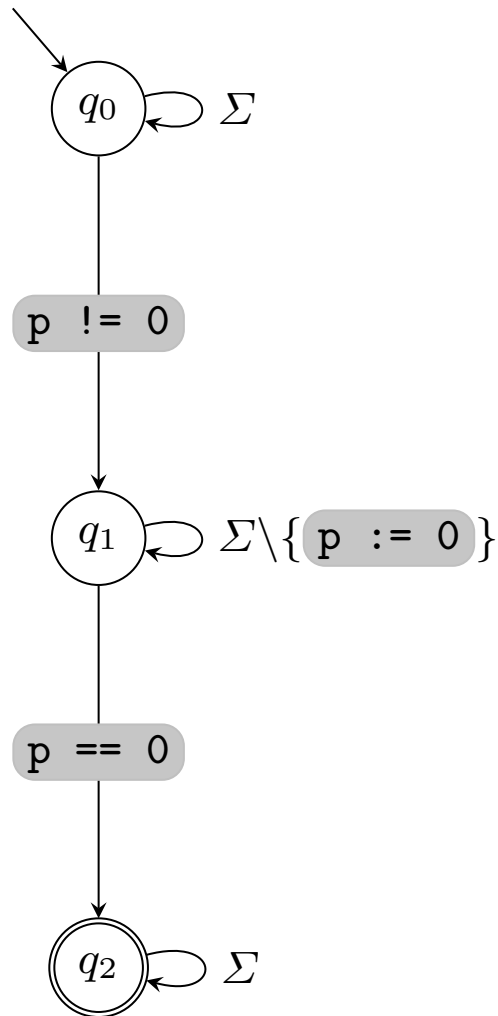


program automaton not subset of data automaton

new error trace

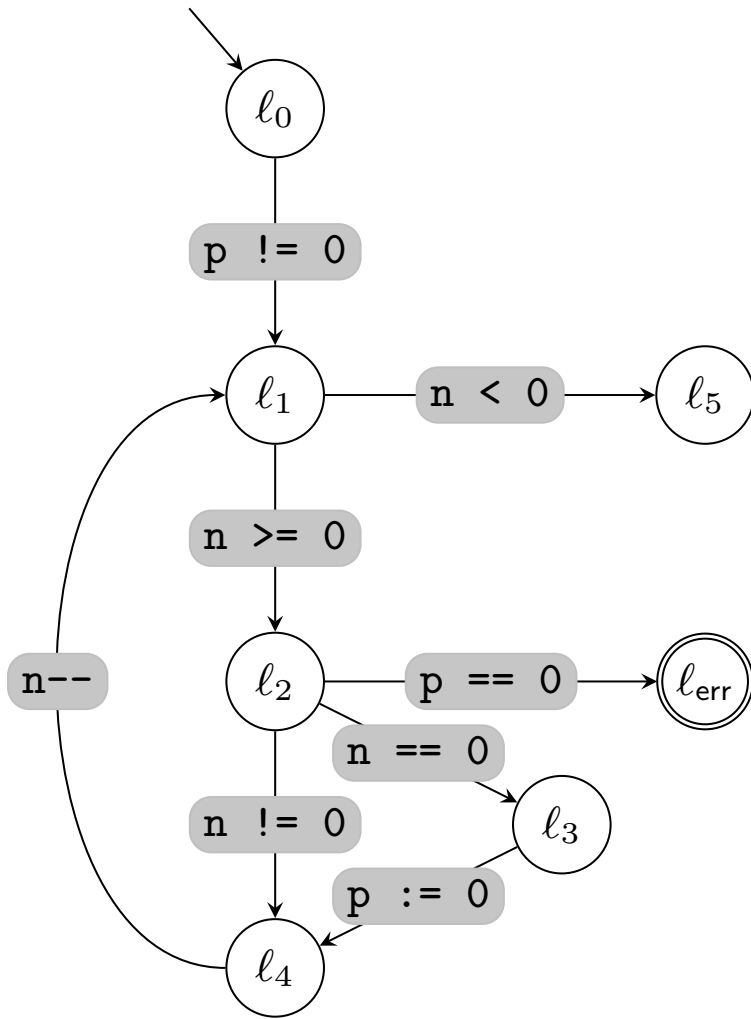


(p != 0)
(n >= 0)
(n == 0)
(p := 0)
(n--)
(n >= 0)
(p == 0)



(p != 0)
 (n >= 0)
 (n == 0)
 (p := 0)
 (n--)
 (n >= 0)
 (p == 0)

data automaton does not accept new error trace



(p != 0)

(n >= 0)

(n == 0)

(p := 0)

(n--)

(n >= 0)

(p == 0)

(n == 0)

(n--)

(n >= 0)

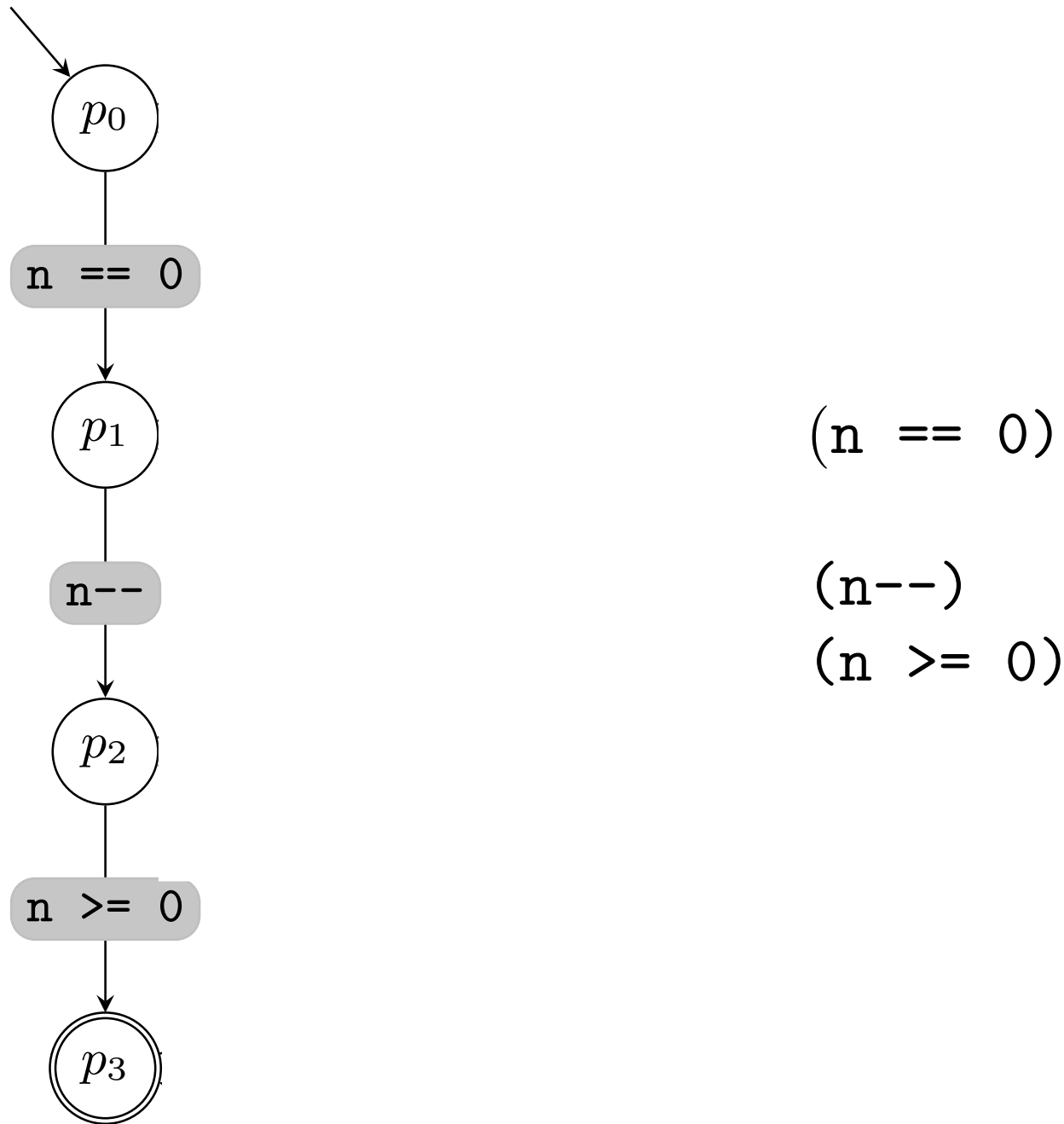
trace **infeasible**, take **unsatisfiable** core

`(n == 0)`

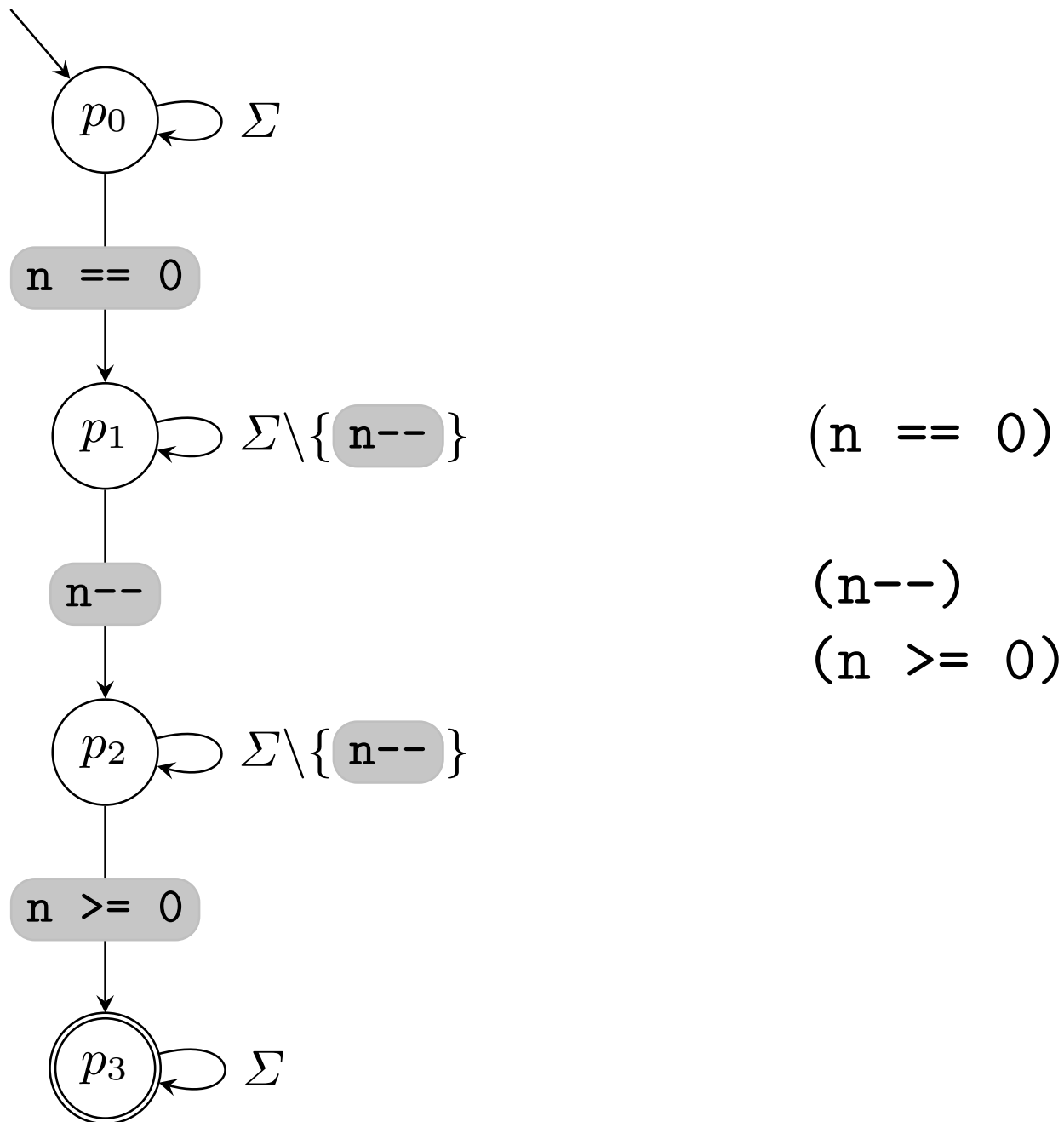
`(n--)`

`(n >= 0)`

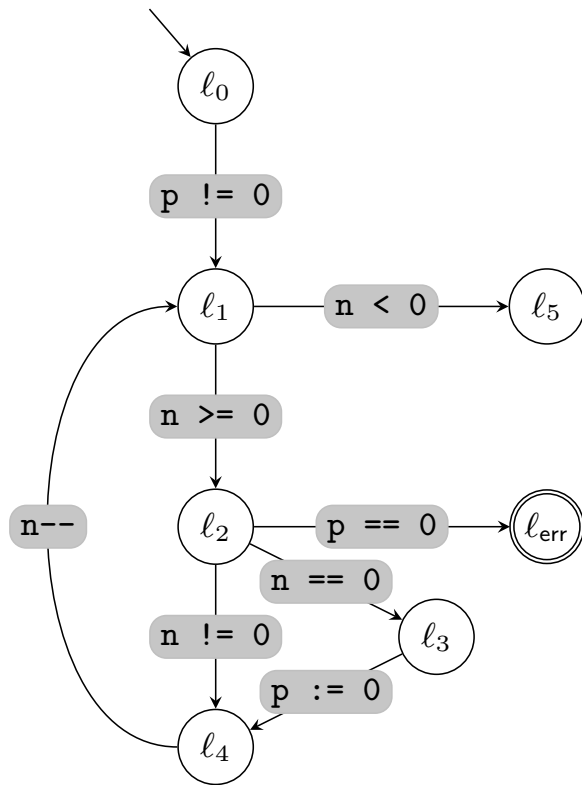
unsatisfiable core



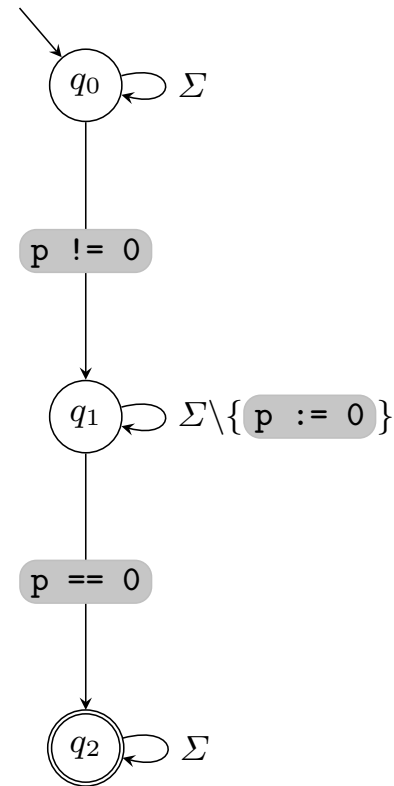
construct second data automaton from unsatisfiable core (step I)



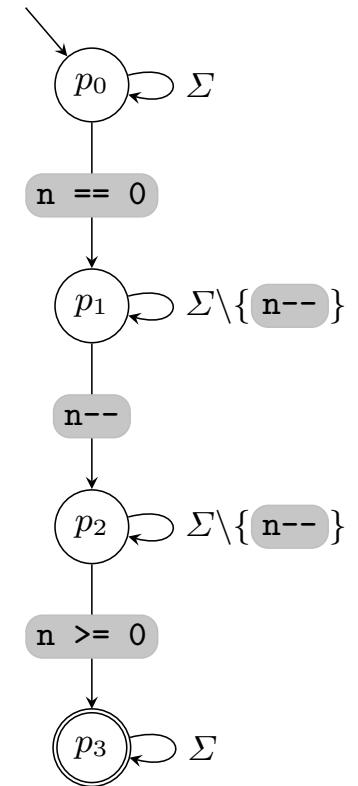
construct second data automaton from unsatisfiable core (step 2)



subset



U



program automaton is subset of union of data automata

program \mathcal{P}

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

1. $w \in \mathcal{A}_{n+1}$
2. $\mathcal{A}_{n+1} \subseteq \{ \text{infeasible traces} \}$

$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n ?$

w infeasible?

yes

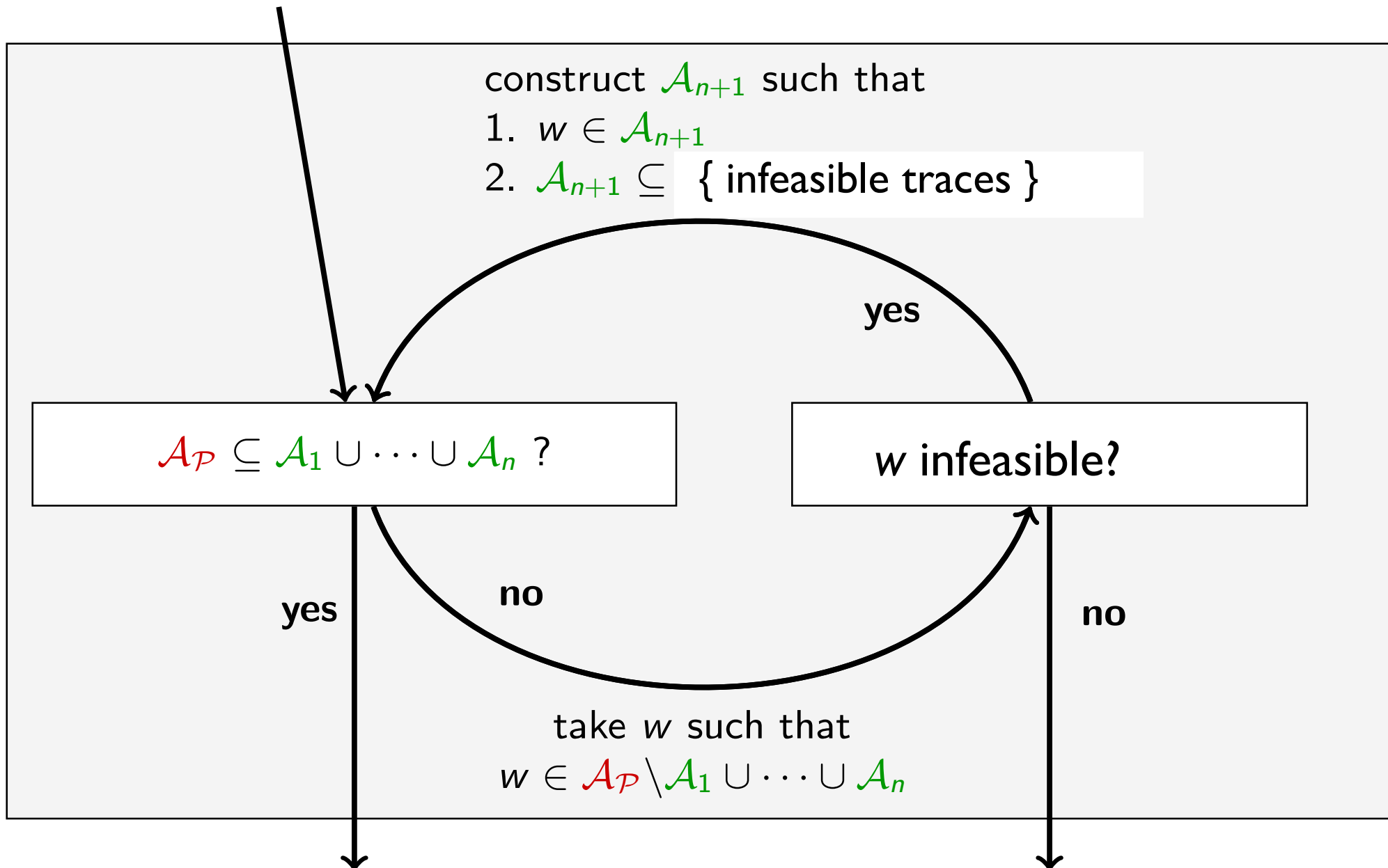
no

no

take w such that
 $w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n$

\mathcal{P} is correct

\mathcal{P} is incorrect



repeat

- take **error trace** in **program**
- check **infeasibility** of **error trace**
- construct **data automaton** from **unsatisfiability** proof

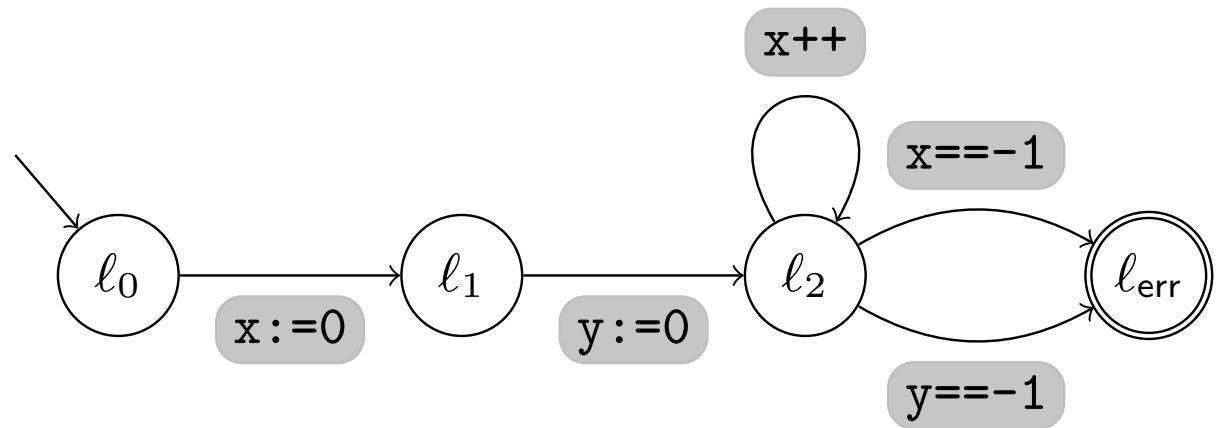
until **program** is subset of union of **data automata**

if automata are constructed from **unsatisfiable core**

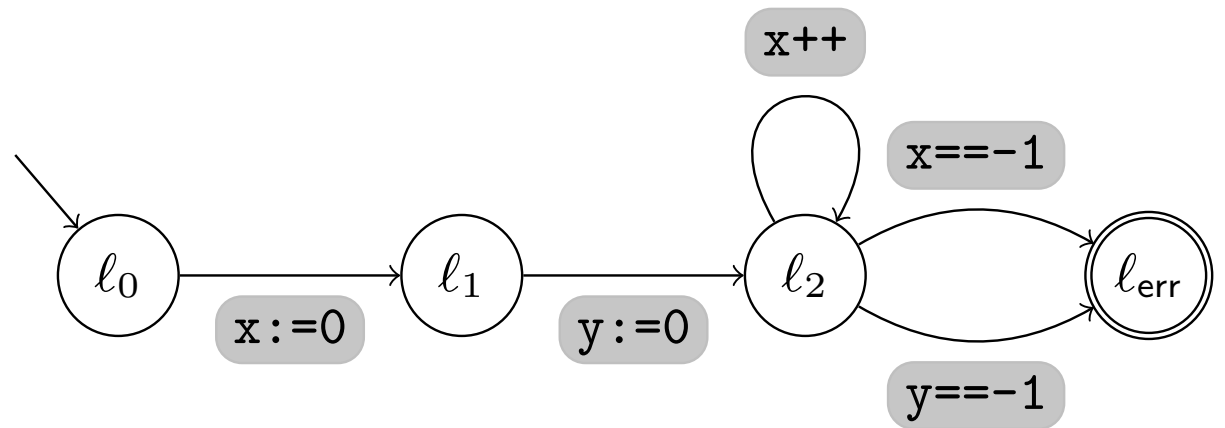
is the verification algorithm complete?

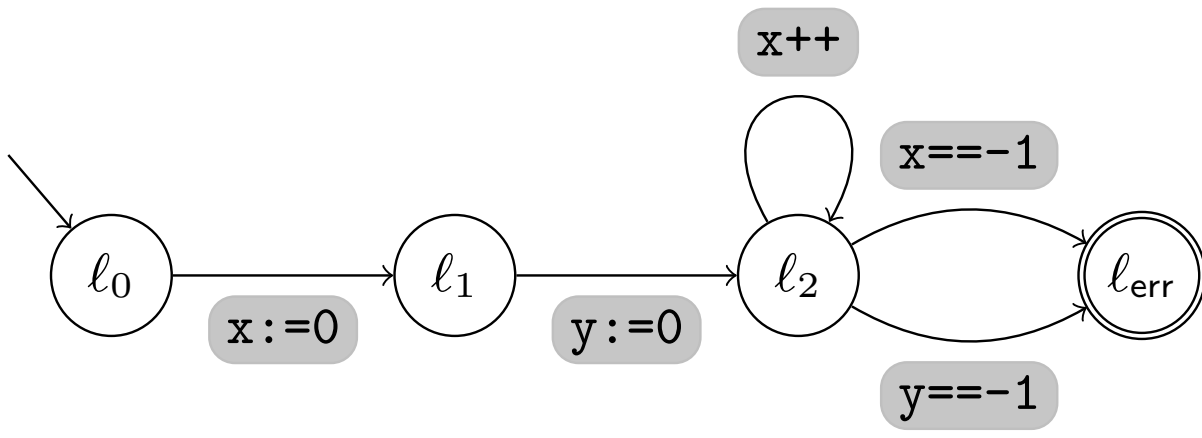
(can we prove every correct program correct?)

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




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





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

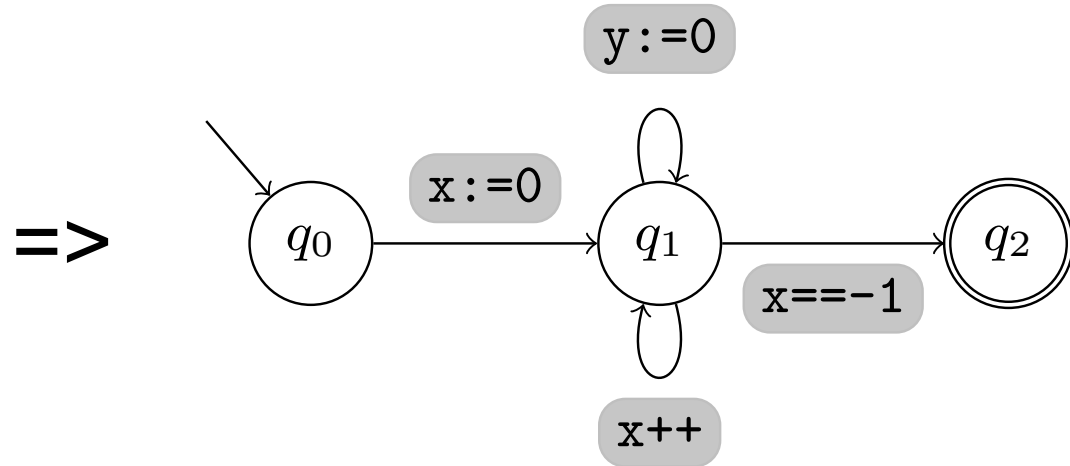
trace infeasible =

trace satisfies pre/postcondition pair (*true*, *false*)

infeasibility proof with Hoare triples

$$\begin{array}{l} \{ \textit{true} \} \quad \mathbf{x} := 0 \quad \{ x \geq 0 \} \\ \{ x \geq 0 \} \quad \mathbf{y} := 0 \quad \{ x \geq 0 \} \\ \{ x \geq 0 \} \quad \mathbf{x}++ \quad \{ x \geq 0 \} \\ \{ x \geq 0 \} \quad \mathbf{x} == -1 \quad \{ \textit{false} \} \end{array}$$

automaton constructed from Hoare triples

$$\begin{array}{l} \{ true \} \quad x:=0 \quad \{ x \geq 0 \} \\ \{ x \geq 0 \} \quad y:=0 \quad \{ x \geq 0 \} \\ \{ x \geq 0 \} \quad x++ \quad \{ x \geq 0 \} \\ \{ x \geq 0 \} \quad x== -1 \quad \{ false \} \end{array}$$


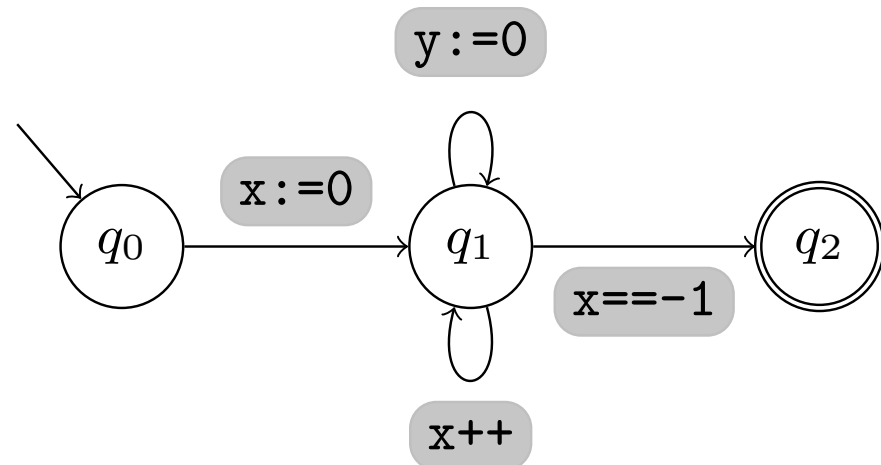
all traces accepted by automaton are **infeasible**

infeasible = satisfy pre/postcondition pair (*true*, *false*)

construction “Hoare proof \mapsto automaton”

Hoare triple \mapsto transition

$\{ true \}$	$x := 0$	$\{ x \geq 0 \}$
$\{ x \geq 0 \}$	$y := 0$	$\{ x \geq 0 \}$
$\{ x \geq 0 \}$	$x++$	$\{ x \geq 0 \}$
$\{ x \geq 0 \}$	$x == -1$	$\{ false \}$



construction “Hoare proof \longmapsto automaton”

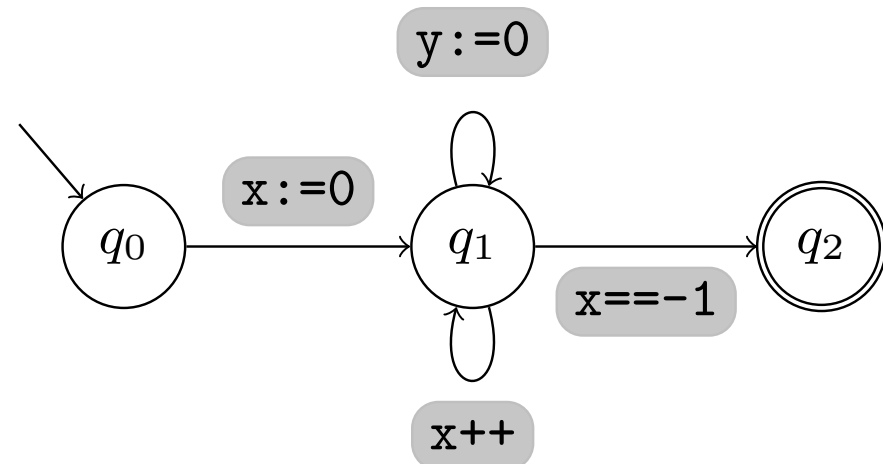
assertion \longmapsto state

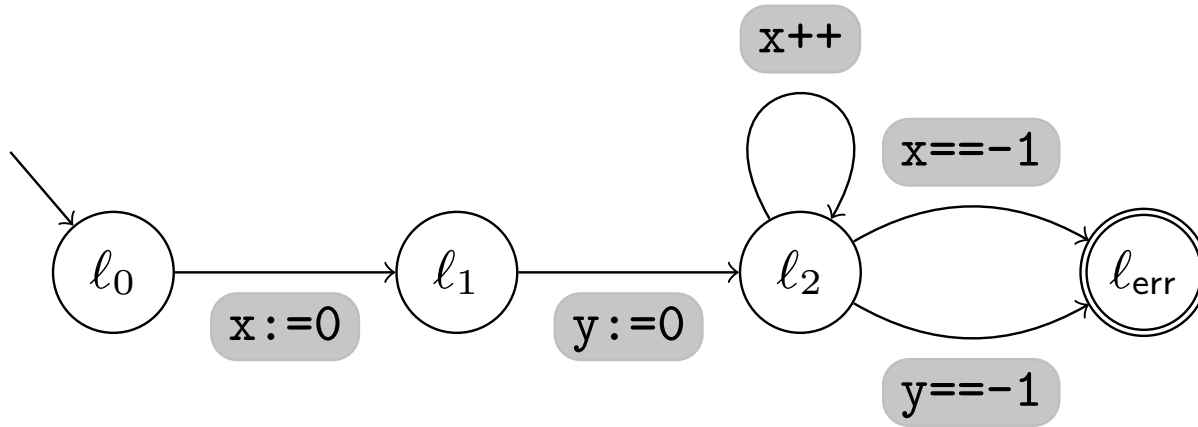
Hoare triple \longmapsto transition

precondition \longmapsto initial state

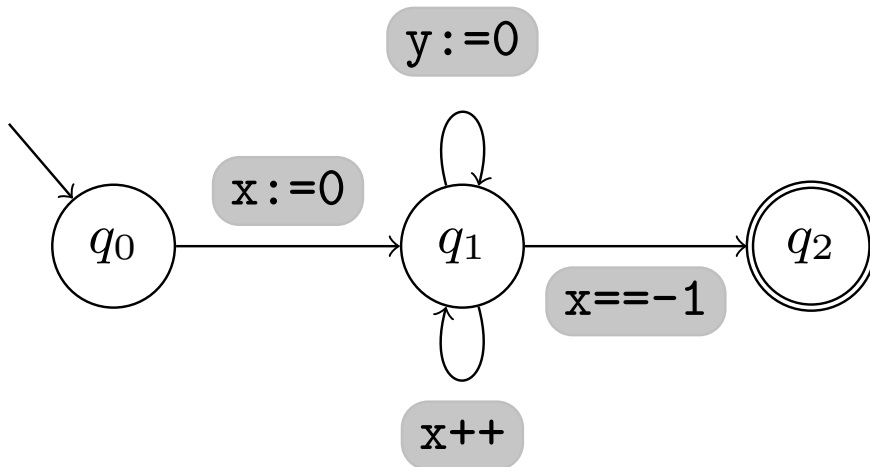
postcondition \longmapsto final state

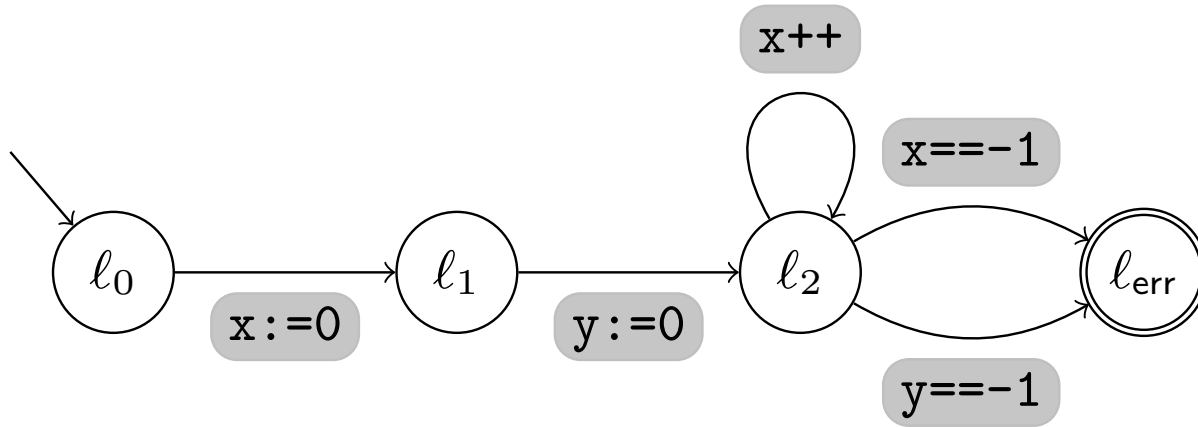
$\{ true \}$	$x:=0$	$\{ x \geq 0 \}$
$\{ x \geq 0 \}$	$y:=0$	$\{ x \geq 0 \}$
$\{ x \geq 0 \}$	$x++$	$\{ x \geq 0 \}$
$\{ x \geq 0 \}$	$x== -1$	$\{ false \}$





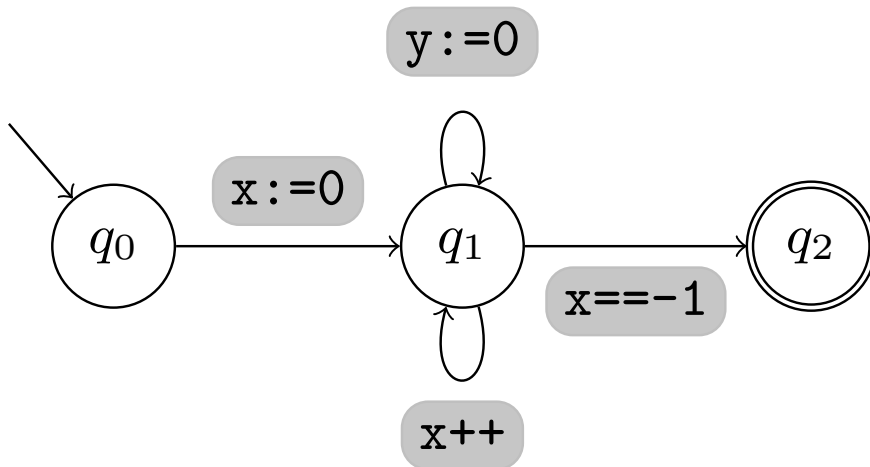
program automaton subset data automaton?





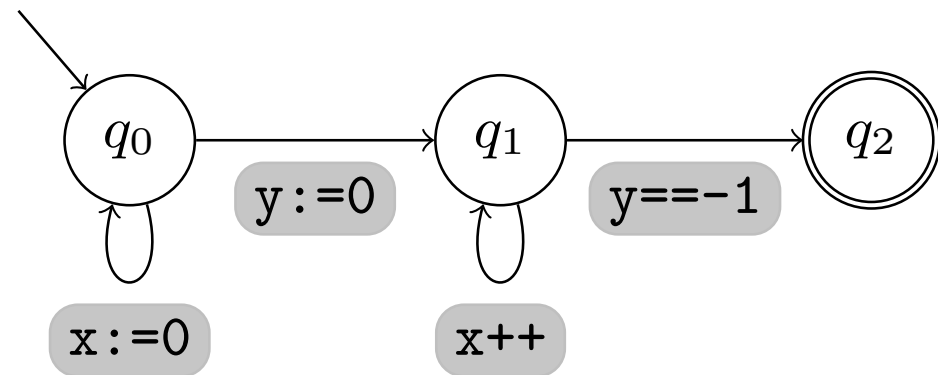
program automaton not subset of data automaton:

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



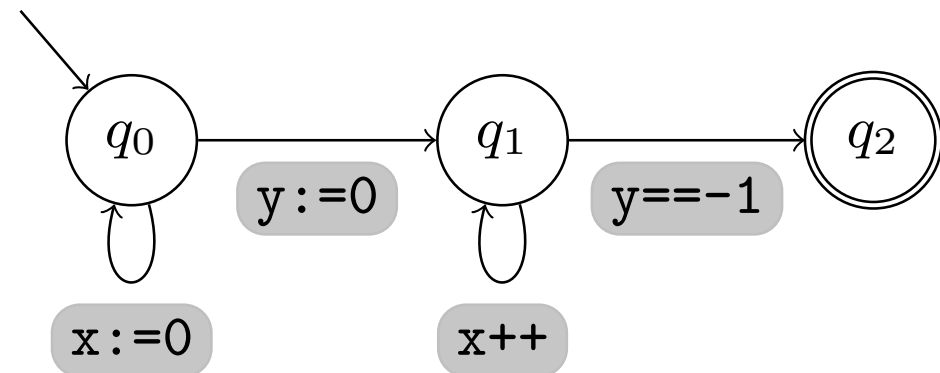
automaton from **Hoare triples** which prove infeasibility

$\{ true \} \quad x:=0 \quad \{ true \}$
 $\{ true \} \quad y:=0 \quad \{ y = 0 \}$
 $\{ y = 0 \} \quad x++ \quad \{ y = 0 \}$
 $\{ y = 0 \} \quad y== -1 \quad \{ false \}$



automaton from **Hoare triples** which prove infeasibility

$\{ true \} \quad x:=0 \quad \{ true \}$
 $\{ true \} \quad y:=0 \quad \{ y = 0 \}$
 $\{ y = 0 \} \quad x++ \quad \{ y = 0 \}$
 $\{ y = 0 \} \quad y== -1 \quad \{ false \}$



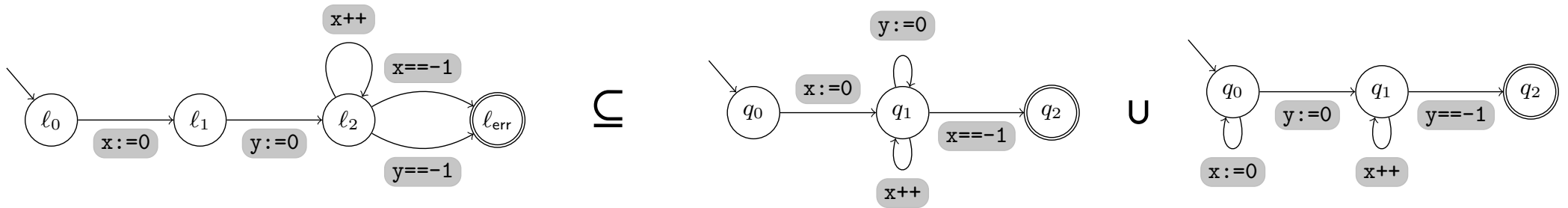
we can construct the same automaton from unsatisfiable core
(since variable x does not appear in unsatisfiable core)

construction of automaton from **unsatisfiable core**
is a special case of
construction of automaton from **Hoare proof**

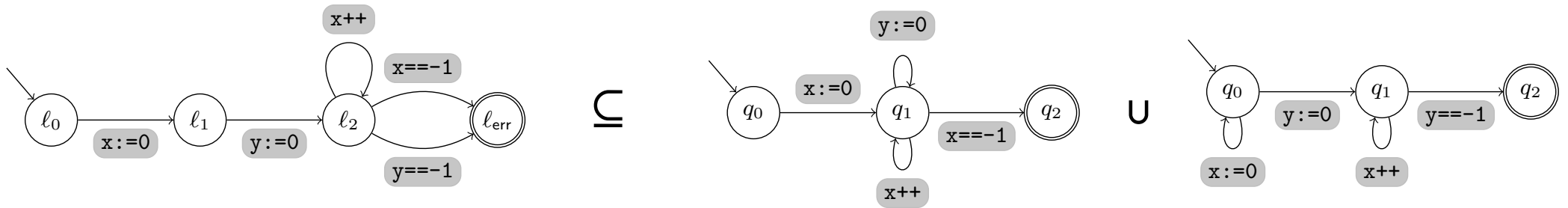
exists proof for infeasibility of trace

\Rightarrow

exists Hoare proof whose assertions are invariant under any statement that does not update a variable in unsatisfiable core



program automaton **subset** of union of data automata ?



program automaton **is** subset of union of data automata!

program \mathcal{P}

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

1. $w \in \mathcal{A}_{n+1}$

2. $\mathcal{A}_{n+1} \subseteq \{ \text{infeasible traces} \}$

$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n ?$

w infeasible?

yes

no

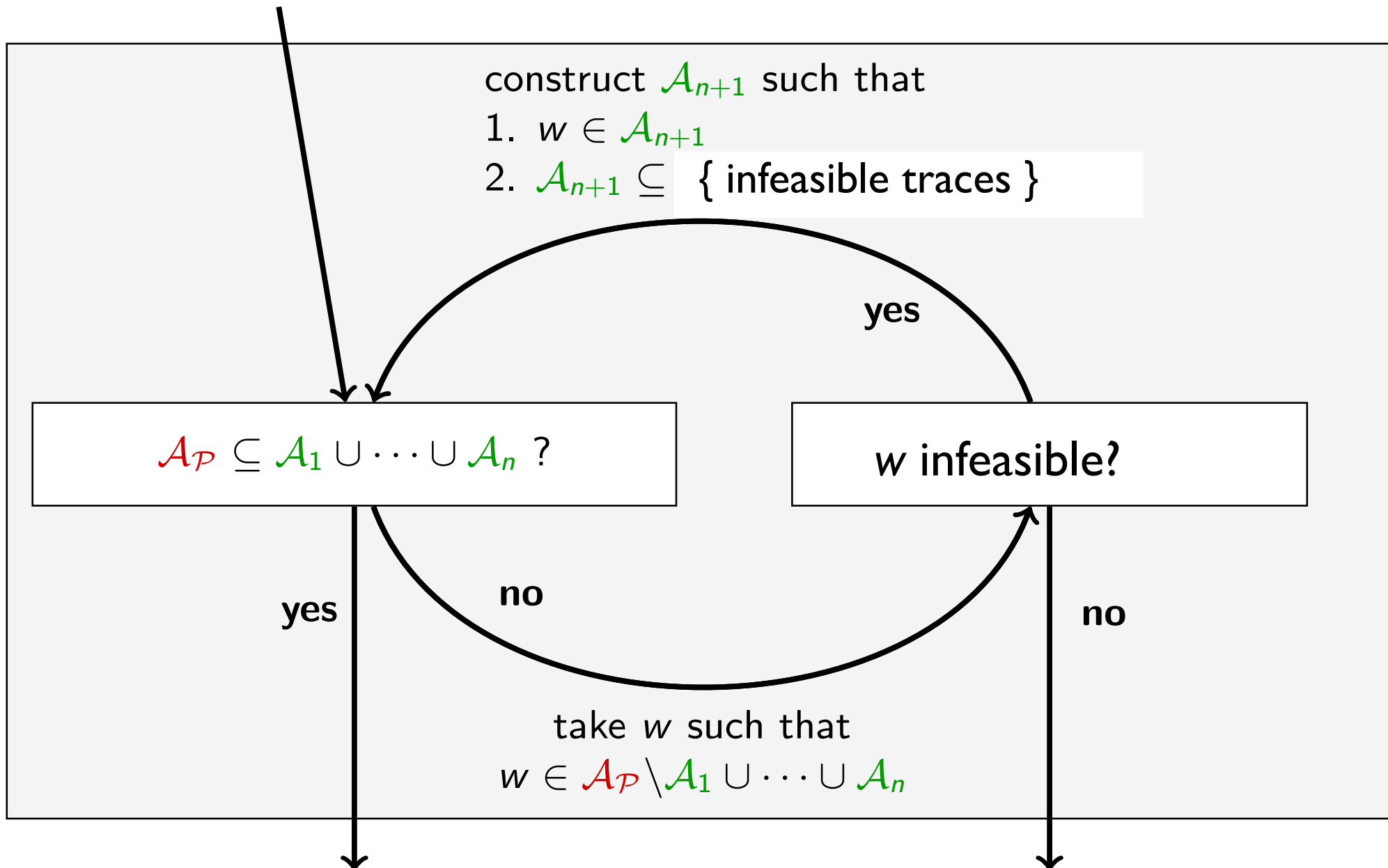
no

take w such that

$w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n$

\mathcal{P} is correct

\mathcal{P} is incorrect



repeat

- take **error trace** in **program**
- check infeasibility of **error trace**
- construct **data automaton** from **Hoare** triples

until **program** is subset of union of **data automata**

repeat

- take **error trace** in **program**

- check **infeasibility** of **error trace**

- construct **data automaton** from **unsatisfiability** proof

until **program** is subset of union of **data automata**

repeat

- take **error trace** in **program**
- check infeasibility of **error trace**
- construct **data automaton** from **Hoare** triples

until **program** is subset of union of **data automata**

completeness of verification algorithm

for every correct program
there exists data automata
such that

program automaton \subseteq union of data automata

conclusion and future work

automaton constructed from **proof**

proof generated by *SMT solver*

(Satisfiability checker Modulo Theory)

a trace is a word

a trace is a program

we can use automata to
express new sets of traces

program is just one particular automaton
program expresses one particular set of traces

“cover program by union of simple automata”

Automizer

Uni-Freiburg : SWT - Ultimate - rekonnq

Uni-Freiburg : SWT - U... <https://monteverdi.informatik.uni-freiburg.de/tomcat//Website/?task=VerifyC#>

ULTIMATE WEB-INTERFACE

Task: Verify C
Sample: McCarthy91.c
Tool: Trace Abstraction
Trace abstraction toolchain

SETTINGS

EXECUTE

[Show editor fullscreen](#)
Choose File No file selected

```
12 /*@ requires \true;
i 13  @ ensures x > 101 || \result == 91;
14  @*/
i 15  int f91(int x);
16
i 17  int f91(int x) {
18    if (x > 100)
19      return x -10;
20  else {
i 21    return f91(f91(x+11));
22  }
23 }
24
25
26
```

	Line	Ultimate Result
	21	procedure precondition always holds
	21	procedure precondition always holds
	13	procedure postcondition always holds

C and Boogie, safety and termination

sequential programs nondeterministic finite automata
termination Buchi automata
recursion nested word automata
concurrency alternating finite automata
unbounded parallelism predicate automata
proofs that count Petri net \subseteq counting automaton

data base of automata

automata constructed from proofs

program \mathcal{P}

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

1. $w \in \mathcal{A}_{n+1}$
2. $\mathcal{A}_{n+1} \subseteq \{ \text{infeasible traces} \}$

$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n ?$

w infeasible?

yes

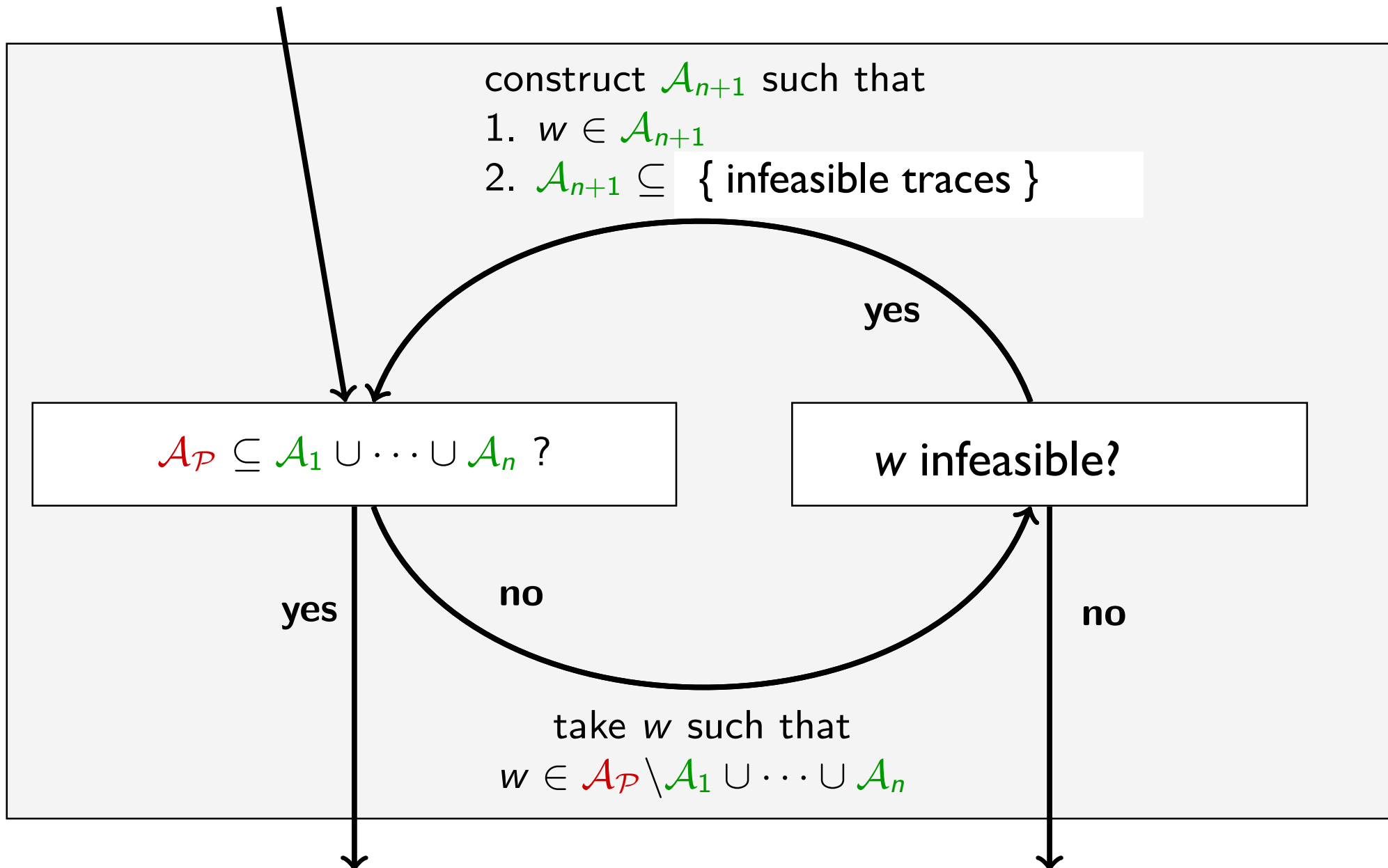
no

no

take w such that
 $w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \dots \cup \mathcal{A}_n$

\mathcal{P} is correct

\mathcal{P} is incorrect



construct automaton from proof of **incorrectness**

error diagnosis

statement *irrelevant* if on loop in automaton

classify error paths

error paths equivalent if same automaton