Search Modulo Theory

Andreas Podelski
University of Freiburg

ℓ_0: `assume p != 0;`

ℓ_1: `while(n >= 0)`

`{`

ℓ_2:

`if(n == 0)`

`{`

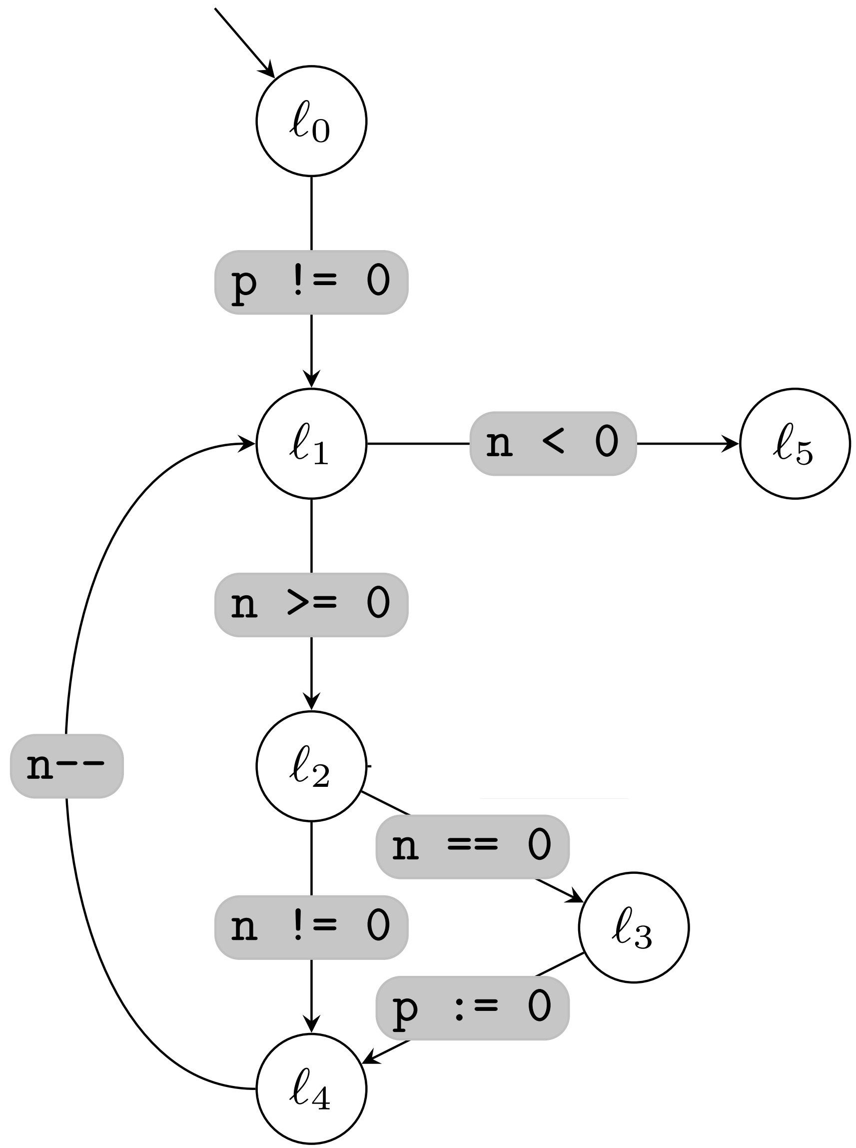
ℓ_3: `p := 0;`

`}`

ℓ_4: `n--;`

`}`

ℓ_5:



ℓ_0: `assume p != 0;`

ℓ_1: `while(n >= 0)`

```
{
```

ℓ_2: `    assert p != 0;`

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

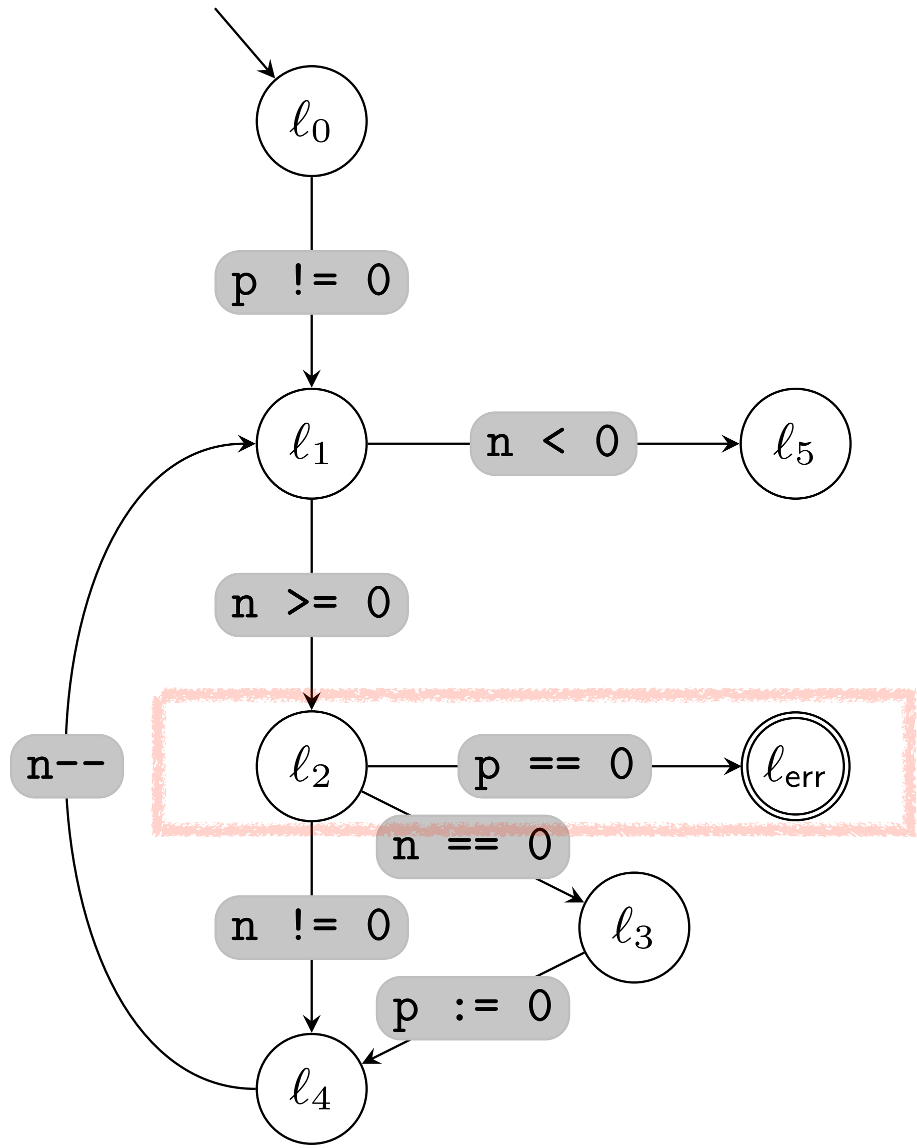
ℓ_3: `            p := 0;`

```
        }
```

ℓ_4: `        n--;`

```
    }
```

ℓ_5:

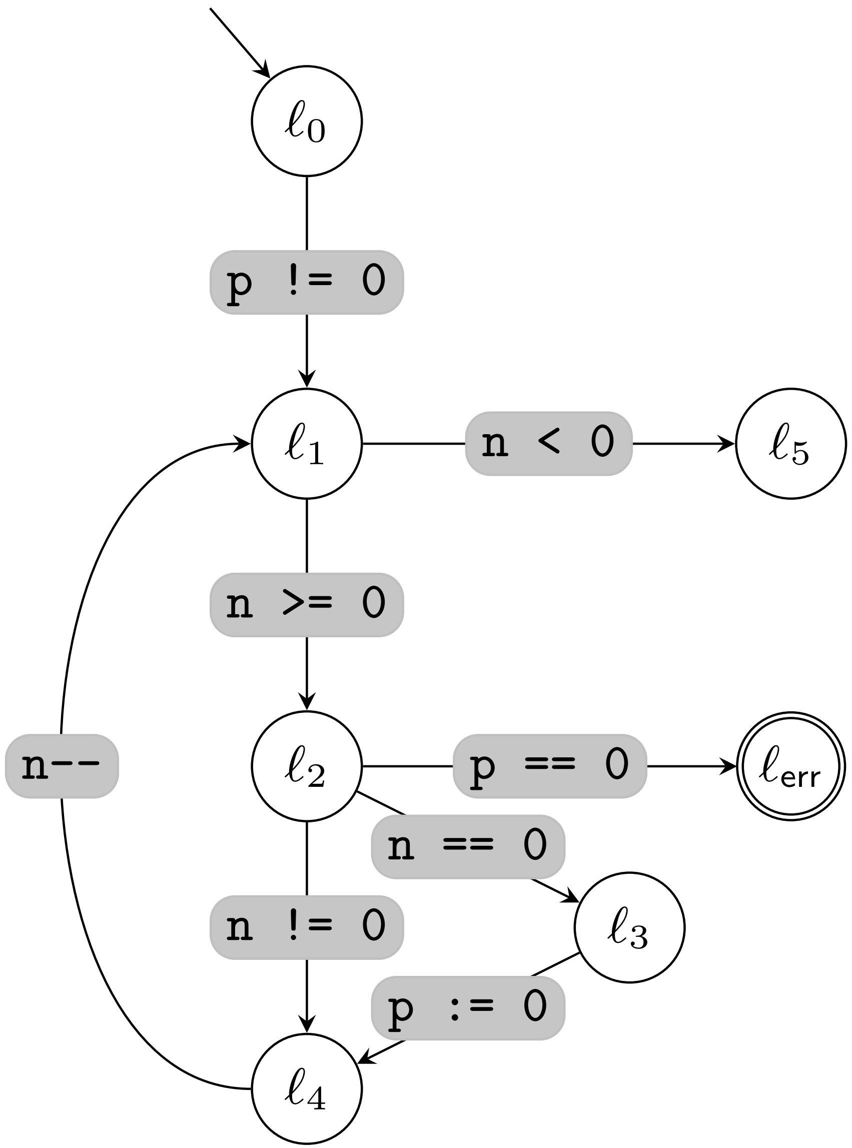


```
ℓ0: assume p != 0;

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

unsatisfiable *Modulo Theory*

```
x == 1 ; x == -1 ;
```

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

```
x := 1 ; x == -1 ;
```

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

Automated Program Verification

Andreas Podelski
University of Freiburg

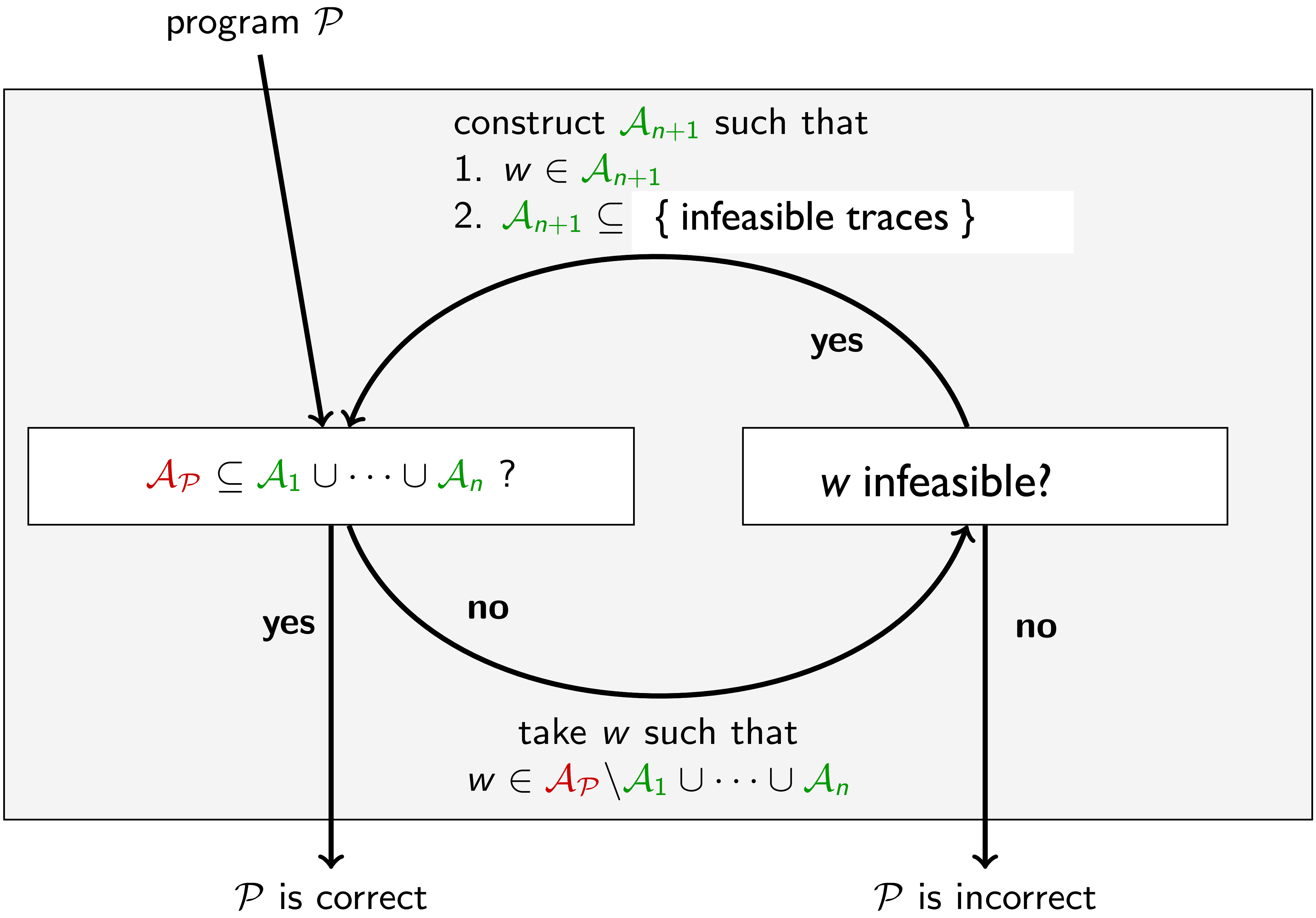
joint work with
Matthias Heizmann and Jochen Hoenicke
University of Freiburg
Azadeh Farzan and Zachary Kincaid
University of Toronto

au*t*omaton

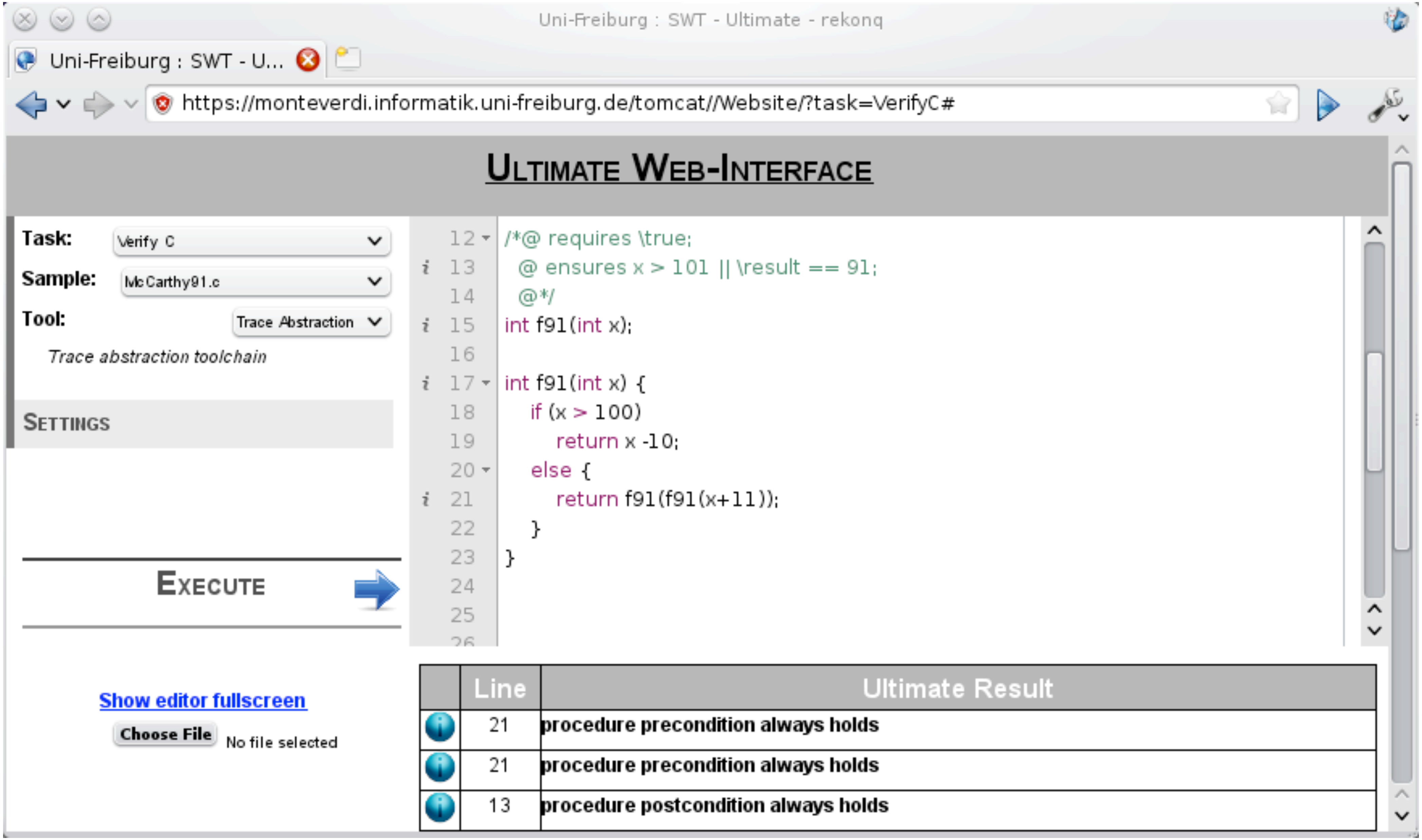
|ôˈtämətən|

auto*ma*tion

|ˌôtəˈmāSHən|


program $\mathcal{P}$
construct $\mathcal{A}_{n+1}$ such that
1. $w \in \mathcal{A}_{n+1}$
2. $\mathcal{A}_{n+1} \subseteq$ { infeasible traces }
yes
$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$?
w infeasible?
yes
no
no
take w such that
$w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$
$\mathcal{P}$ is correct
$\mathcal{P}$ is incorrect

Ultimate Automizer



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

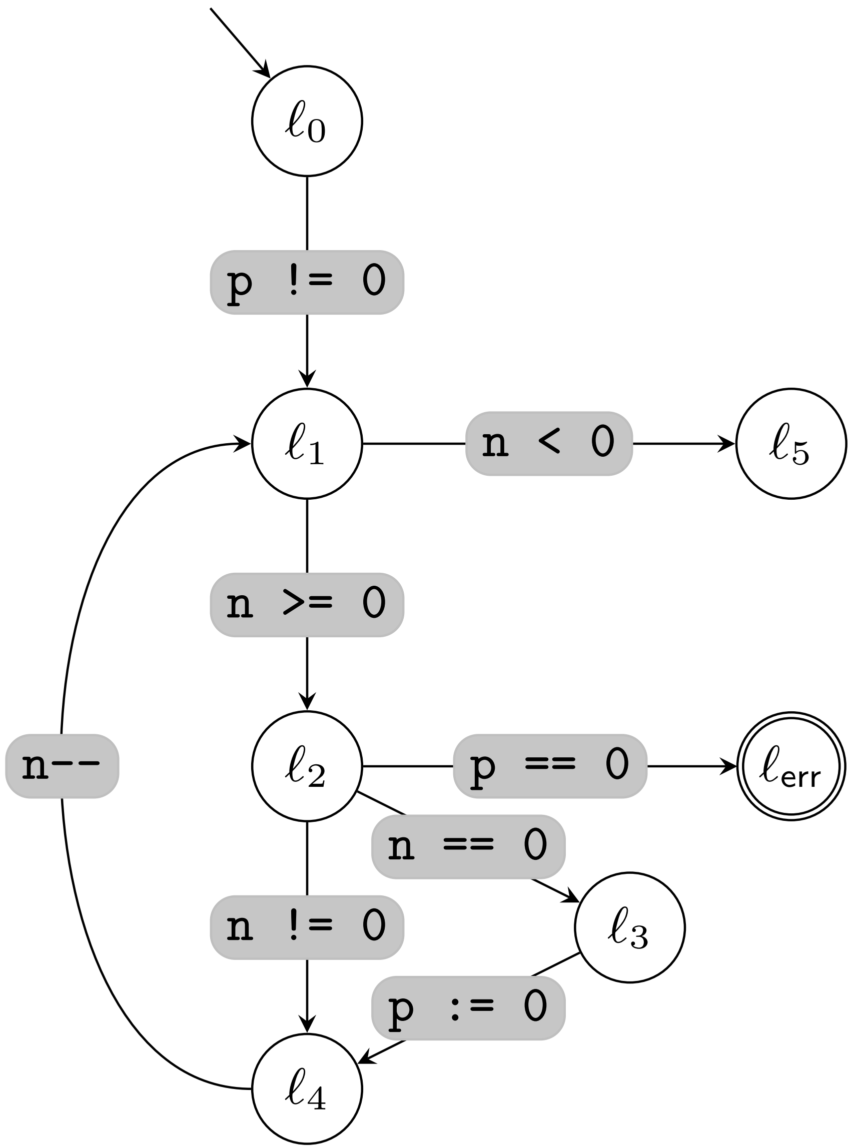
search *Modulo Theory*

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

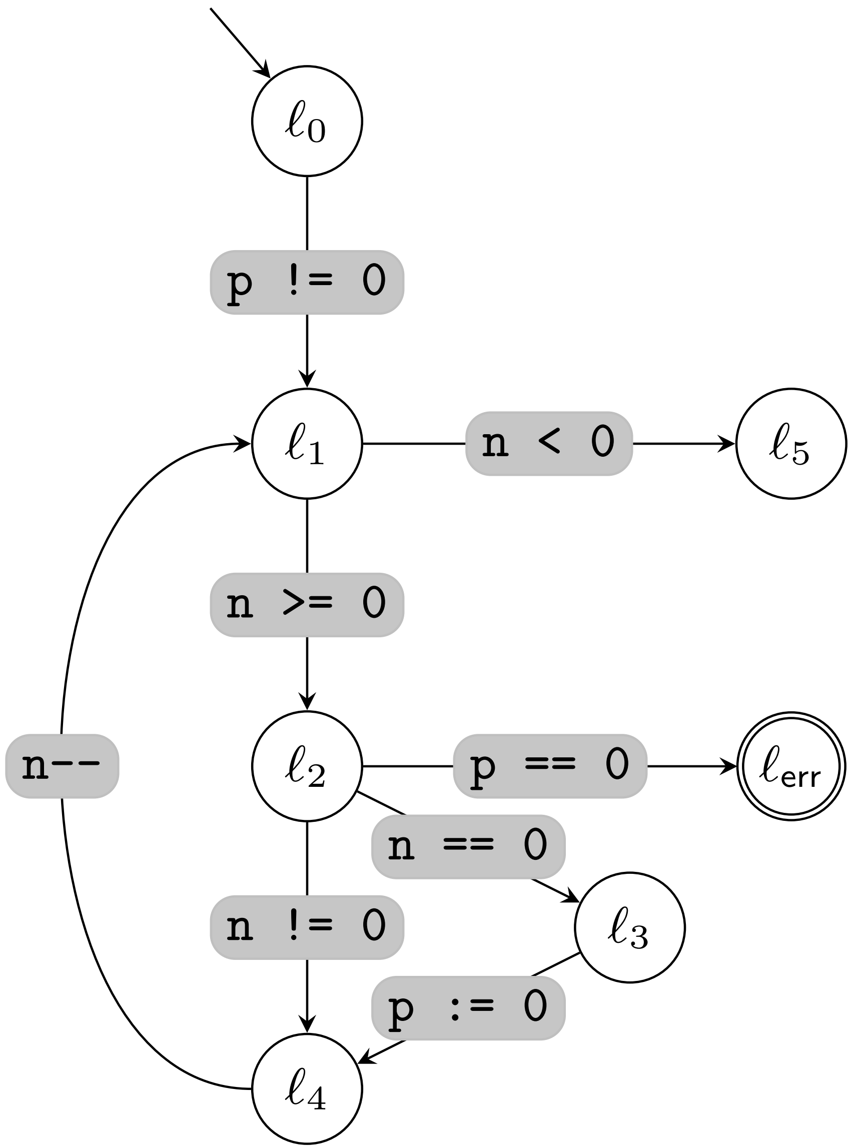
lemmas are automata (sets of paths)
automata constructed from proofs of *SMT* solver

SMT: Satisfiability Modulo Theory

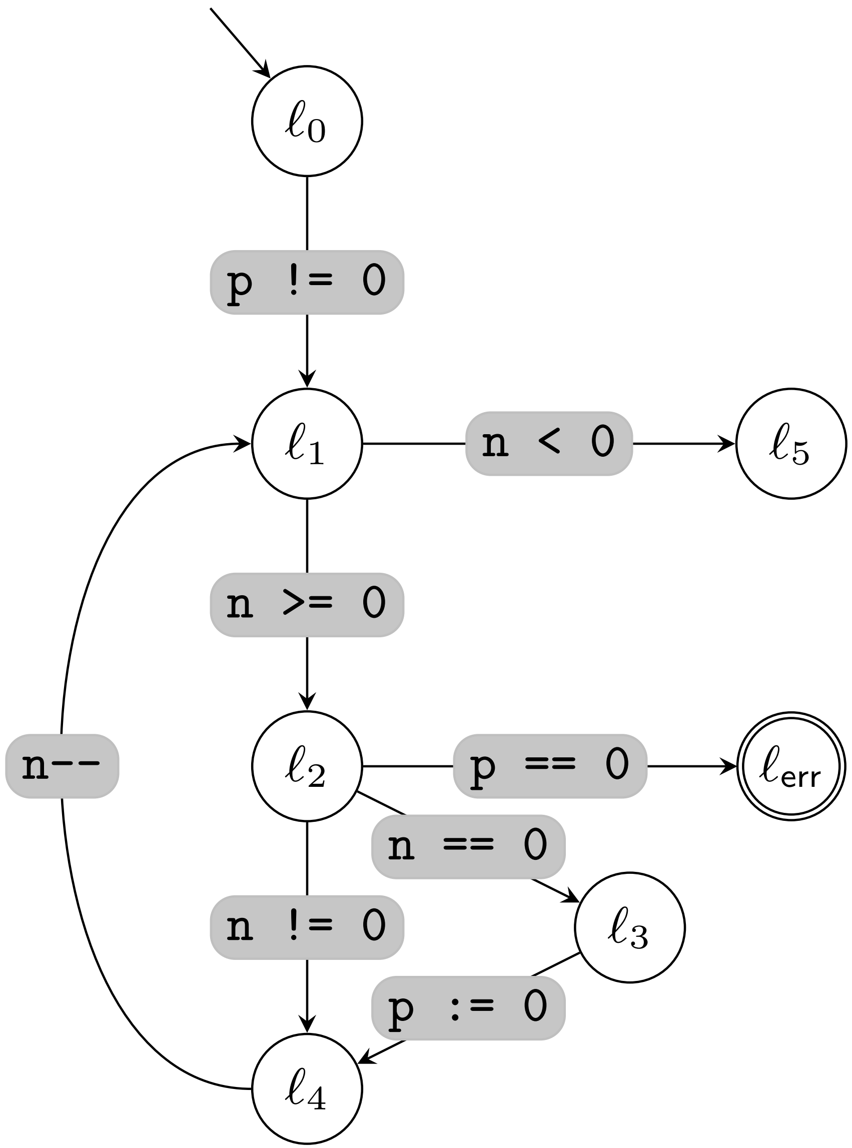
error trace: sequence of statements along path to error location



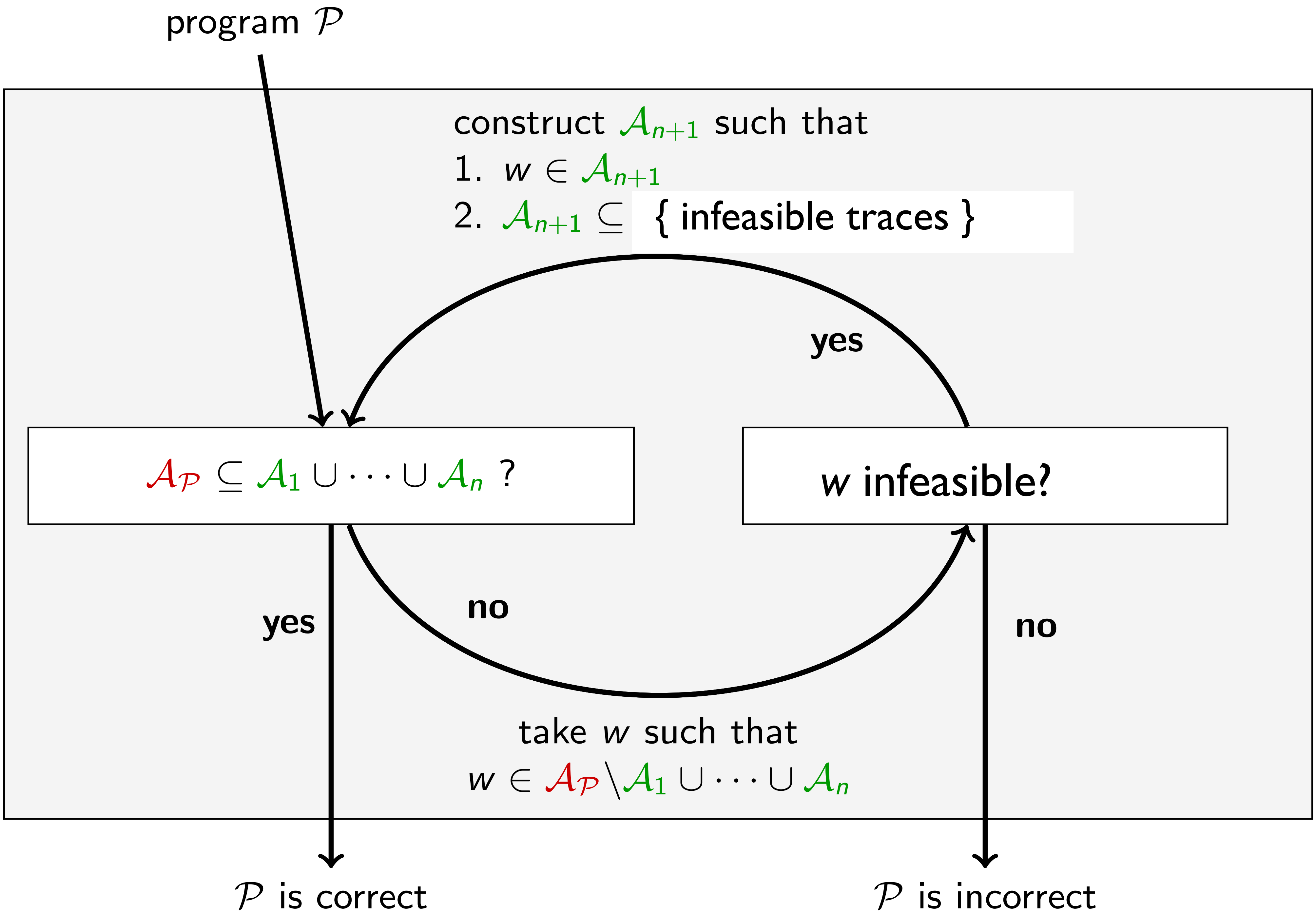
error trace: sequence of statements along path to error location

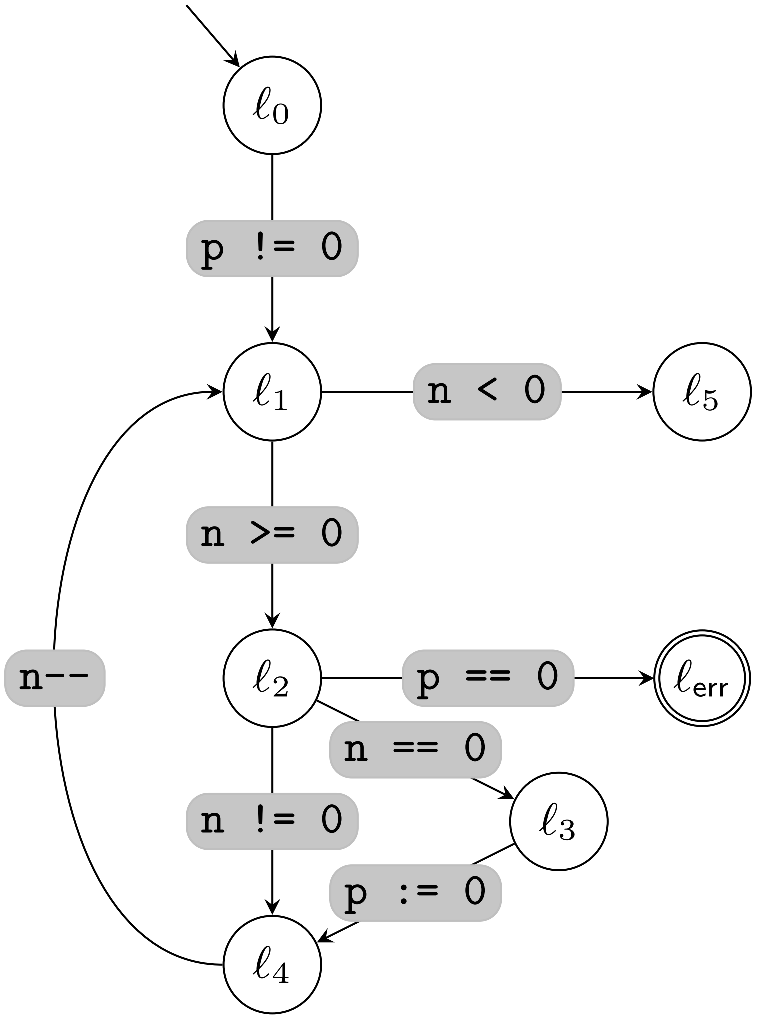


error trace: word accepted by **program automaton** program correct = no execution reaches error location



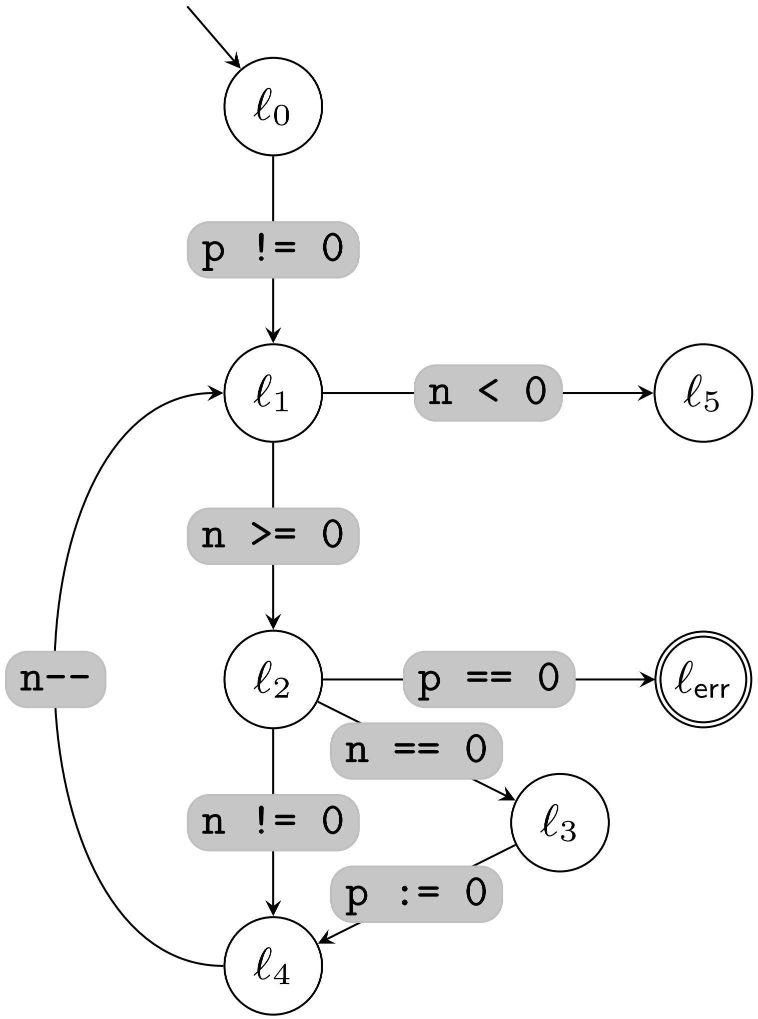
program correct = no feasible error trace


program $\mathcal{P}$
construct $\mathcal{A}_{n+1}$ such that
1. $w \in \mathcal{A}_{n+1}$
2. $\mathcal{A}_{n+1} \subseteq$ { infeasible traces }
yes
$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$?
w infeasible?
yes
no
no
take w such that
$w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$
$\mathcal{P}$ is correct
$\mathcal{P}$ is incorrect

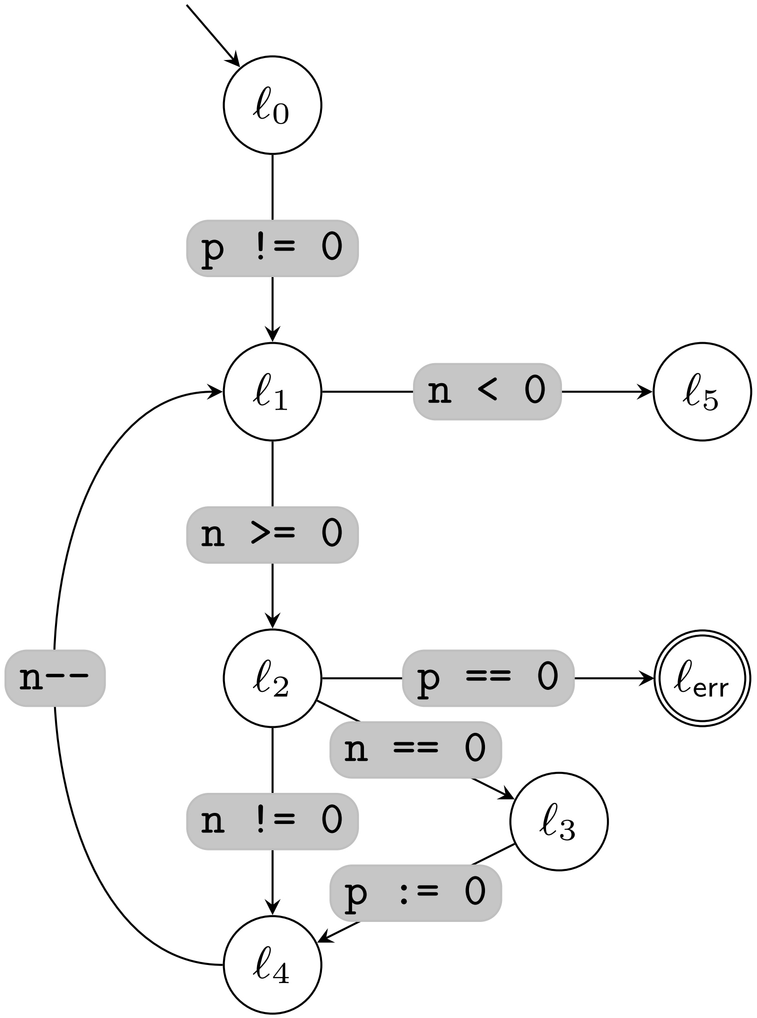


Does there exist
an execution
that leads to error state?

Does there exist
a feasible error trace?

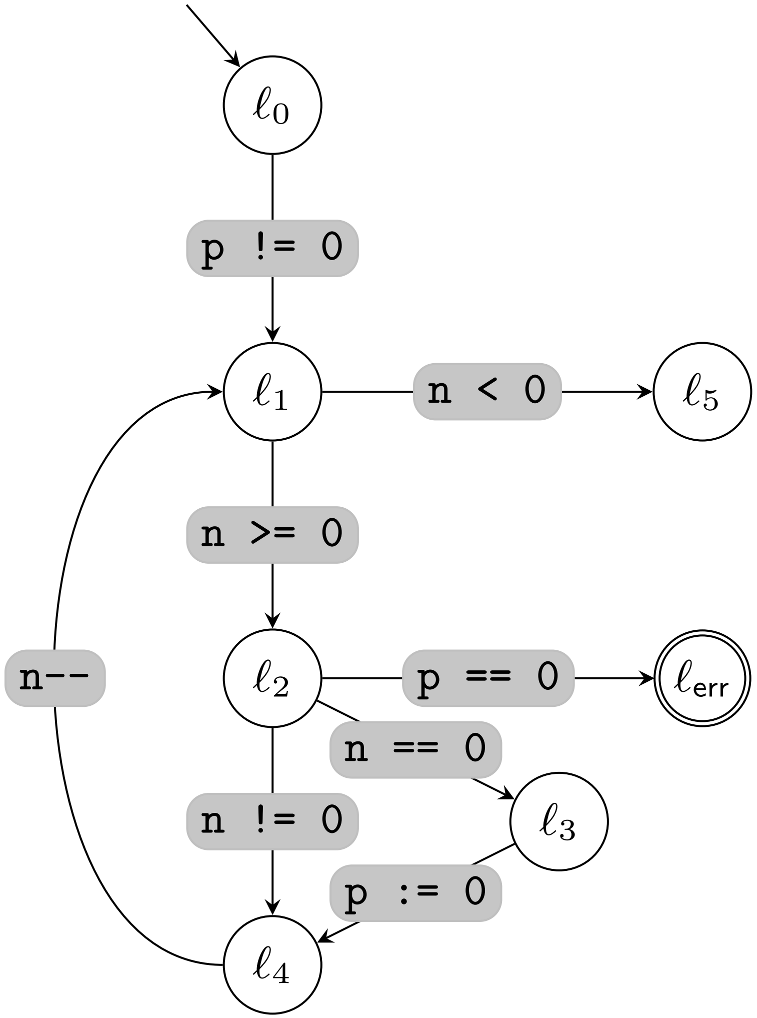


complex control? - just ignore it!




```
(p != 0)
(n >= 0)
(p == 0)
```

error trace: sequence of statements along an error path



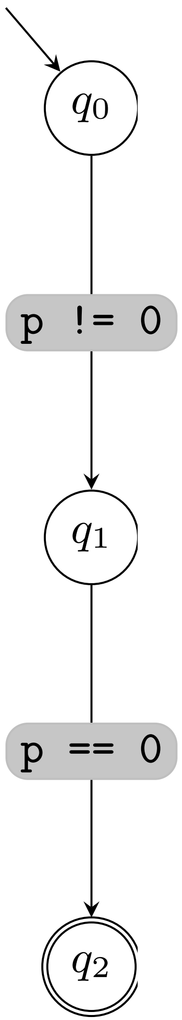
(p != 0)	(p != 0)
(n >= 0)	
(p == 0)	(p==0)

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

```
(p != 0)

(p==0)
```

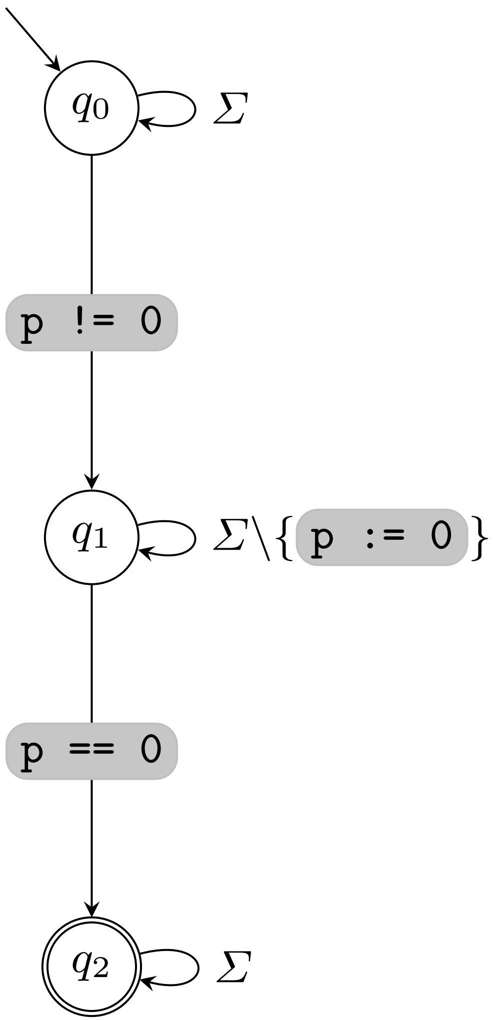
unsatisfiable core




```
(p != 0)

(p==0)
```

construct automaton from unsatisfiable core (step 1)

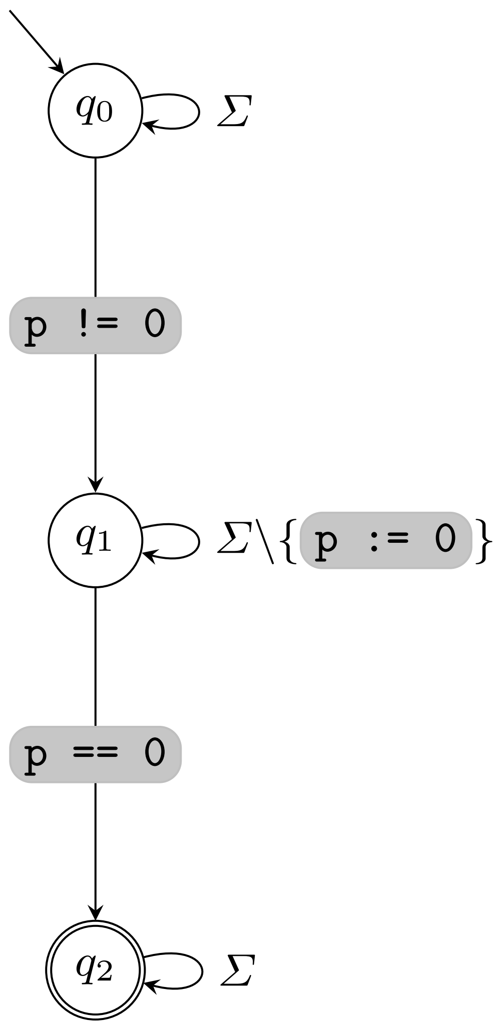


(p != 0)

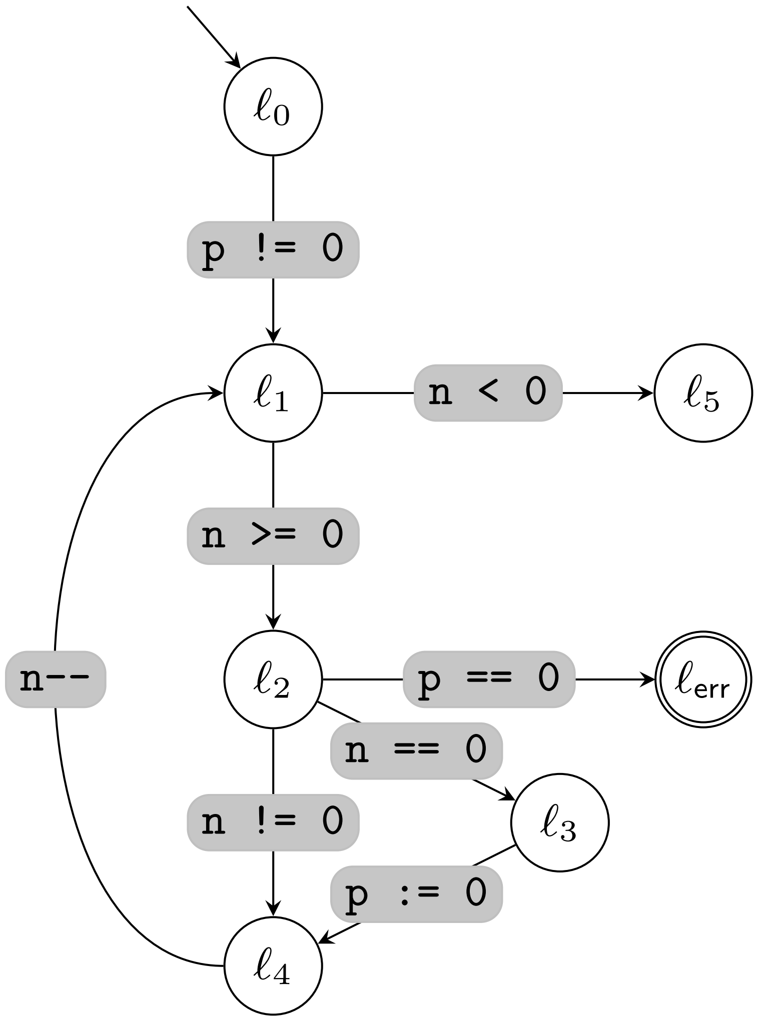
(p==0)

construct automaton from unsatisfiable core (step 2)

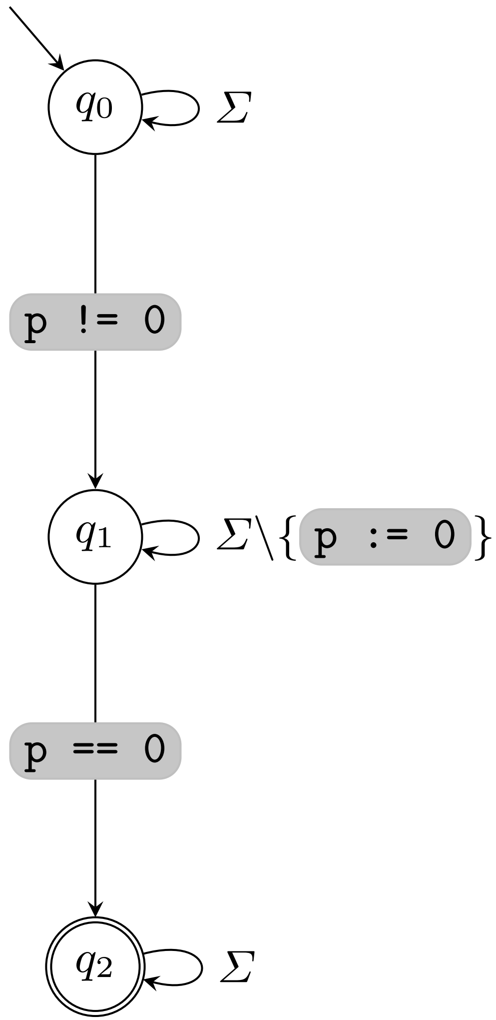
automaton constructed from unsatisfiability proof



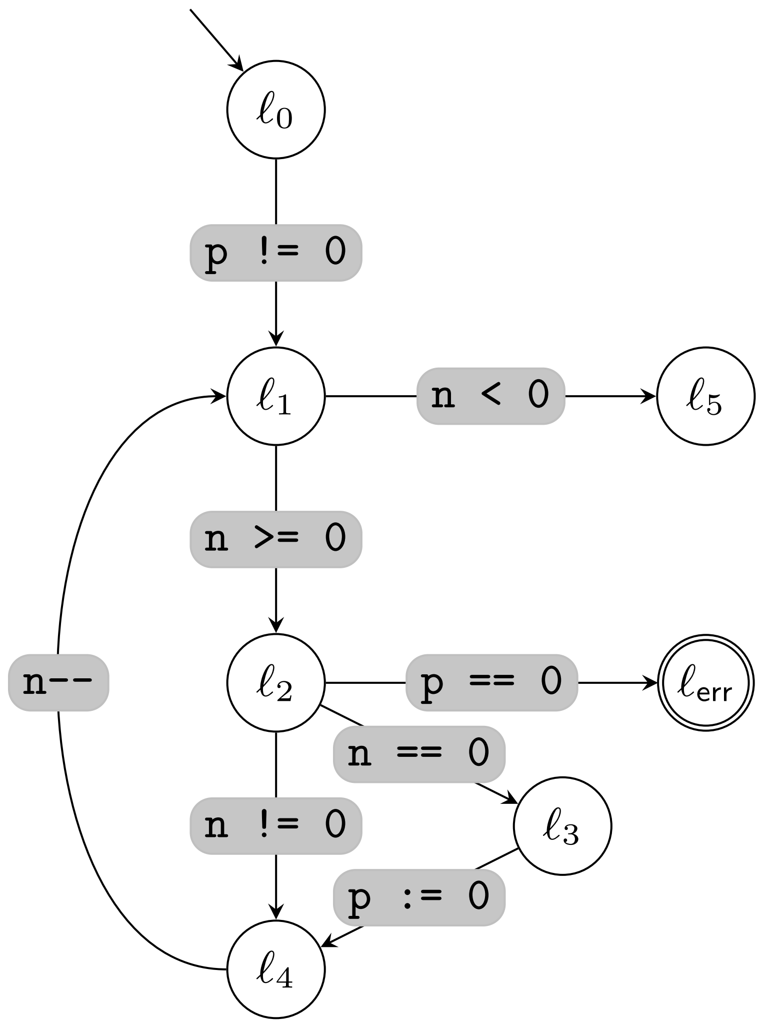
"data automaton" (ignores control of program)



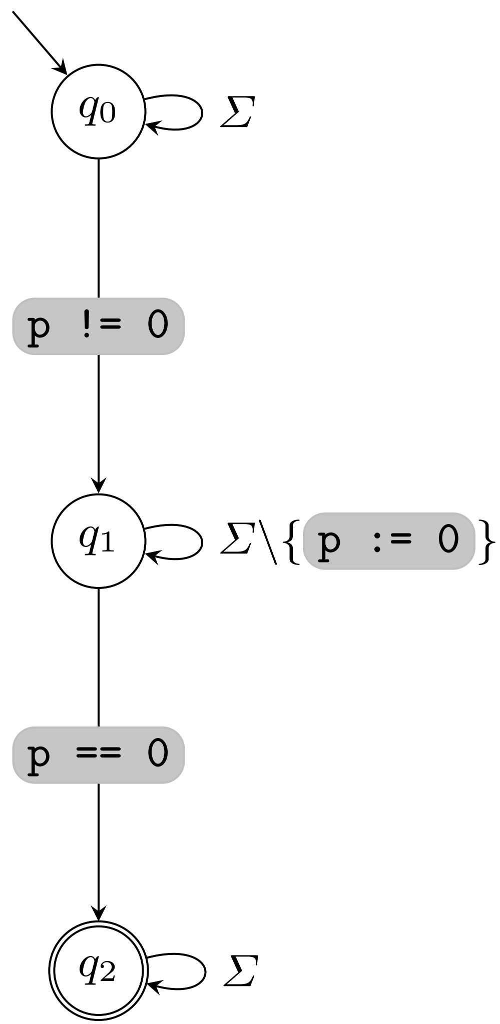
subset?



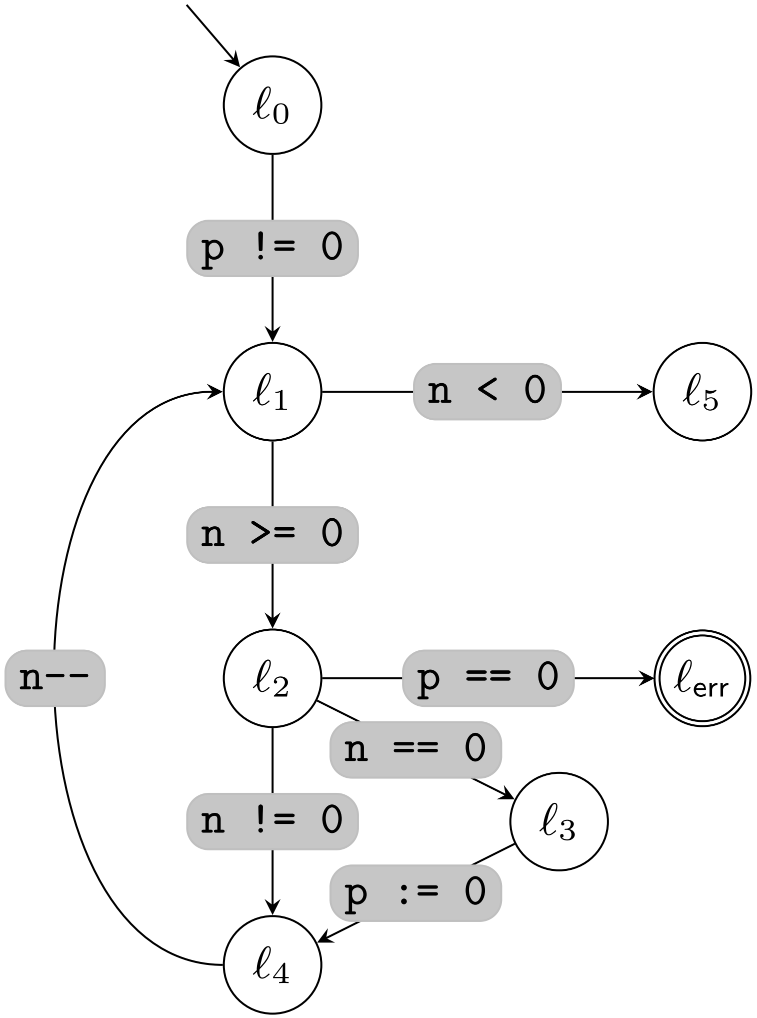
program automaton subset of data automaton ?



subset?

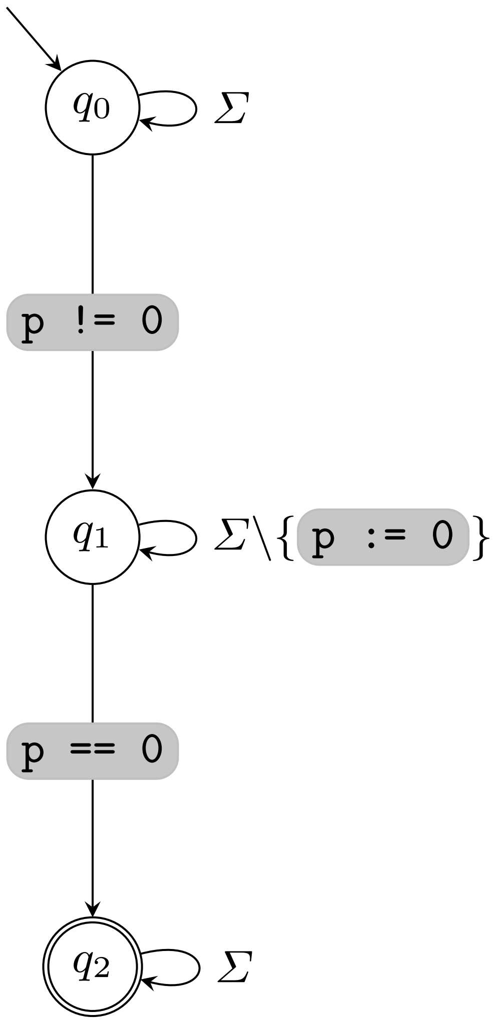


program automaton **not** subset of data automaton



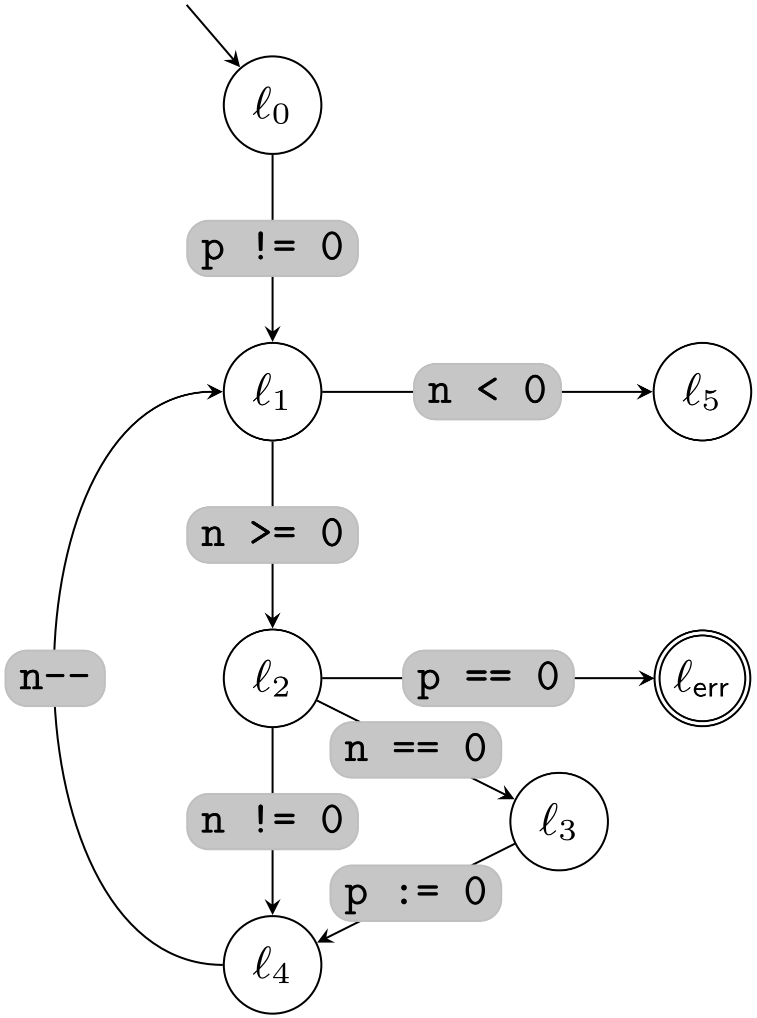
new error trace

```
(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



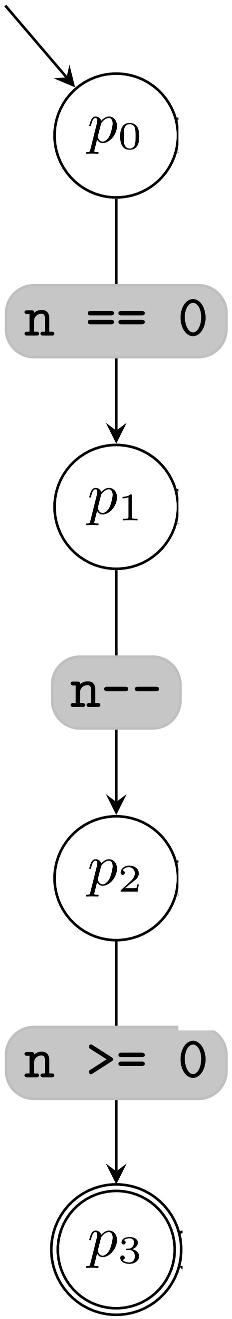
`(p != 0)`	
`(n >= 0)`	
`(n == 0)`	`(n == 0)`
`(p := 0)`	
`(n--)`	`(n--)`
`(n >= 0)`	`(n >= 0)`
`(p == 0)`	

trace infeasible, take unsatisfiable core

```
(n == 0)

(n--)
(n >= 0)
```

unsatisfiable core

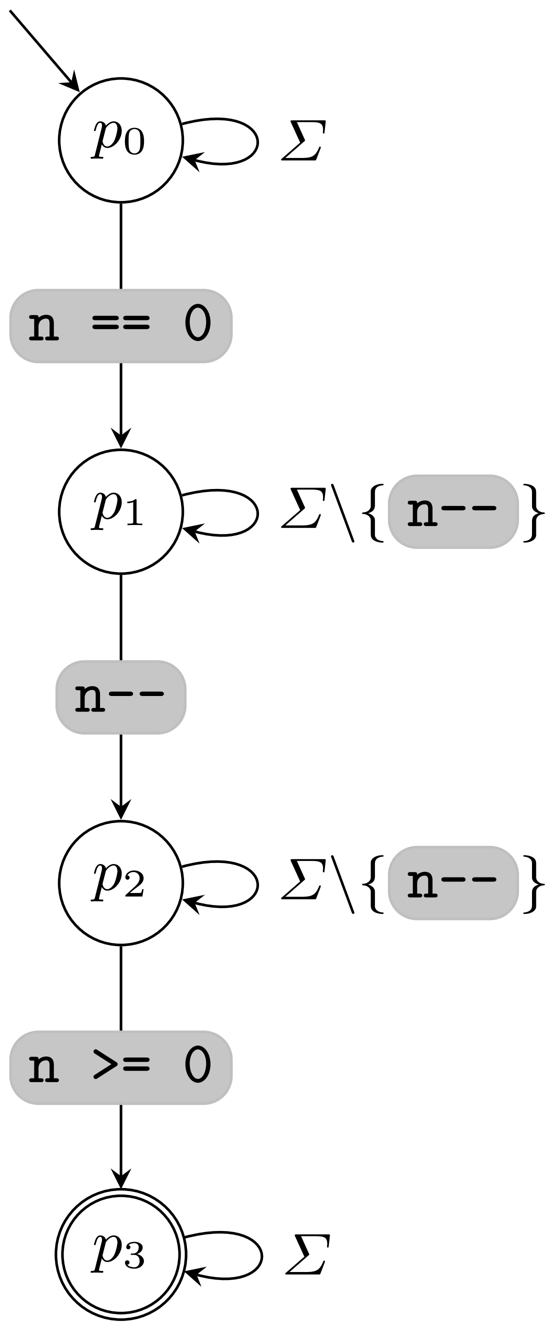


(n == 0)

(n--)

(n >= 0)

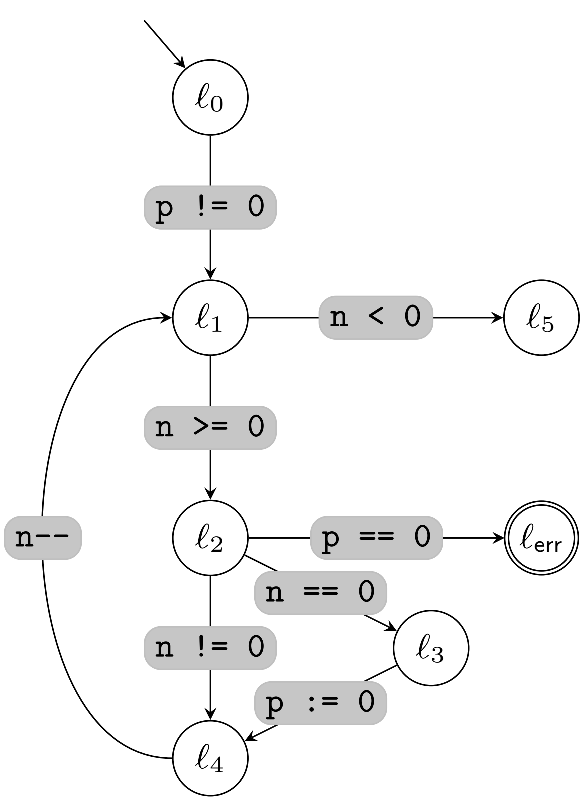
construct second data automaton from unsatisfiable core (step 1)



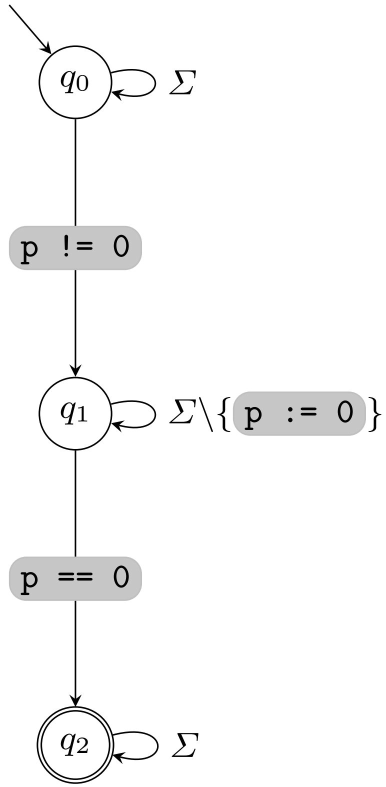

```
(n == 0)

(n--)
(n >= 0)
```

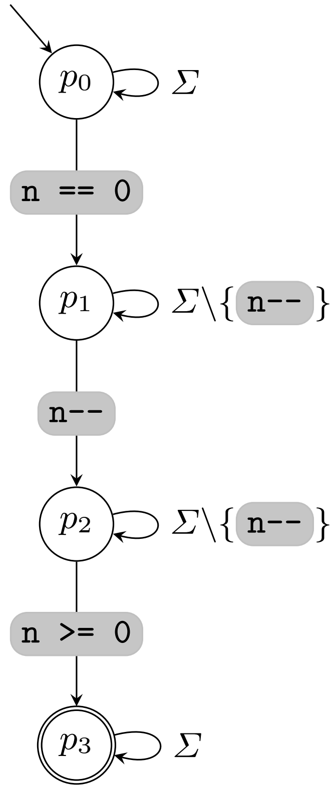
construct second data automaton from unsatisfiable core (step 2)



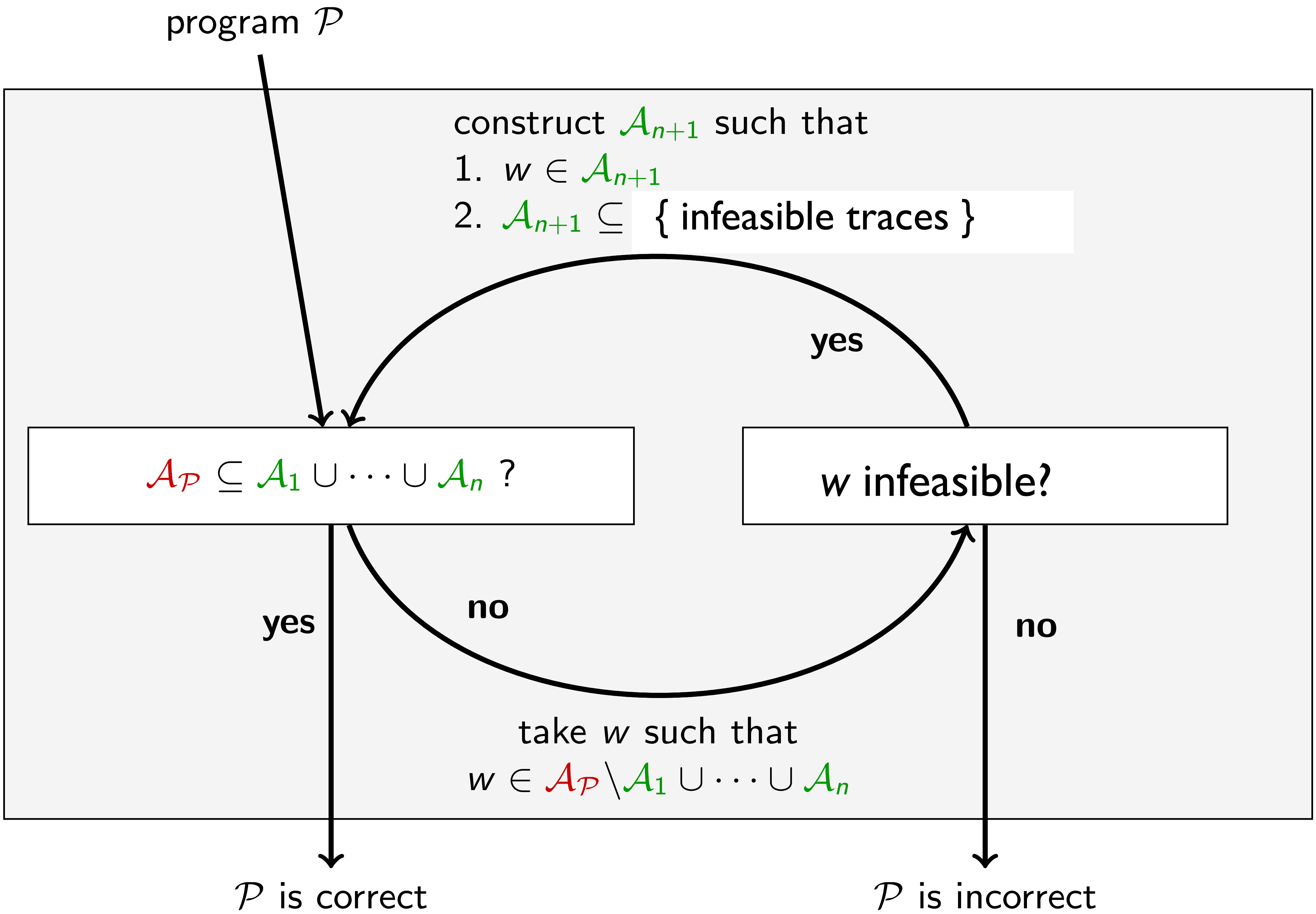
subset



$\cup$



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. $\mathcal{A}_{n+1} \subseteq$ { infeasible traces }
yes
$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$?
w infeasible?
yes
no
no
take w such that
$w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$
$\mathcal{P}$ is correct
$\mathcal{P}$ is incorrect

repeat

- take error trace in program

- check infeasibility of error trace

- construct data automaton from unsatisfiability proof

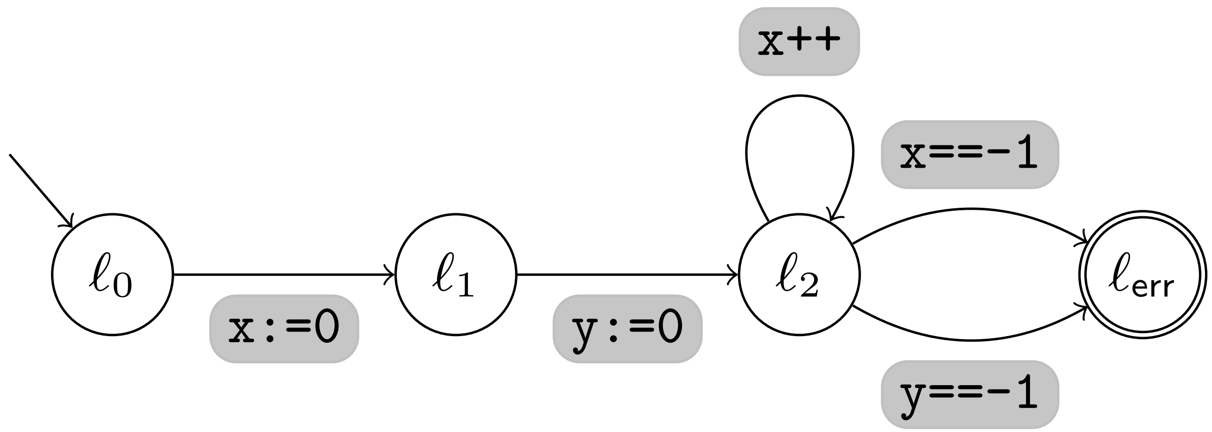
until program is subset of union of data automata

if automata are constructed from unsatisfiable core

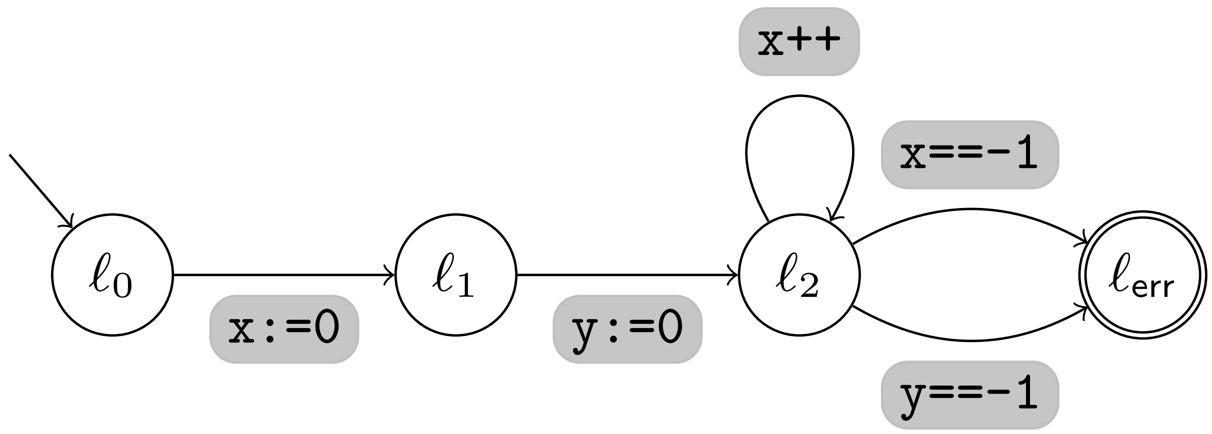
is the verification algorithm complete?

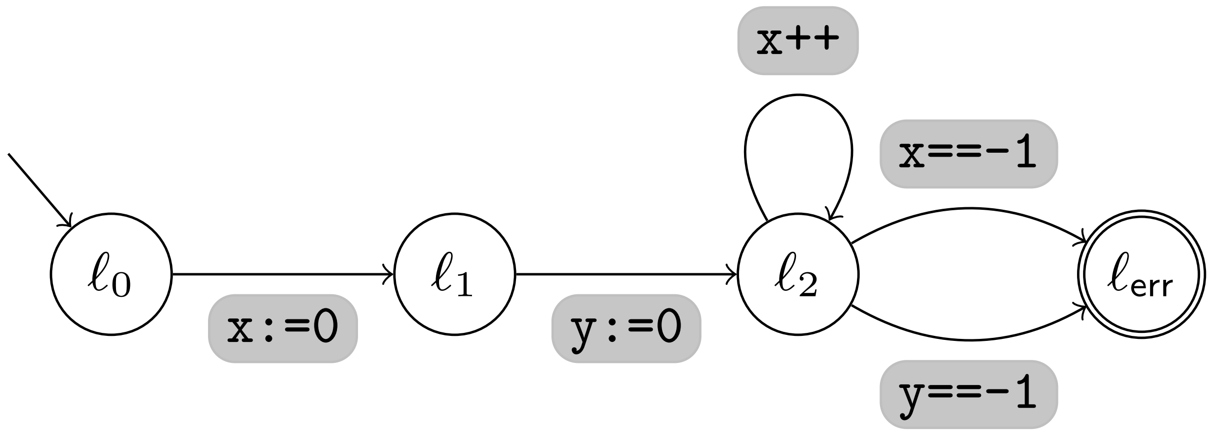
(can we prove every correct program correct?)

ℓ_0: `x := 0;`
ℓ_1: `y := 0;`
ℓ_2: `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
```

trace infeasible =

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

infeasibility proof with Hoare triples

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

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

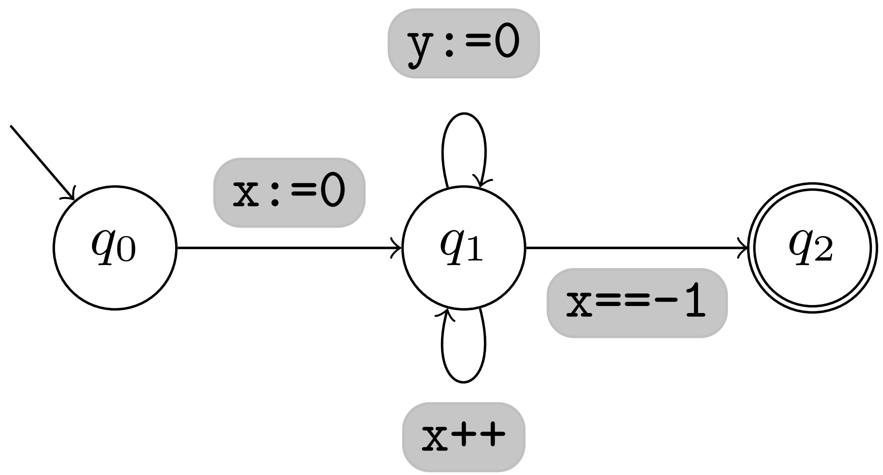
$$\{x \geq 0\}\ \texttt{x++}\ \{x \geq 0\}$$

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

automaton constructed from Hoare triples

$$\begin{array}{llll}\{\ \mathit{true}\ \} & \texttt{x:=0} & \{x \geq 0\} \\ \{x \geq 0\} & \texttt{y:=0} & \{x \geq 0\} \\ \{x \geq 0\} & \texttt{x++} & \{x \geq 0\} \\ \{x \geq 0\} & \texttt{x==-1} & \{\ \mathit{false}\ \}\end{array}$$

=>



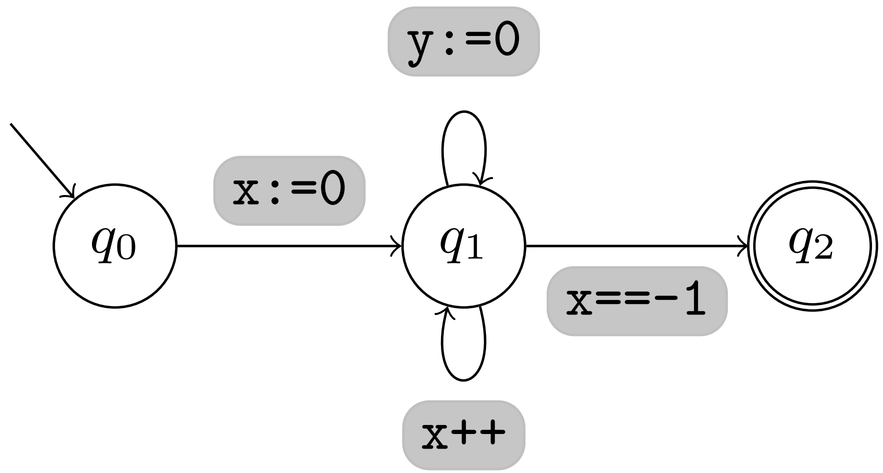
all traces accepted by automaton are infeasible

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

construction "Hoare proof $\longmapsto$ automaton"

Hoare triple $\longmapsto$ transition

$$\begin{array}{llll}
\{\ \mathit{true}\ \} & \texttt{x:=0} & \{x \geq 0\} \\
\{x \geq 0\} & \texttt{y:=0} & \{x \geq 0\} \\
\{x \geq 0\} & \texttt{x++} & \{x \geq 0\} \\
\{x \geq 0\} & \texttt{x==-1} & \{\ \mathit{false}\ \}
\end{array}$$



construction "Hoare proof $\longmapsto$ automaton"

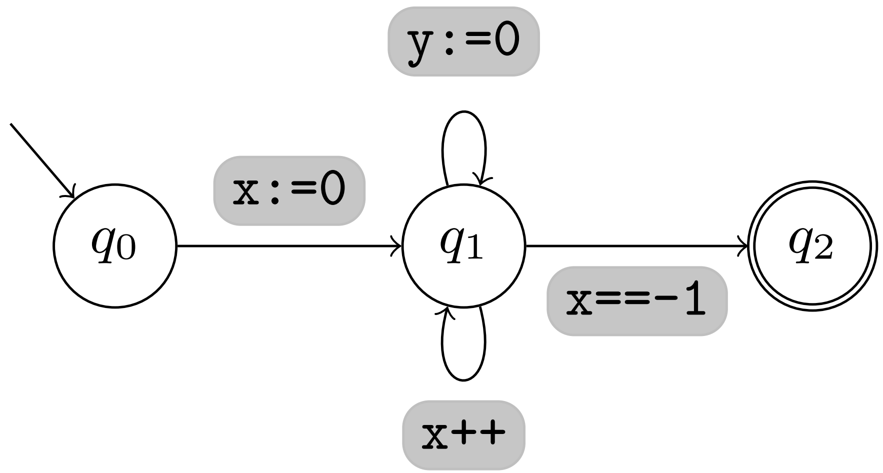
assertion $\longmapsto$ state

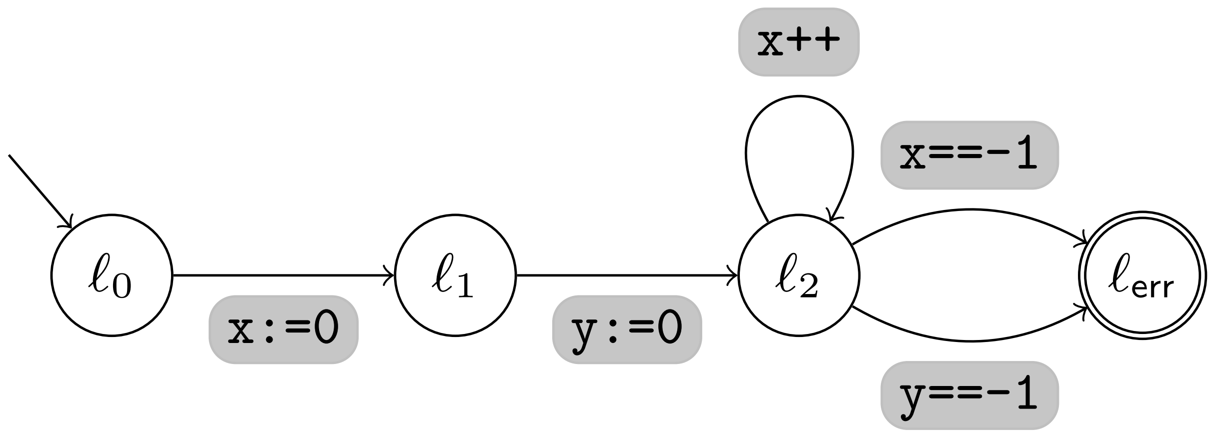
Hoare triple $\longmapsto$ transition

precondition $\longmapsto$ initial state

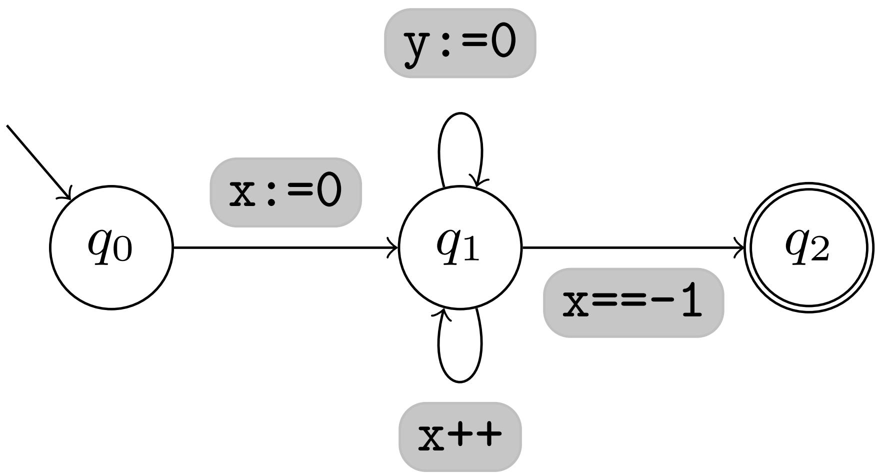
postcondition $\longmapsto$ final state

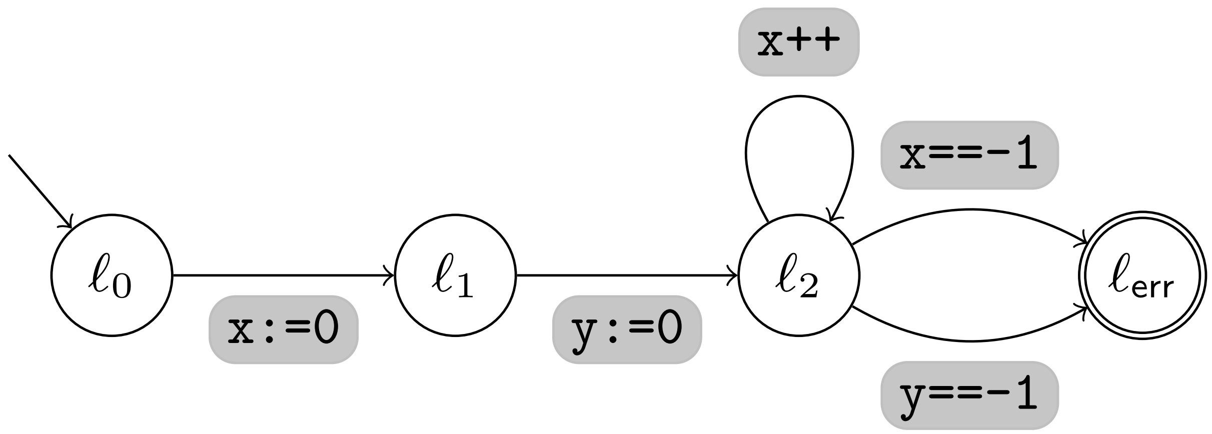
$$\begin{array}{lcl}
\{\ \mathit{true}\ \} & \texttt{x:=0} & \{x \geq 0\} \\
\{x \geq 0\} & \texttt{y:=0} & \{x \geq 0\} \\
\{x \geq 0\} & \texttt{x++} & \{x \geq 0\} \\
\{x \geq 0\} & \texttt{x==-1} & \{\ \mathit{false}\ \}
\end{array}$$





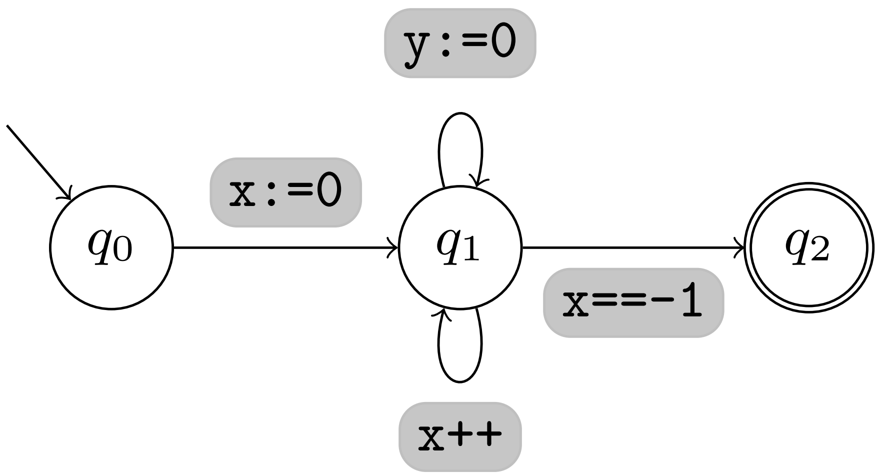
program automaton subset data automaton?





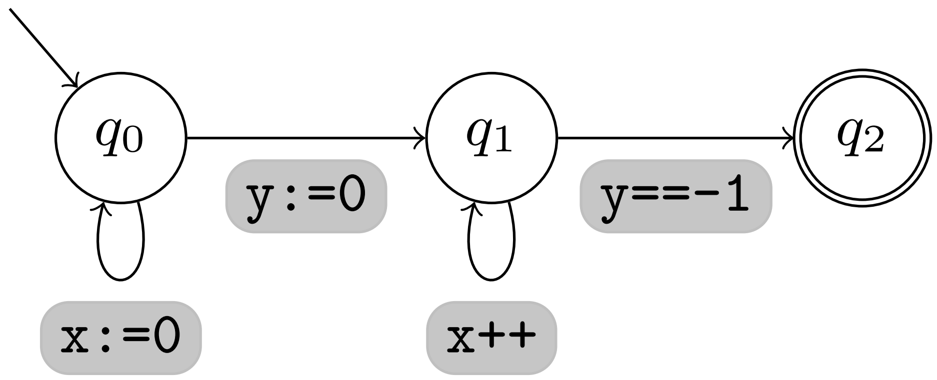
program automaton not subset of data automaton:

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



automaton from Hoare triples which prove infeasibility

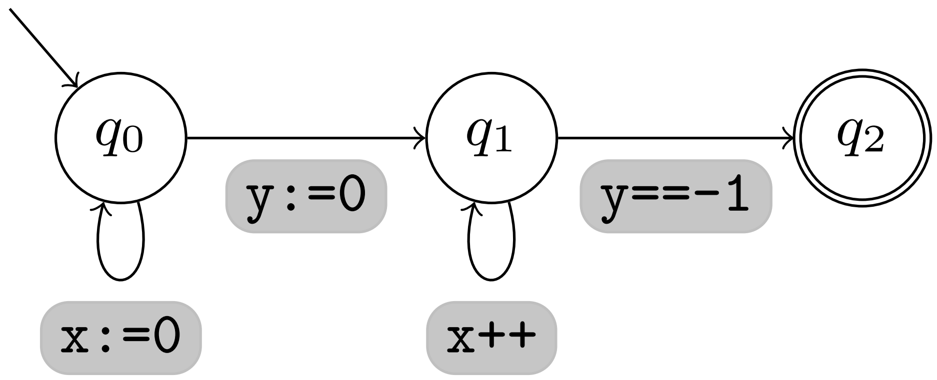
$\{\ true\ \}$ `x:=0` $\{\ true\ \}$
$\{\ true\ \}$ `y:=0` $\{y = 0\}$
$\{y = 0\}$ `x++` $\{y = 0\}$
$\{y = 0\}$ `y==-1` $\{\ false\ \}$



x

automaton from Hoare triples which prove infeasibility

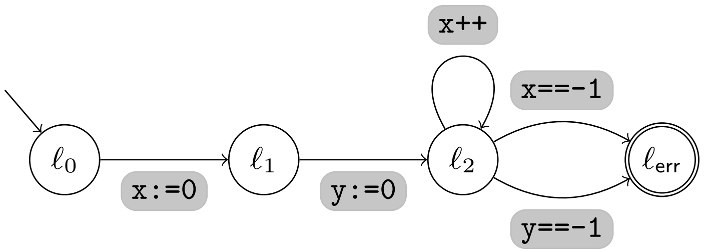
$$\{\ \mathit{true}\ \}\ \texttt{x:=0}\ \{\ \mathit{true}\ \}$$
$$\{\ \mathit{true}\ \}\ \texttt{y:=0}\ \{y = 0\}$$
$$\{y = 0\}\ \texttt{x++}\ \{y = 0\}$$
$$\{y = 0\}\ \texttt{y==-1}\ \{\ \mathit{false}\ \}$$



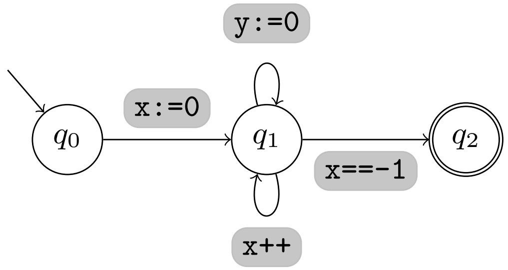
we can construct the same automaton from unsatisfiable core (since variable `x` does not appear in unsatisfiable core)

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

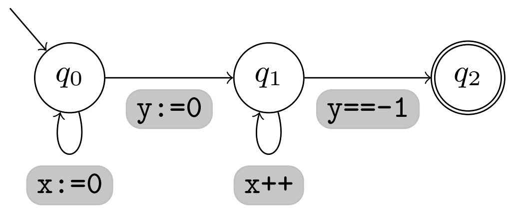
exists proof for infeasibility of trace
$\Rightarrow$
exists Hoare proof whose assertions are invariant under any statement that does not update a variable in unsatisfiable core



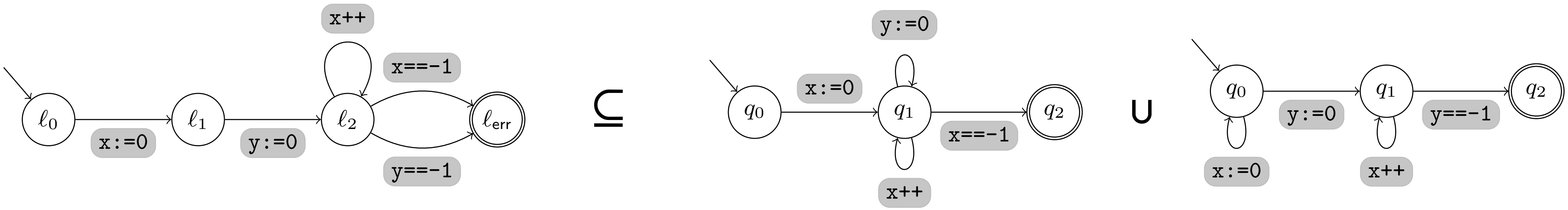
$\subseteq$



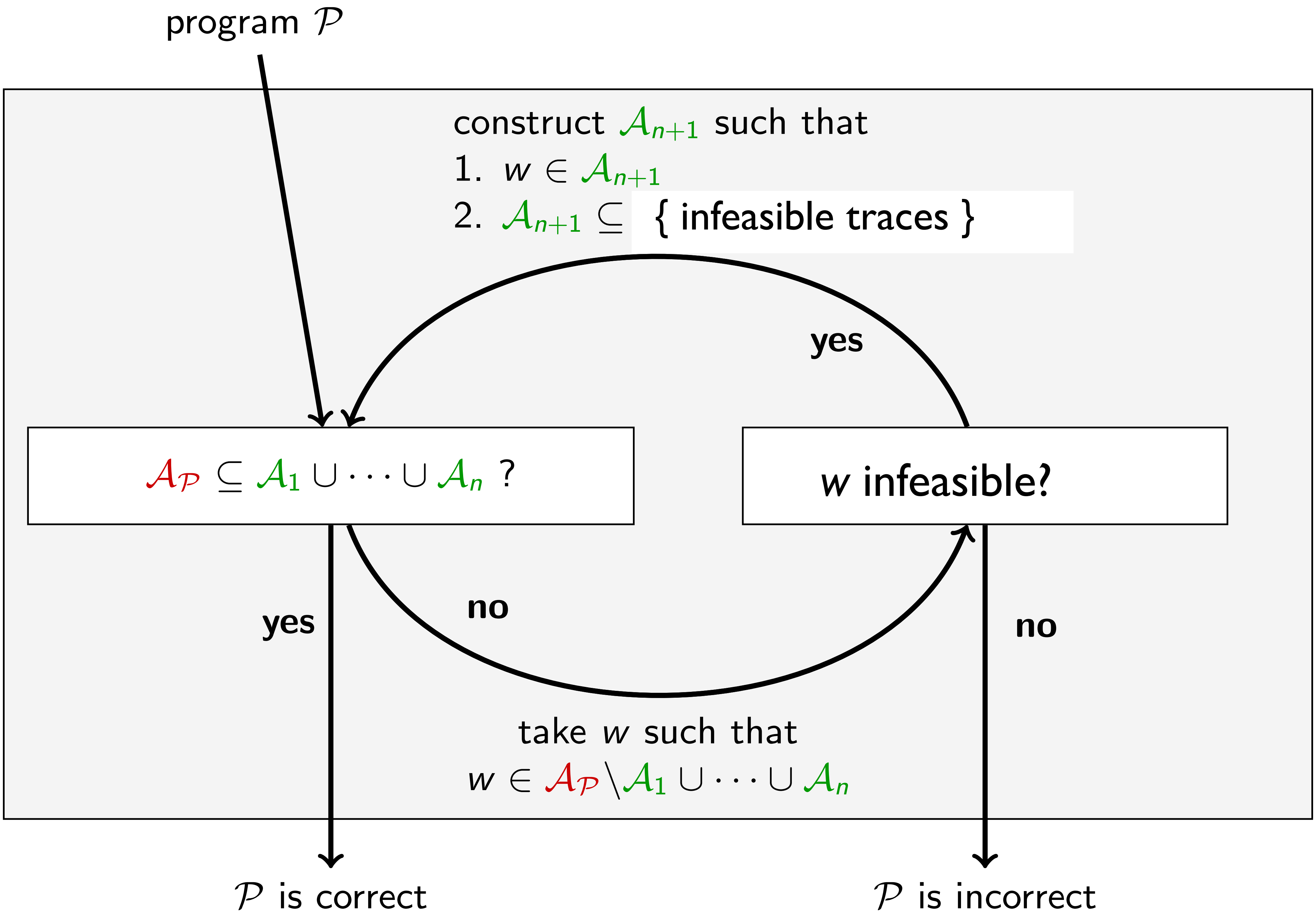
$\cup$



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. $\mathcal{A}_{n+1} \subseteq$ { infeasible traces }
yes
$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$?
w infeasible?
yes
no
no
take w such that
$w \in \mathcal{A}_{\mathcal{P}} \backslash \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$
$\mathcal{P}$ 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

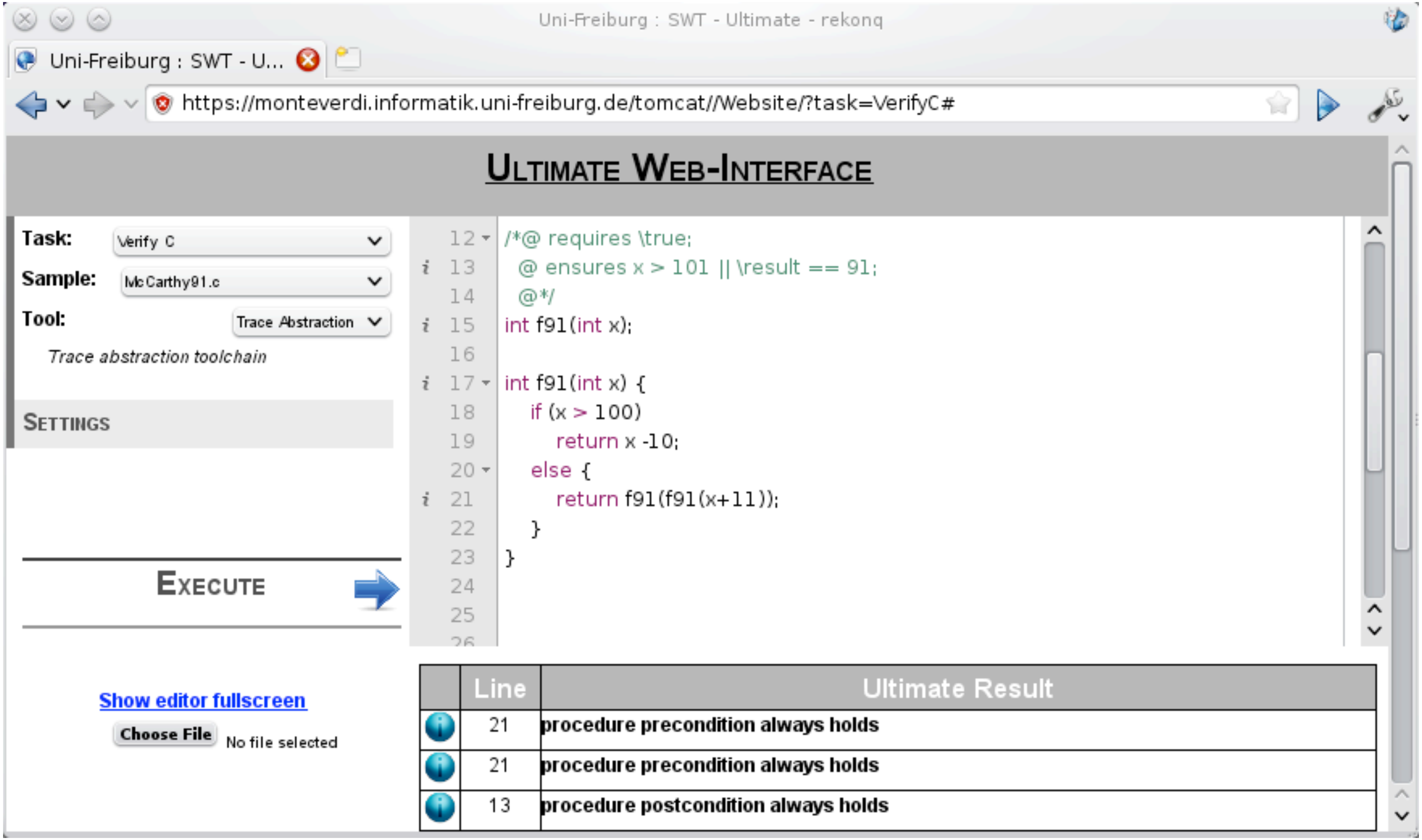
a trace is a program

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



C and Boogie, safety and termination

sequential programs .... nondeterministic finite automata

termination .... Buchi automata

recursion .... nested word automata

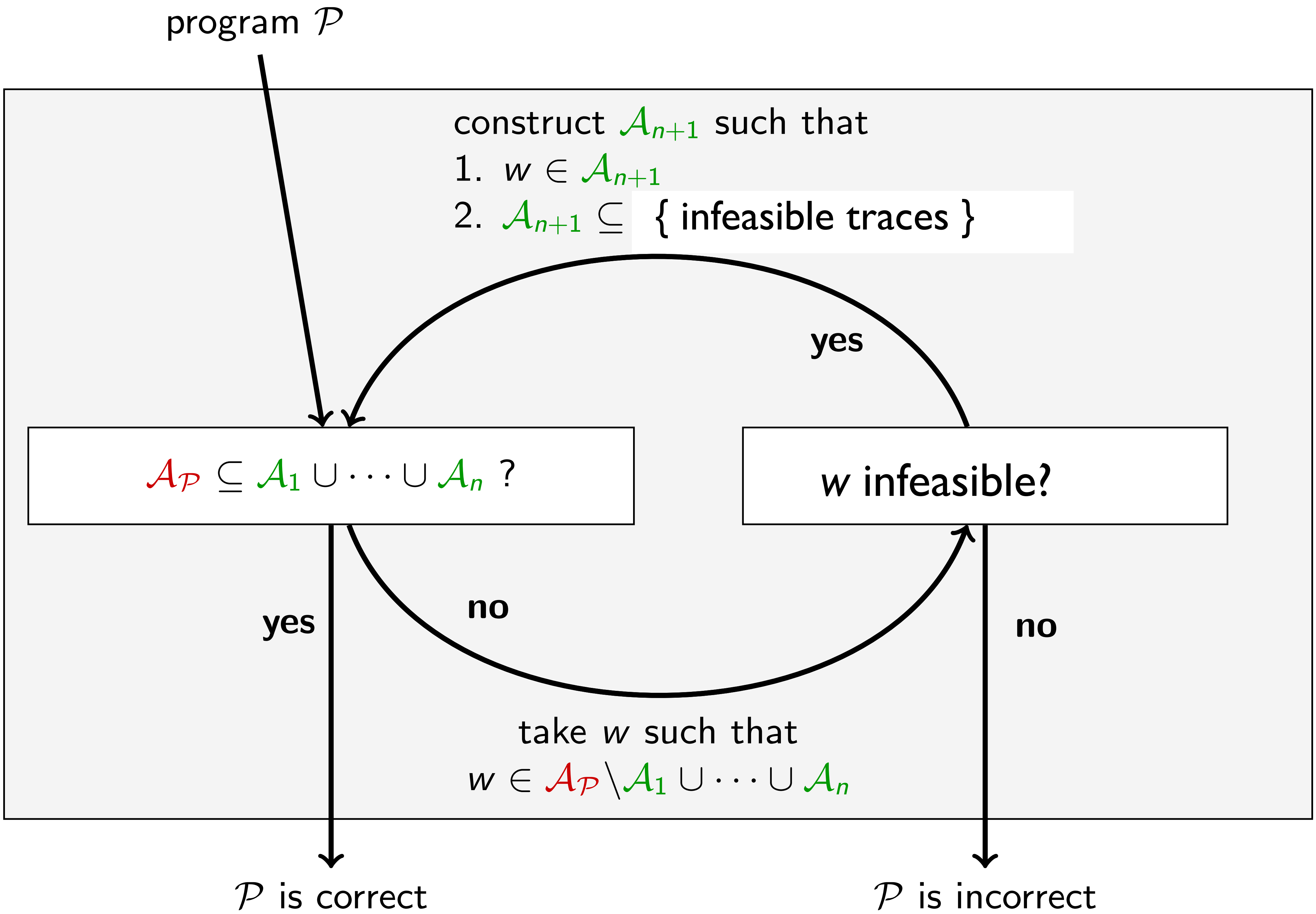
concurrency .... alternating finite automata

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. $\mathcal{A}_{n+1} \subseteq$ { infeasible traces }
yes
$\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$?
w infeasible?
yes
no
no
take w such that
$w \in \mathcal{A}_{\mathcal{P}} \setminus \mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n$
$\mathcal{P}$ 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