Proof Spaces

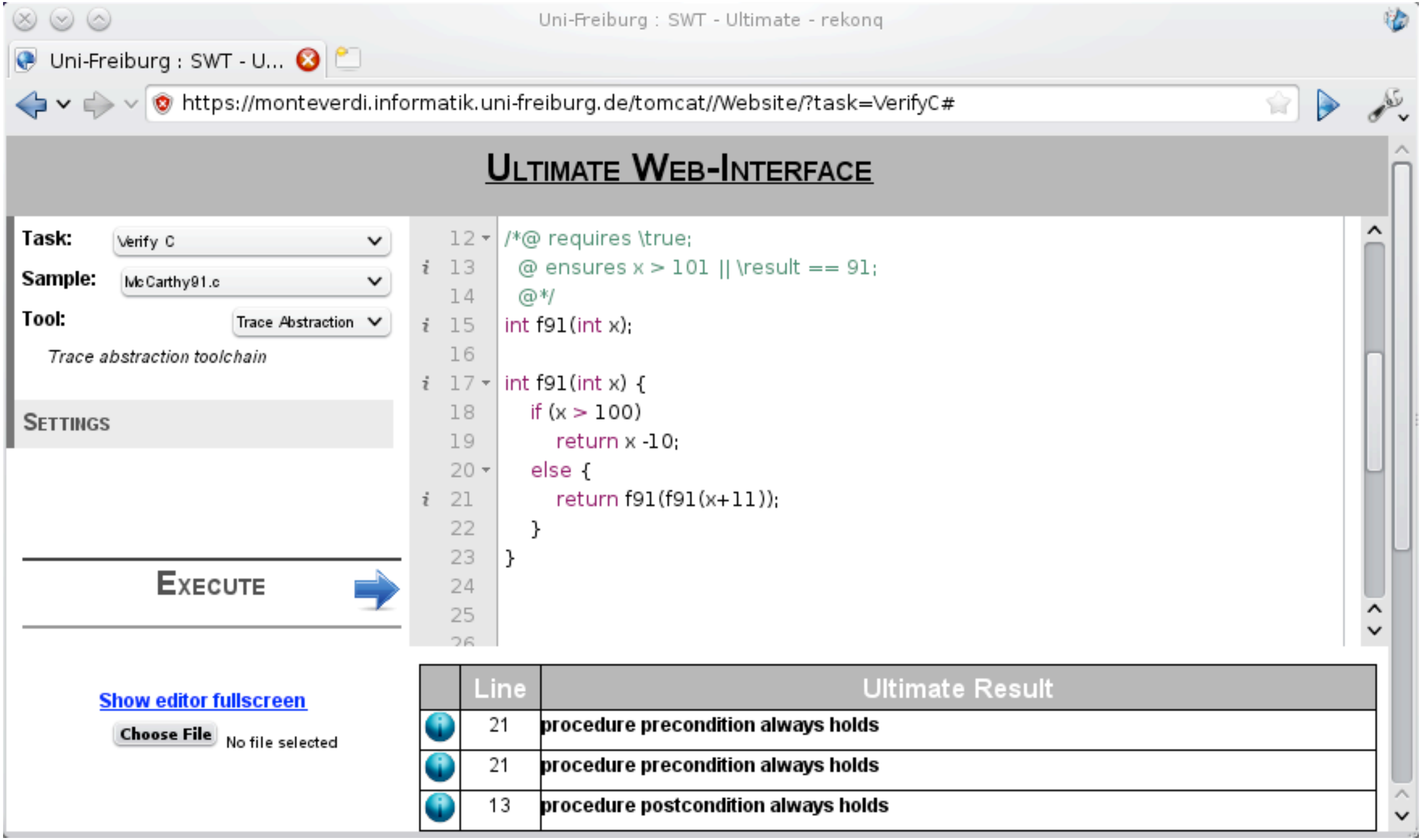
Andreas Podelski

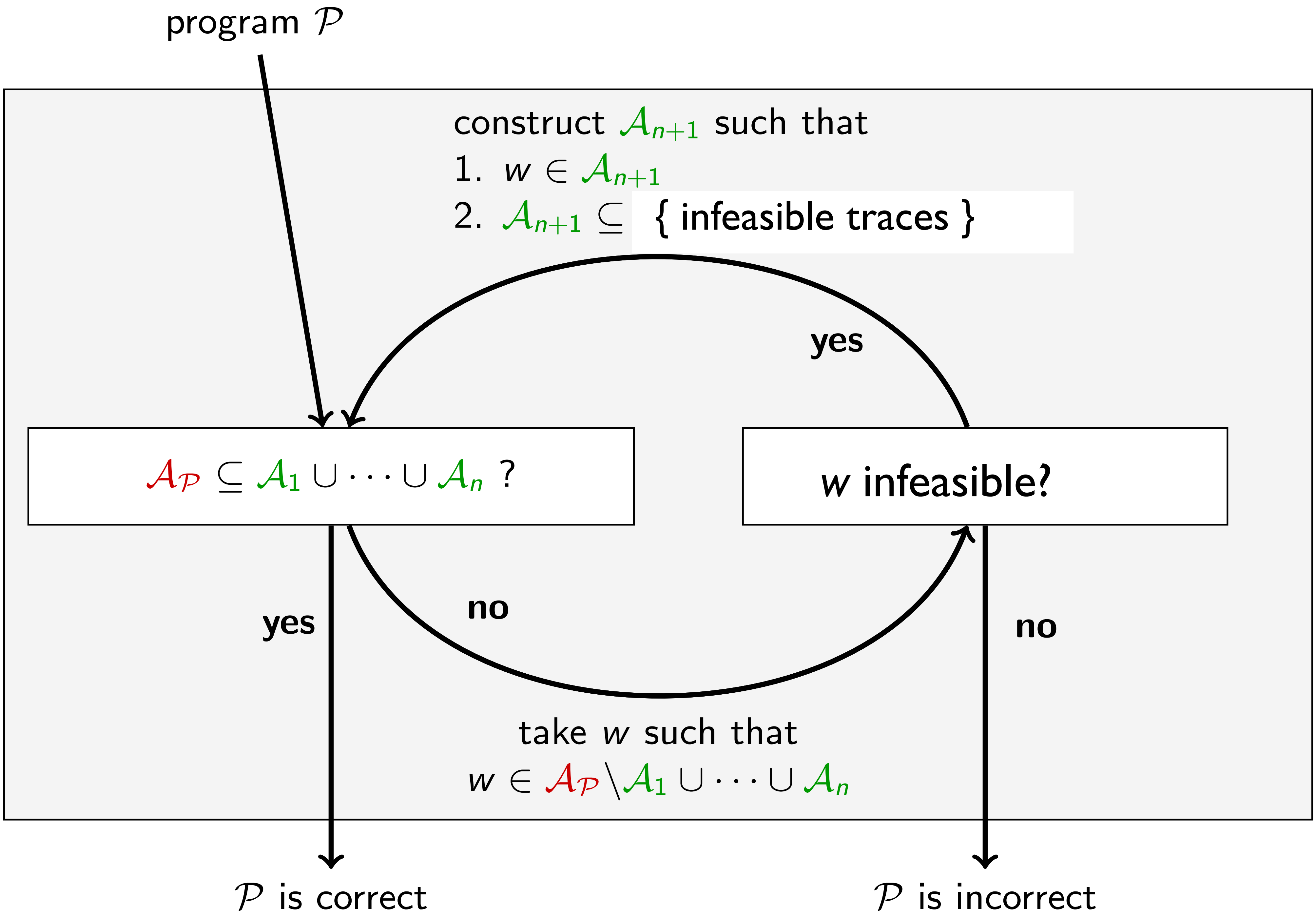
joint work with:
Matthias Heizmann, Jürgen Christ, Daniel Dietsch,
Jochen Hoenicke, Azadeh Farzan, Zachary Kincaid,
Markus Lindenmann, Betim Musa, Christian Schilling,
Alexander Nutz, Stefan Wissert, Evren Ermis

proof spaces

- new paradigm for automatic verification
- automata
- Marc Segelken: ω-Cegar [CAV 2007]
- verification for networked traffic control systems

Ultimate Automizer




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

Matthias Heizmann, Jürgen Christ, Daniel Dietsch, Jochen Hoenicke, Azadeh Farzan, Zachary Kincaid, Markus Lindenmann, Betim Musa, Christian Schilling, Alexander Nutz, Stefan Wissert, Evren Ermis

- Refinement of Trace Abstraction. SAS 2009
- Nested interpolants. POPL 2010
- Interpolant Automata. ATVA 2012
- Ultimate Automizer with SMTInterpol - (Competition Contribution). TACAS 2013
- Automata as Proofs. VMCAI 2013
- Inductive data flow graphs. POPL 2013
- Software Model Checking for People Who Love Automata. CAV 2013
- Ultimate Automizer with Unsatisfiable Cores - (Competition Contribution). TACAS 2014
- Termination Analysis by Learning Terminating Programs. CAV 2014
- Proofs that count. POPL 2014:
- Ultimate Automizer with Array Interpolation - (Competition Contribution). TACAS 2015
- Automated Program Verification. LATA 2015
- Fairness Modulo Theory: A New Approach to LTL Software Model Checking. CAV 2015
- Proof Spaces for Unbounded Parallelism. POPL 2015

invited talk: ETAPS 2012, ATVA 2012, VMCAI 2013, CAV 2013, LATA 2015

proof spaces

- new paradigm for automatic verification
- automata
- Marc Segelken: ω-Cegar [CAV 2007]
- verification for networked traffic control systems

The AVACS Vision

To Cover the Model- and Requirement Space of
Complex Safety Critical Systems

with Automatic Verification Methods

Giving Mathematical Evidence
of Compliance of Models

To Dependability, Coordination, Control
and Real-Time Requirements

Automating Verification of Cooperation, Control, and Design in Traffic Applications *

Werner Damm[1,2], Alfred Mikschl[1], Jens Oehlerking[1], Ernst-Rüdiger Olderog[1], Jun Pang[1], André Platzer[1], Marc Segelken[2], and Boris Wirtz[1]

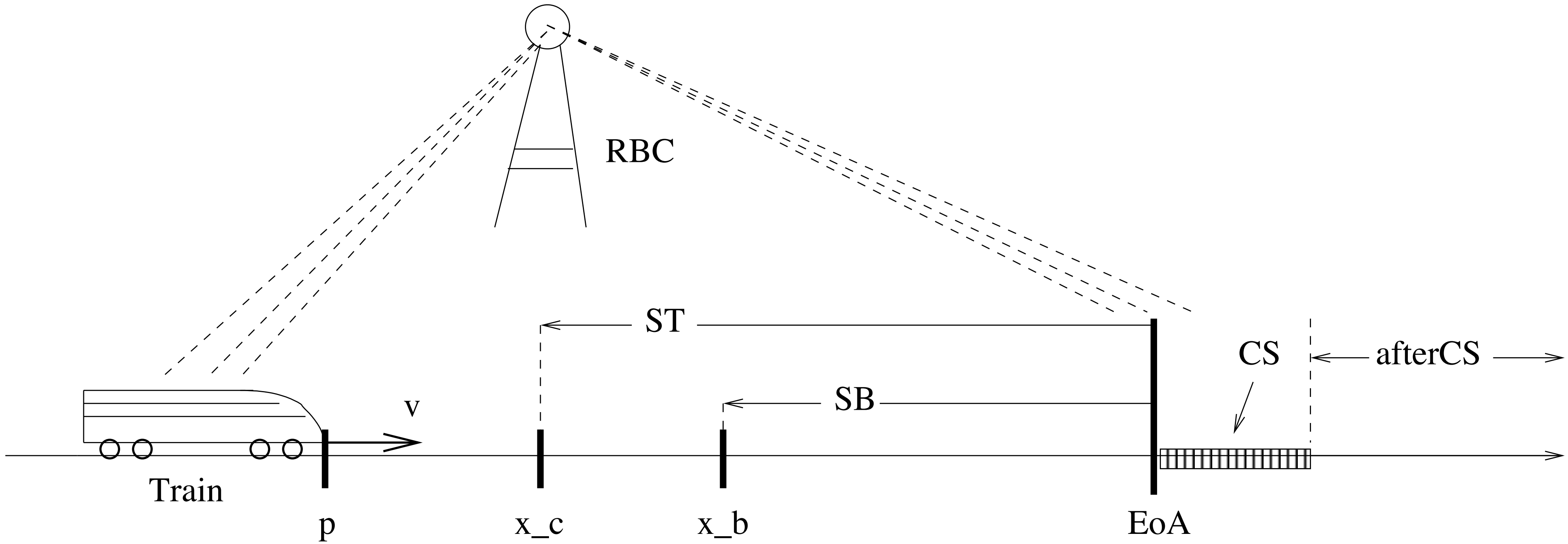


Fig. 4. Radio-based train control

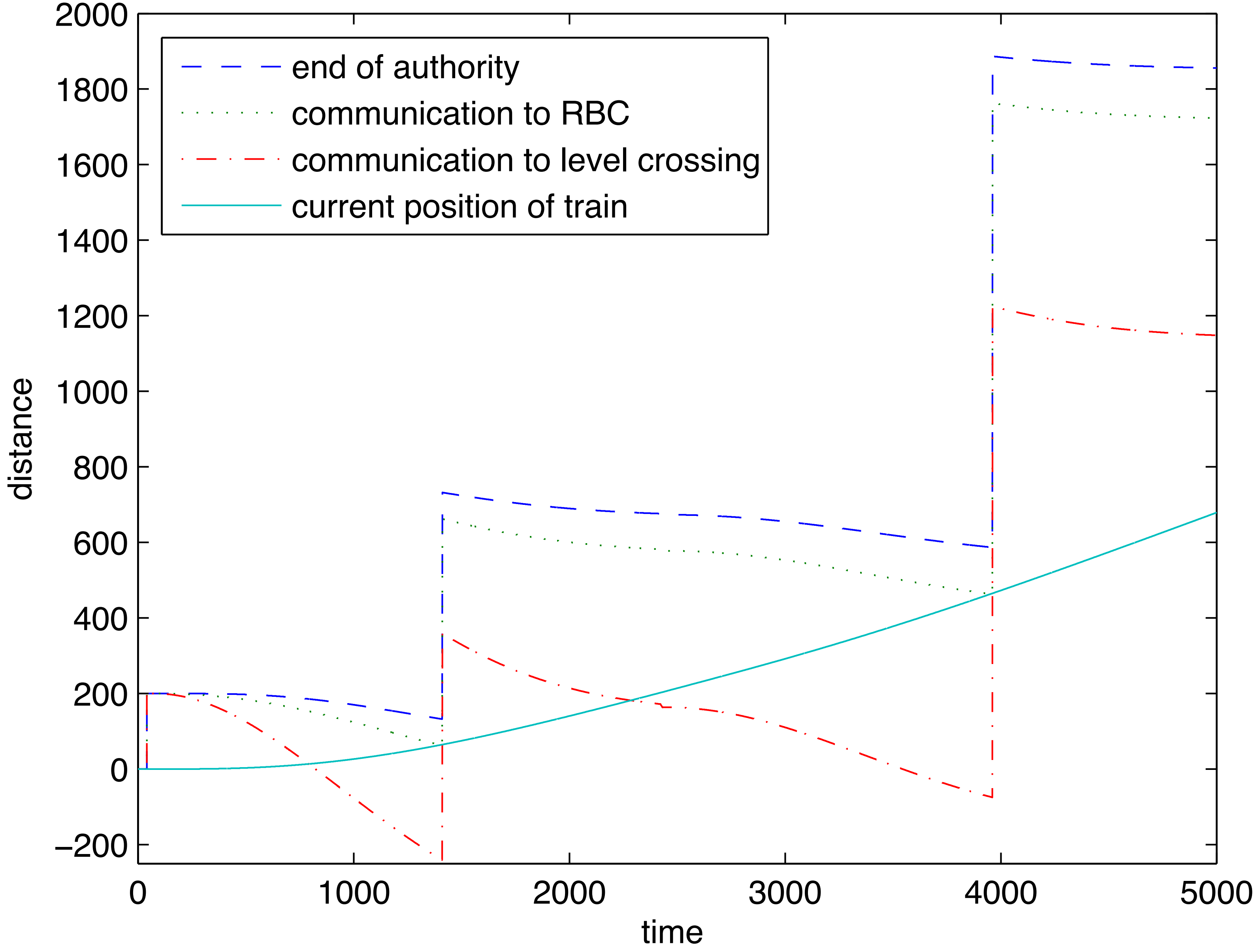


Fig. 5. Snapshot of dynamic calculations

holistic verification methodology

dedicated methods for:

- cooperation layer
- control layer
- design layer

model checking for discrete hybrid systems

- Lin AIGs
- ω-Cegar

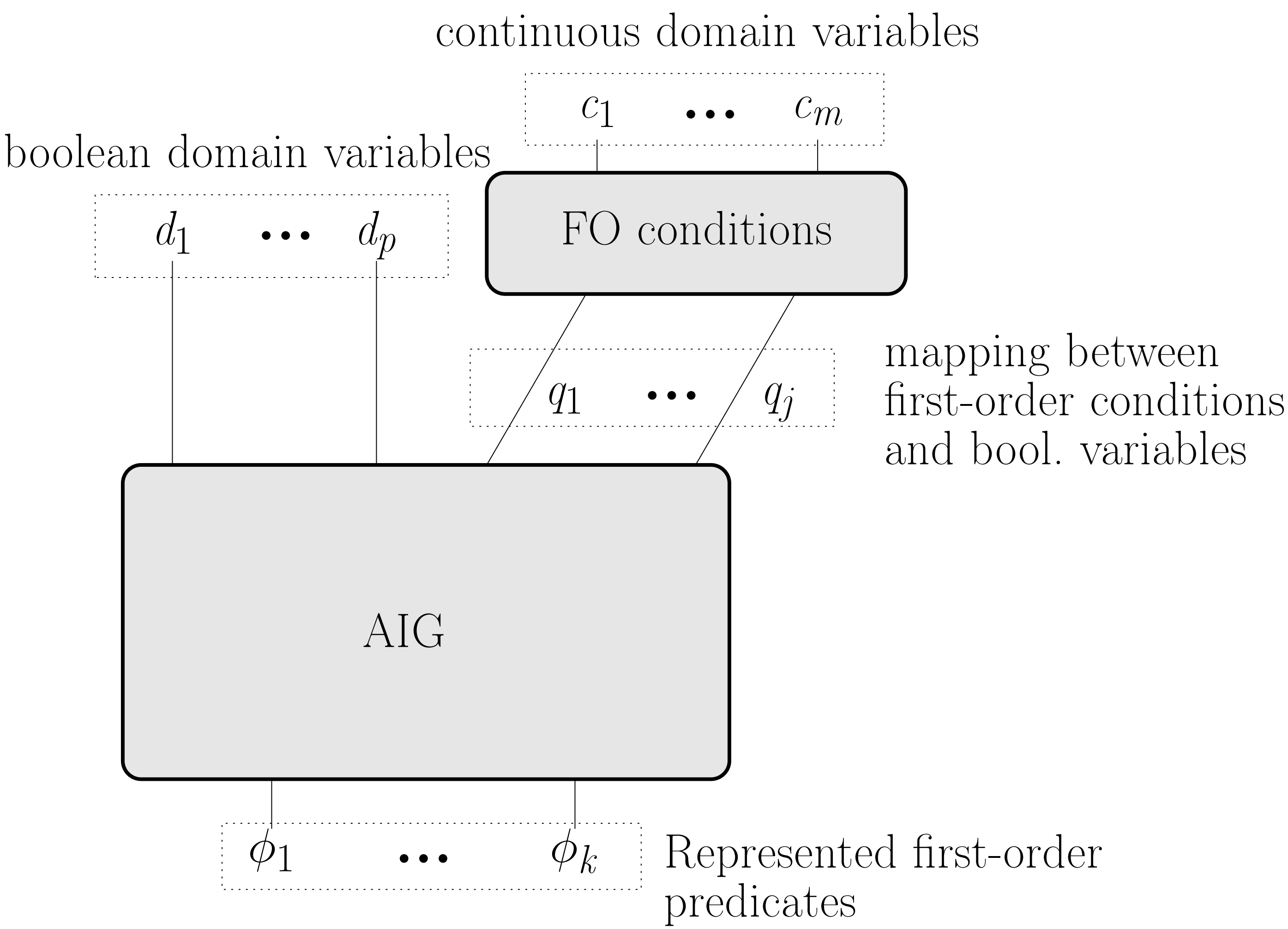


Fig. 17. The Lin-AIG structure

proof spaces

- new paradigm for automatic verification
- automata
- Marc Segelken: ω-Cegar [CAV 2007]
- verification for networked traffic control systems

Abstraction and Counterexample-guided Construction of ω-automata for Model Checking of Step-discrete linear Hybrid Models*

Marc Segelken

CAV 2007, LNCS 4590, pp. 433–448, 2007.

* This research was partially supported by the German Research Foundation (DFG) under contract SFB/TR 14 AVACS, see www.avacs.org

Construction of ω-automaton. Thus we follow a strategy of completely ruling out generalized conflicts by constructing an ω-automaton A_C that accepts all runs not containing any known conflict as a subsequence. Considering partial regulation laws as atomic characters and C as the set of all previously detected generalized conflicts, the behavior of A_C can be described by an LTL formula:

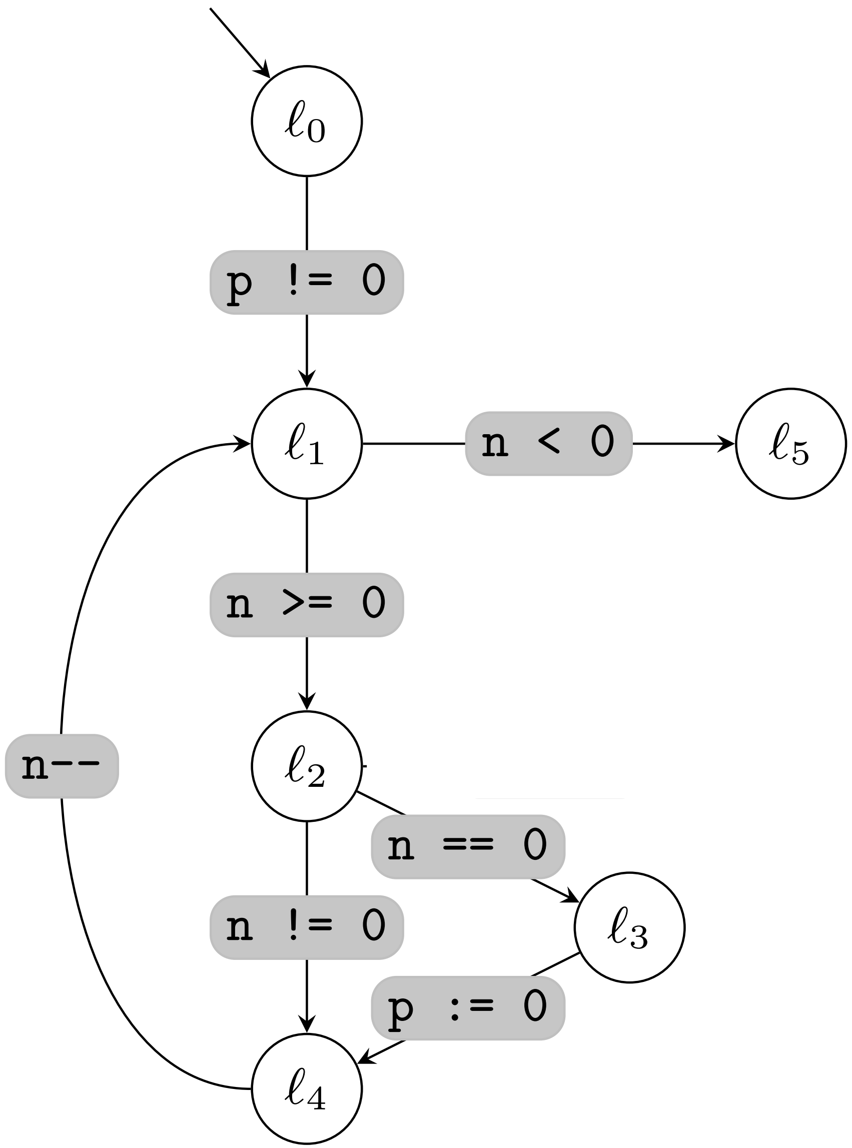
$$A_C \models \neg\mathbf{F} \bigvee_{(\rho_1,\rho_2,\ldots,\rho_k)\in C} (\rho_1 \wedge \mathbf{X}(\rho_2 \wedge \mathbf{X}(\ldots \wedge \mathbf{X}\rho_n))) \tag{21}$$

automata over an unusual alphabet ...

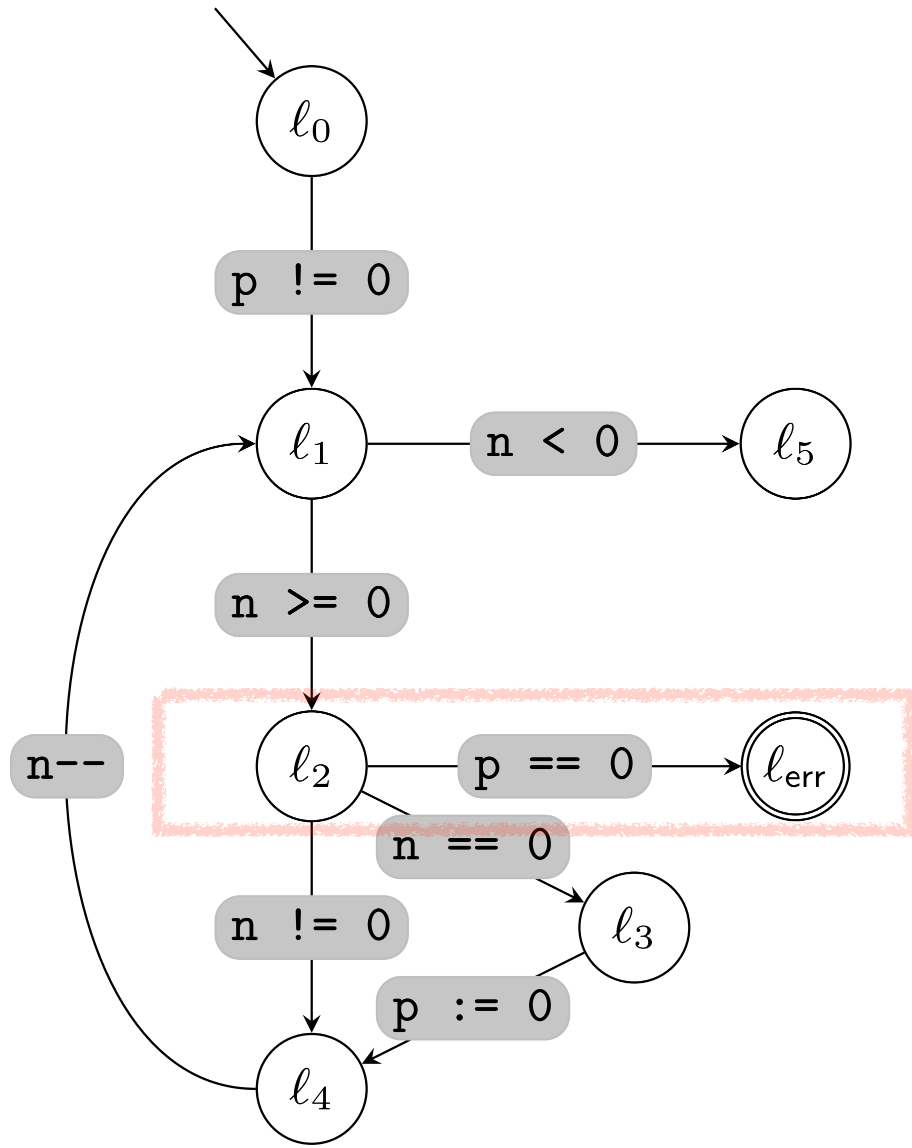
proof spaces

- new paradigm for automatic verification
- automata
- Marc Segelken: ω-Cegar [CAV 2007]
- verification for networked traffic control systems

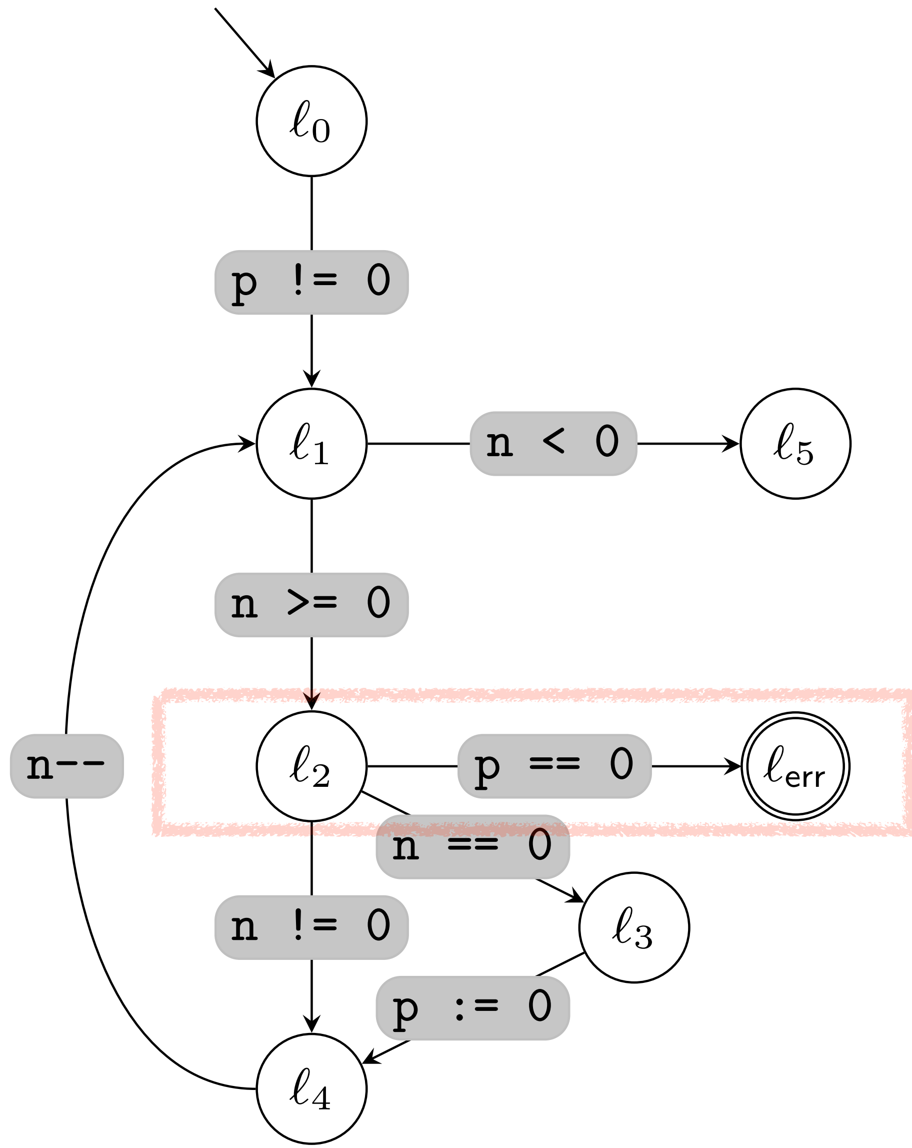
```
ℓ0: assume p != 0;

ℓ1: while(n >= 0)
    {
ℓ2:

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

ℓ5:
```



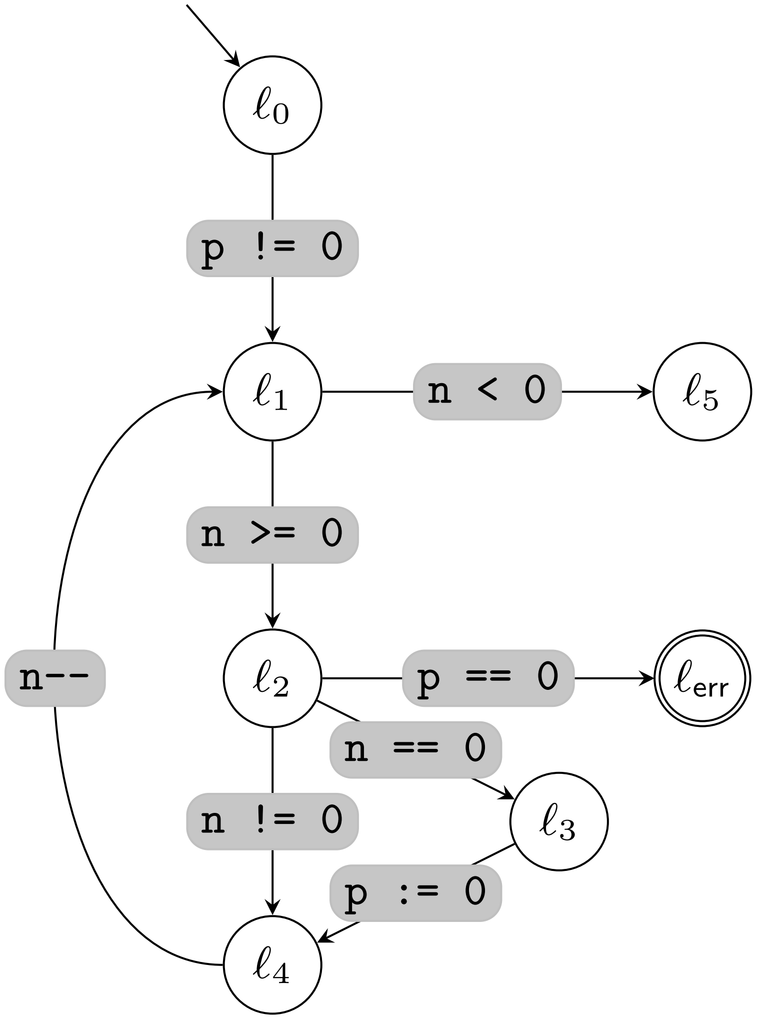
```
ℓ0: assume p != 0;

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:
```

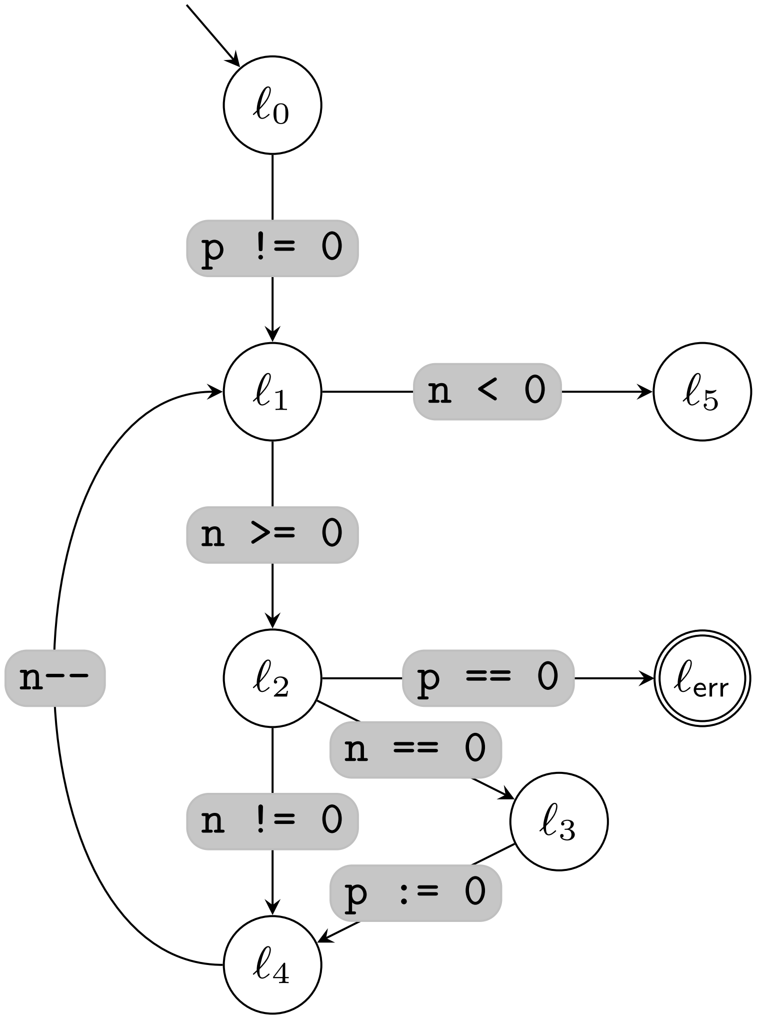


no execution violates assertion = no execution reaches error location

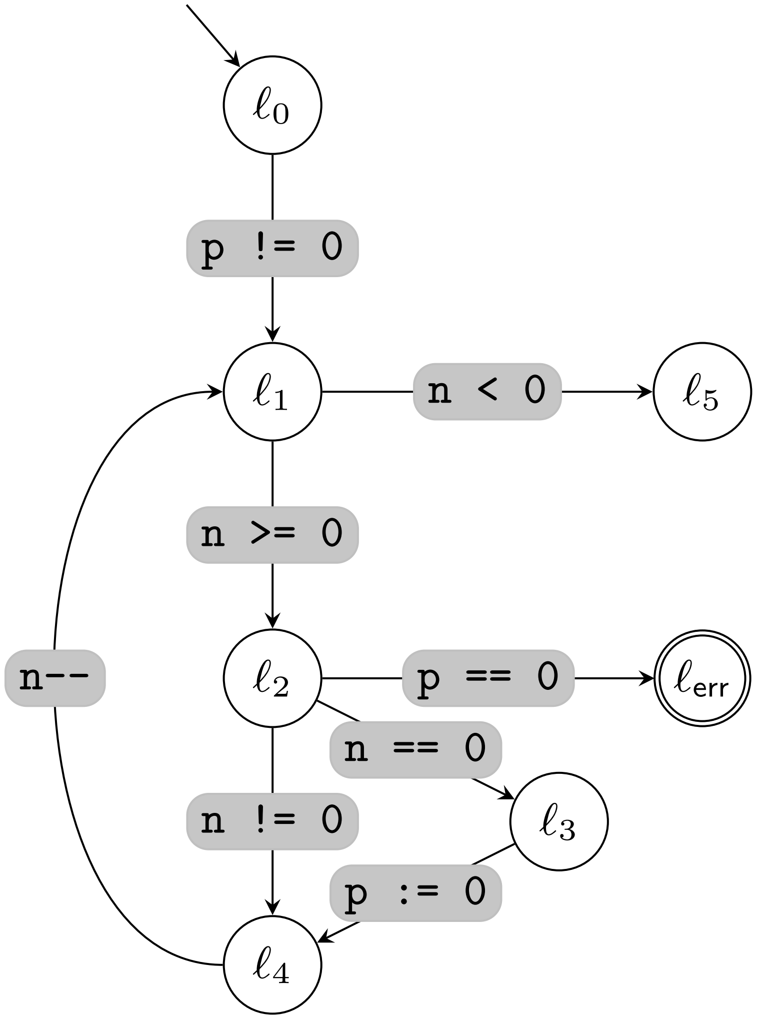


automaton

alphabet: {statements}




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

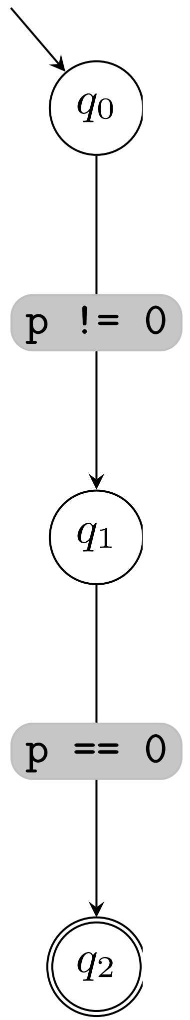


(p != 0) (p != 0)

(n >= 0)

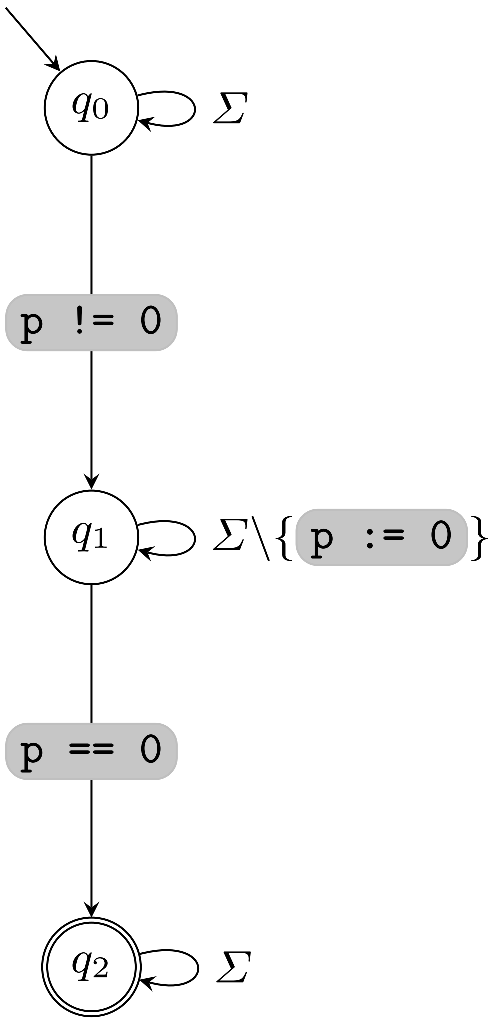
(p == 0) (p==0)

```
(p != 0)

(p==0)
```



(p != 0)

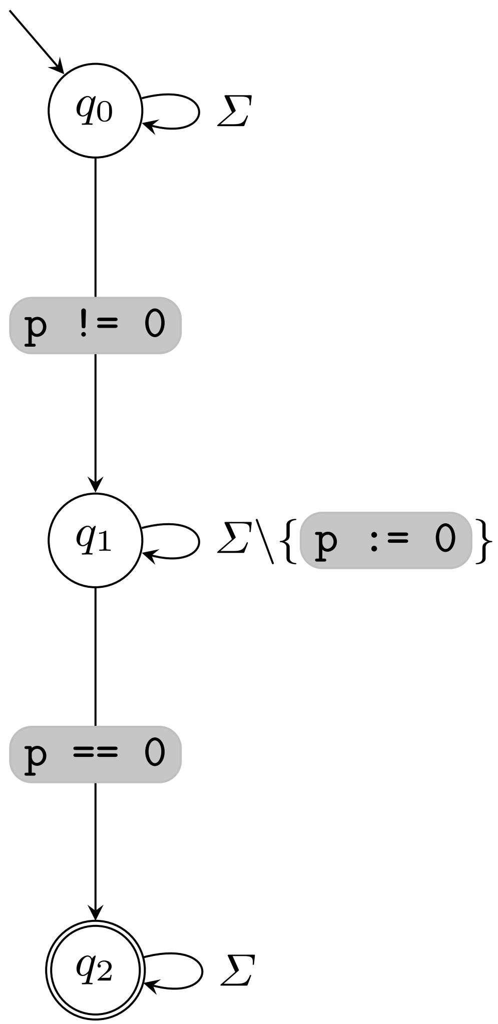
(p==0)



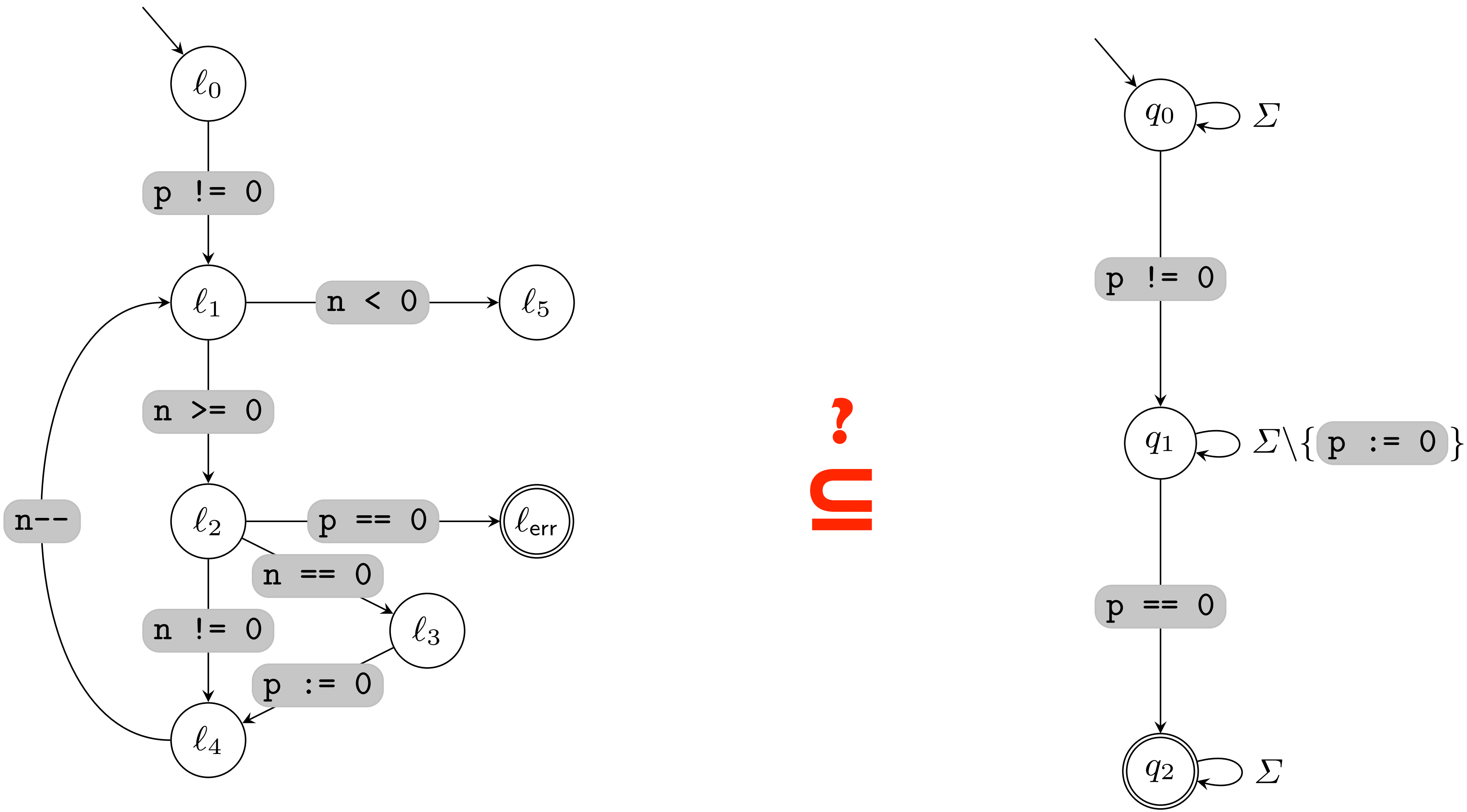
(p != 0)

(p==0)

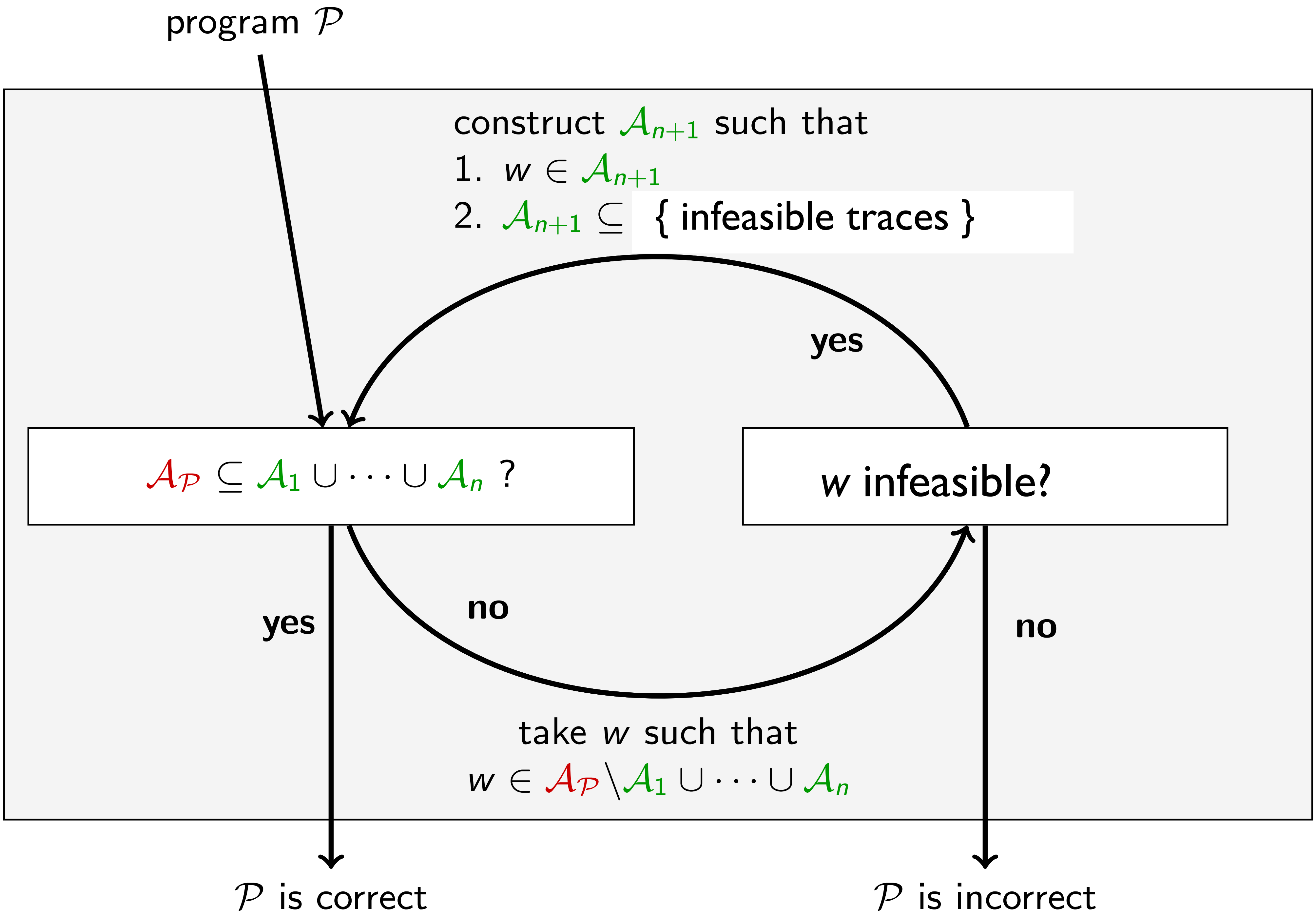
automaton constructed from unsatisfiability proof

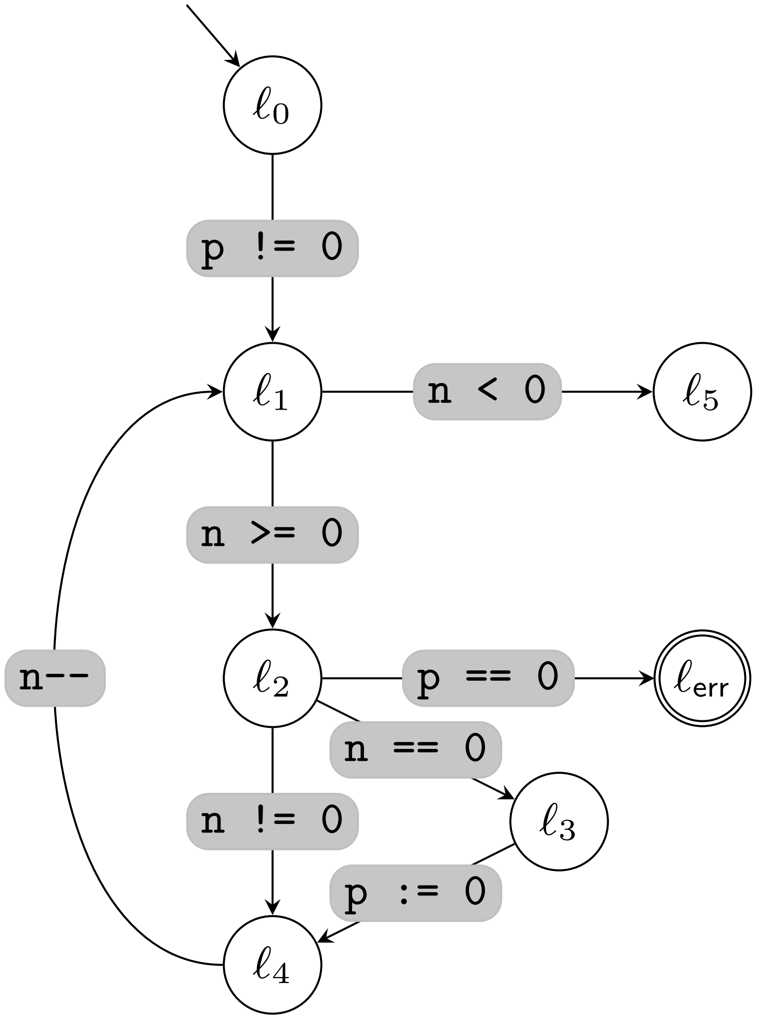


accepts all traces with the *same* unsatisfiability proof



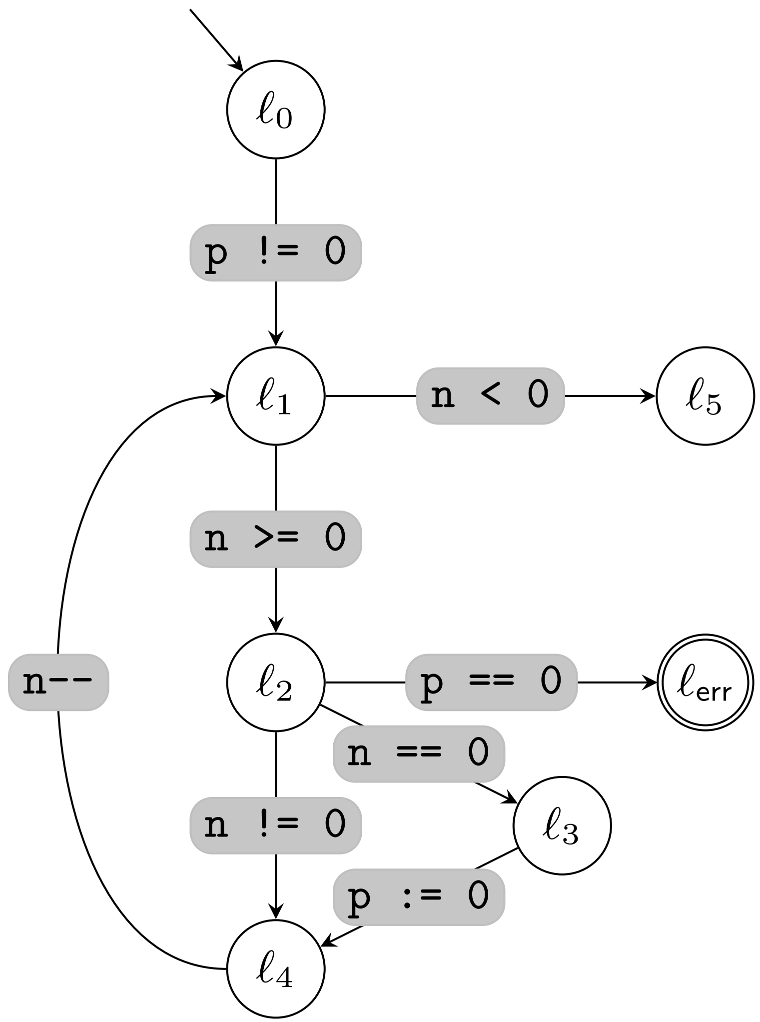
does a proof exist for every trace ?


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



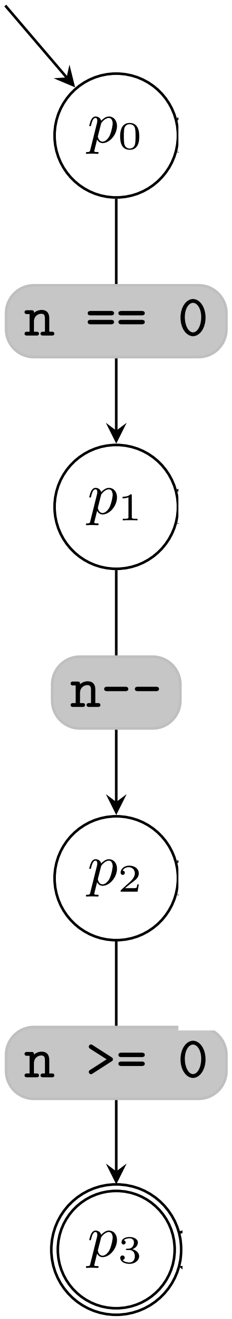
new trace:

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

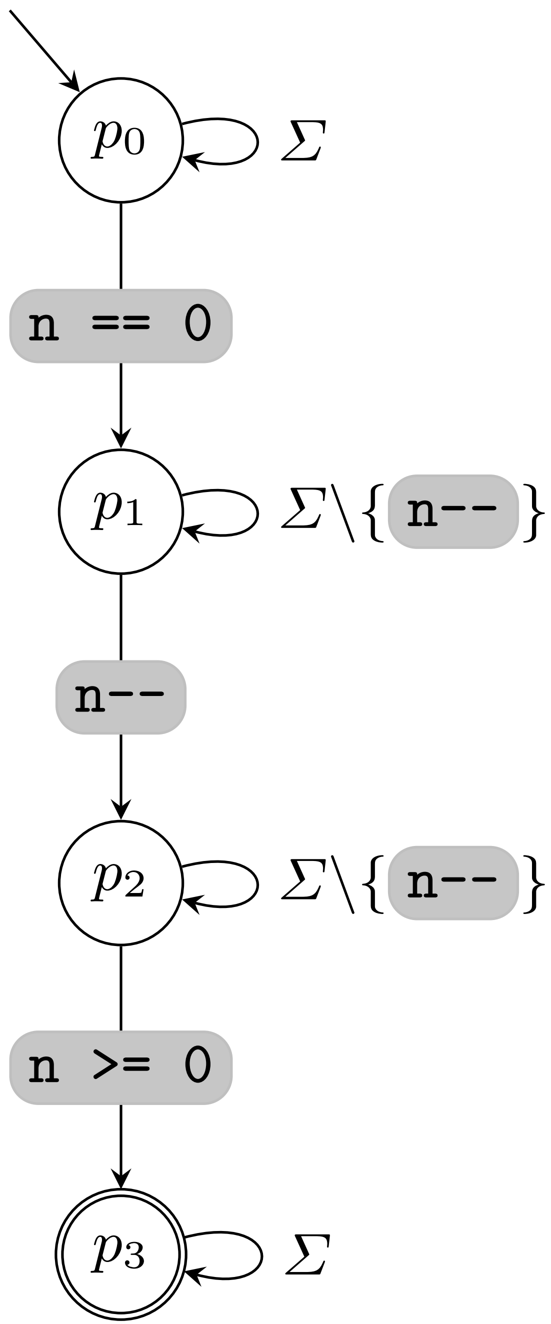


(p != 0)	
(n >= 0)	
(n == 0)	(n == 0)
(p := 0)	
(n--)	(n--)
(n >= 0)	(n >= 0)
(p == 0)	

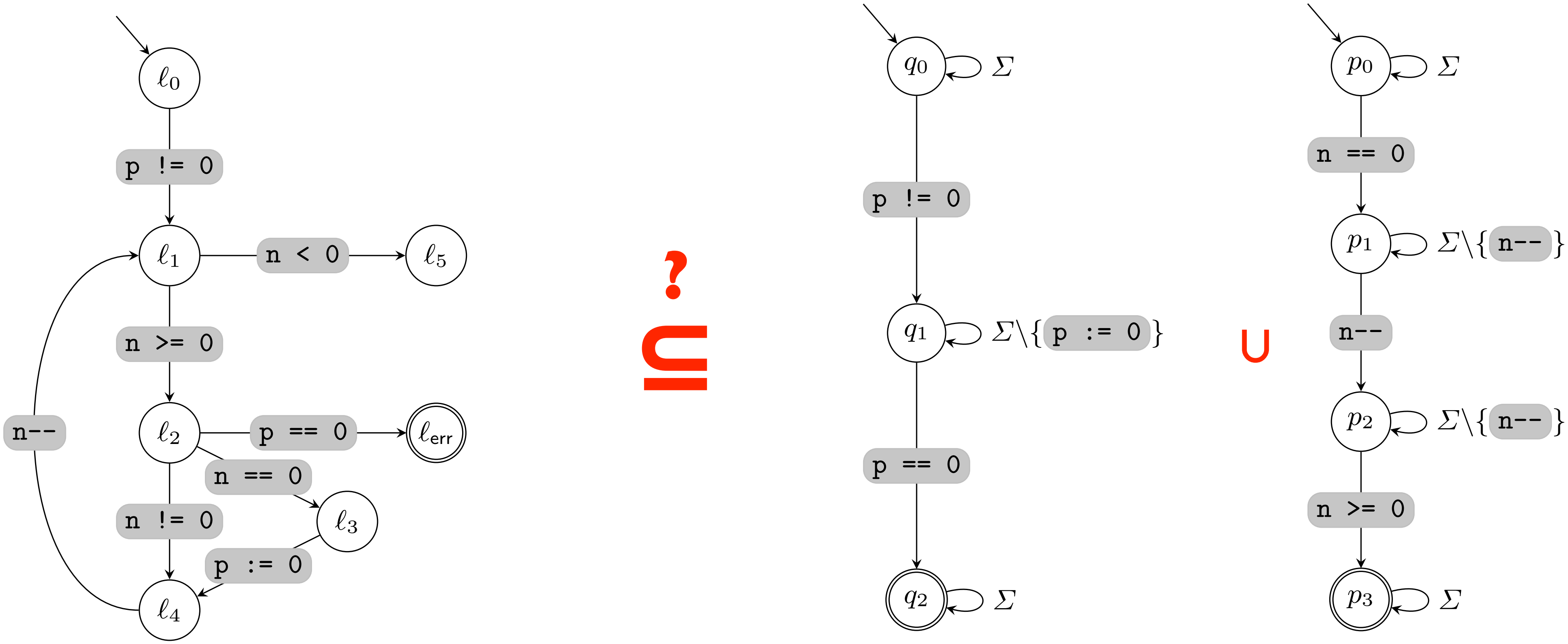
```
(n == 0)

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



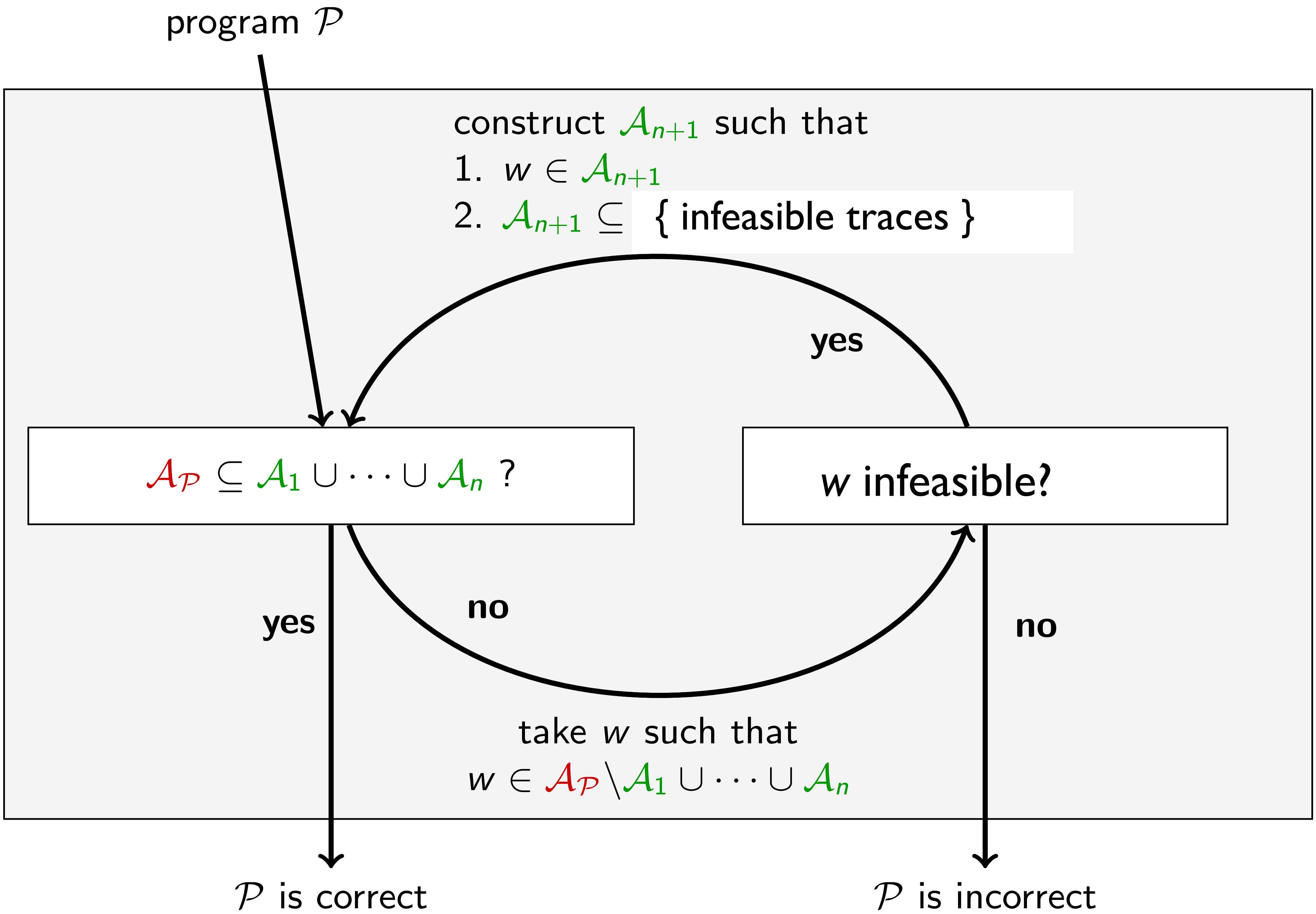

```
(n == 0)

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


p_0
Σ
n == 0
p_1
$\Sigma \setminus \{$ n-- $\}$
n--
p_2
$\Sigma \setminus \{$ n-- $\}$
n >= 0
p_3
Σ

```
(n == 0)

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



does a proof exist for every 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

automata constructed from unsatisfiable core

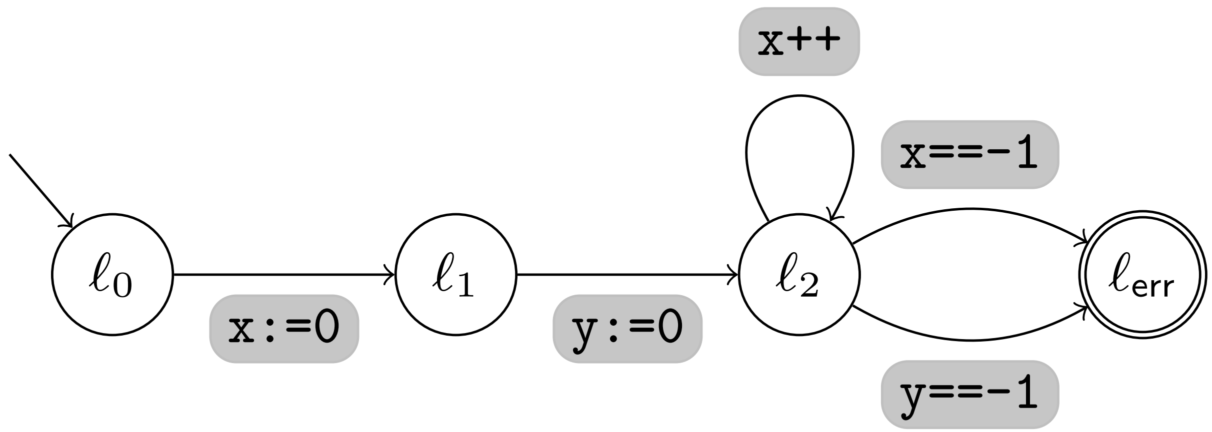
are not sufficient in general

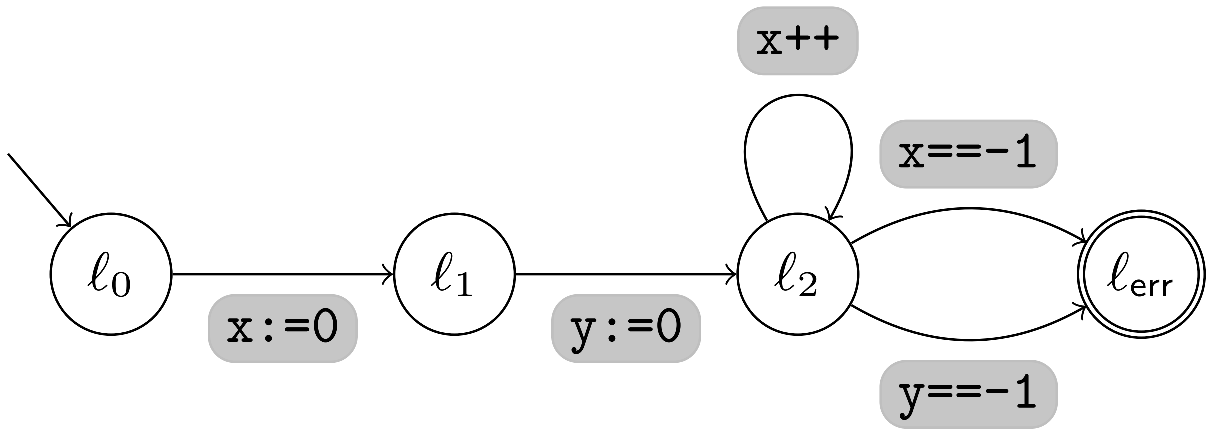
(verification algorithm not complete)

proof spaces

- new paradigm for automatic verification
- automata
- Marc Segelken: ω-Cegar [CAV 2007]
- verification for networked traffic control systems

ℓ_0: `x := 0;`
ℓ_1: `y := 0;`
ℓ_2: `while(nondet) {x++;}`
`assert(x != -1);`
`assert(y != -1);`






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

Hoare triples proving infeasibility :

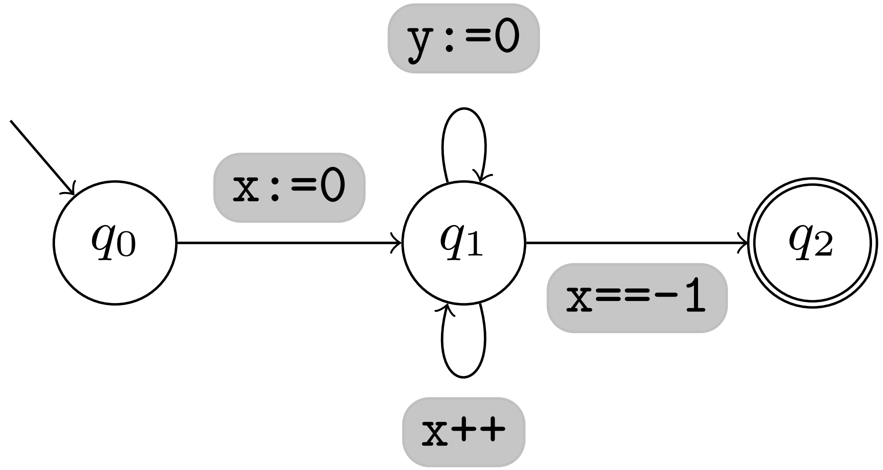
$$\{\ true\ \}\ \texttt{x:=0}\ \{x \geq 0\}$$
$$\{x \geq 0\}\ \texttt{y:=0}\ \{x \geq 0\}$$
$$\{x \geq 0\}\ \texttt{x++}\ \{x \geq 0\}$$
$$\{x \geq 0\}\ \texttt{x==-1}\ \{\ false\ \}$$

infeasibility ⇔ pre/postcondition pair *(true, false)*

Hoare triples $\longmapsto$ automaton

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

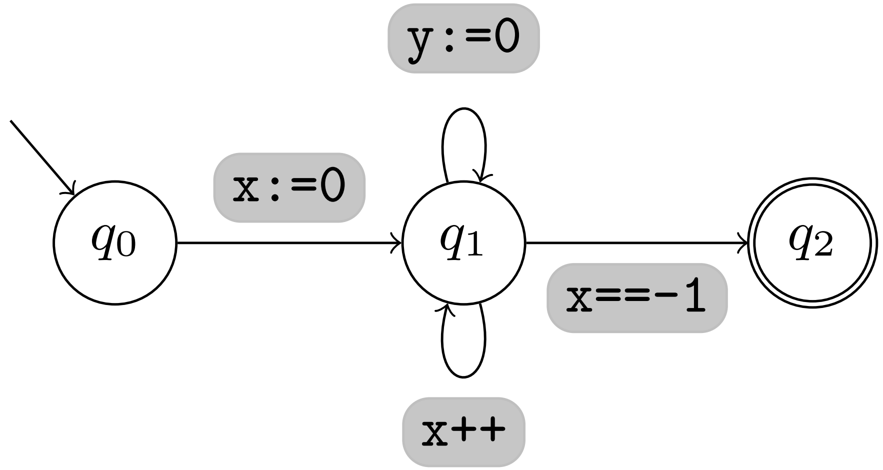
$\longmapsto$



Hoare triples $\longmapsto$ automaton

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

$\longmapsto$



sequencing of Hoare triples $\longmapsto$ run of automaton

inference rule for sequencing

$\{p\}$ **s** $\{q'\}$

$\{q'\}$ **s'** $\{q\}$

$\{p\}$ **s ; s'** $\{q\}$

proof space

infinite space of Hoare triples "{*pre*} *trace* {*post*}"

closed under inference rule of sequencing

generated from finite basis of Hoare triples "{*pre*} *stmt* {*post*}" proof of sample trace:

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

finite basis of Hoare triples "{*pre*} *stmt* {*post*}"

can be obtained from proofs of sample traces

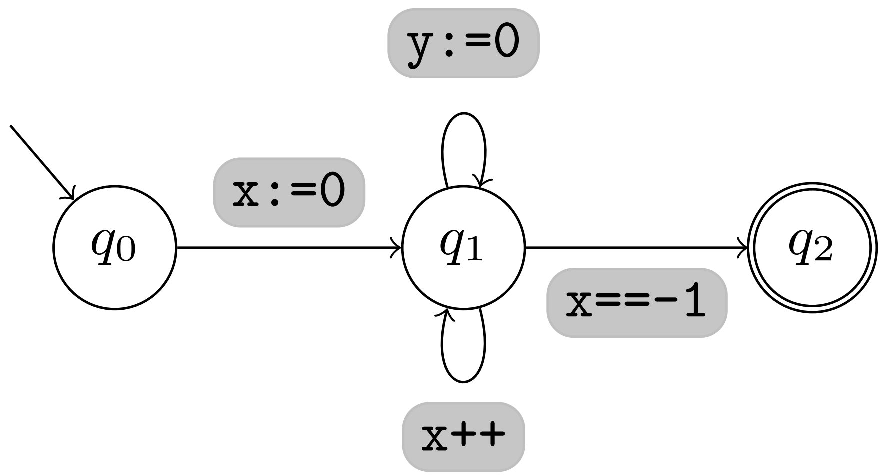
proof space

infinite space of Hoare triples "{*pre*} *trace* {*post*}"

closed under inference rule of sequencing finite basis of Hoare triples "{*pre*} *stmt* {*post*}" $\longmapsto$ automaton

$\{\ true\ \}$ `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` $\{\ false\ \}$

$\longmapsto$



sequencing of Hoare triples in basis $\longmapsto$ run of automaton

proof space contains "{*true*} *trace* {*false*}"

if

exists sequencing of Hoare triples in basis

if

exists accepting run of automaton

proof space

infinite space of Hoare triples "{*pre*} *trace* {*post*}"

closed under inference rule of sequencing

generated from finite basis of Hoare triples "{*pre*} *stmt* {*post*}" paradigm:

- construct proof space
- check proof space

simplify task for program verification:

Don’t give a proof.

Show that a proof *exists*.

automata:
existence of accepting run

inclusion check:
show that, for every word in the given set,
an accepting run *exists*

simplify task for program verification:

Show that,
for every program execution,
a proof exists.