Improved Ackermannian lower bound for the Petri nets reachability problem

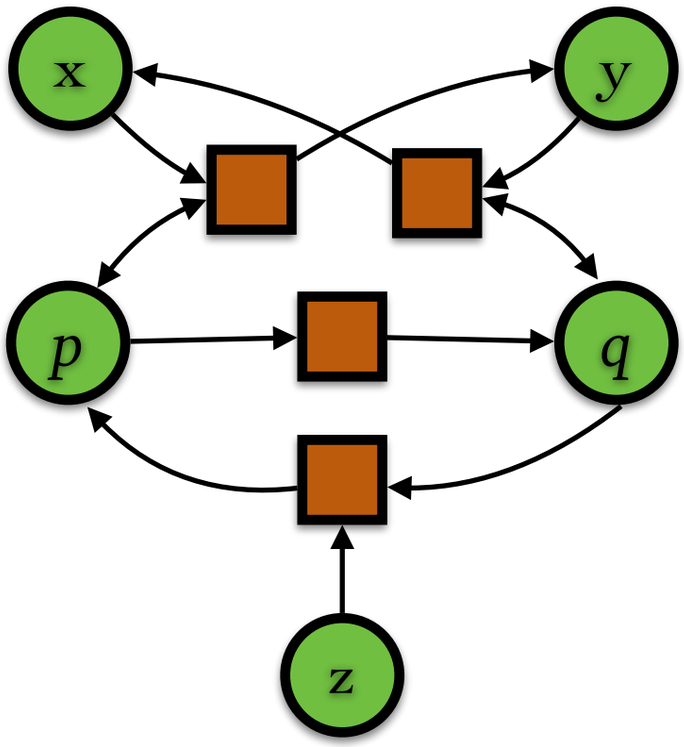
[Czerwiński, L., Lazic, Leroux, Mazowiecki 2019]
[Czerwiński, Orlikowski 2021]
[Leroux 2021]
[L. 2022]

Sławomir Lasota
University of Warsaw

Concurrency theory, 2022.04

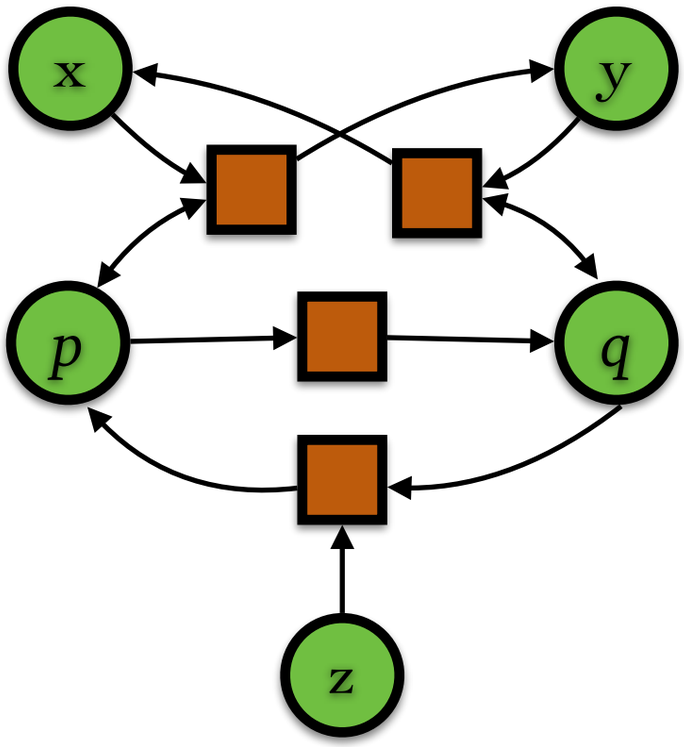
Many faces of Petri nets

- Petri nets:

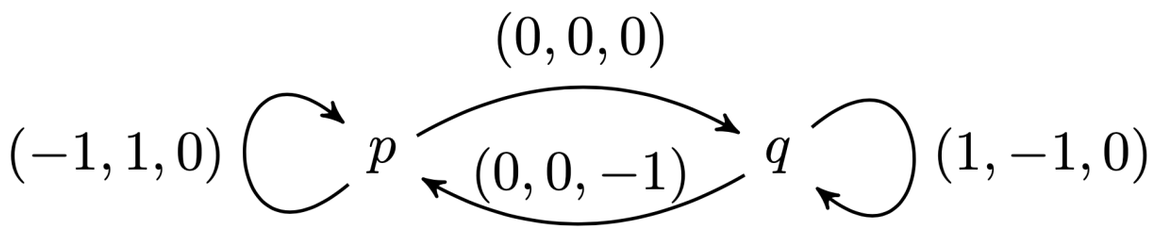


Many faces of Petri nets

- Petri nets:

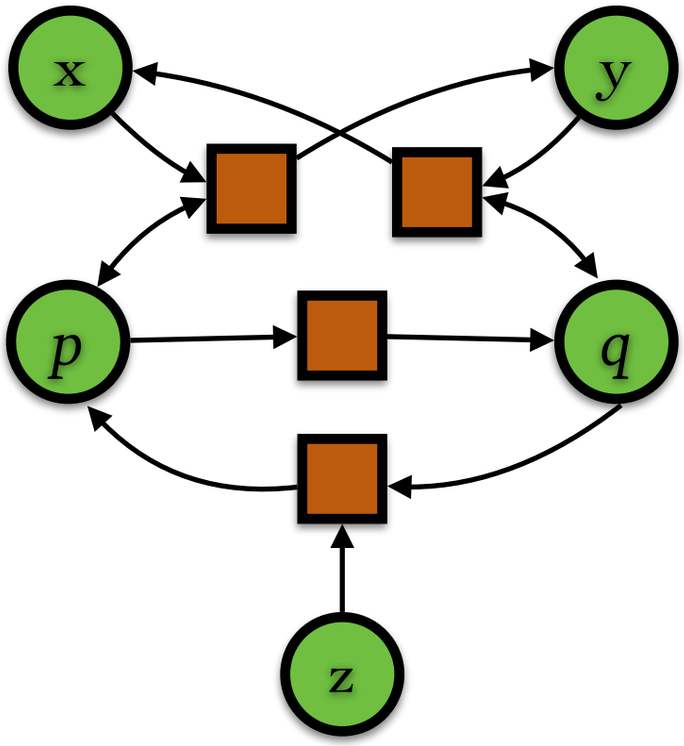


- vector addition systems with states:

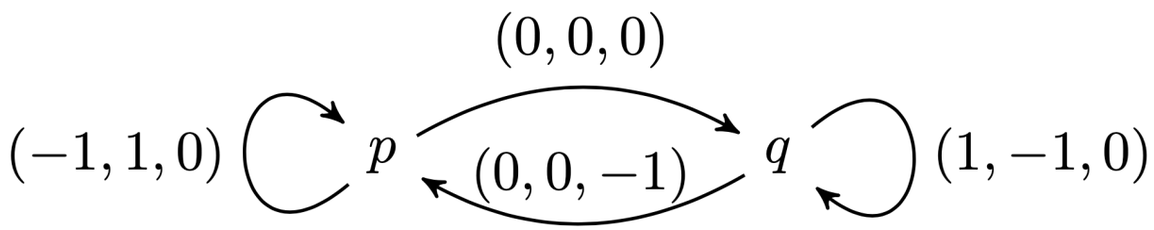


Many faces of Petri nets

- Petri nets:



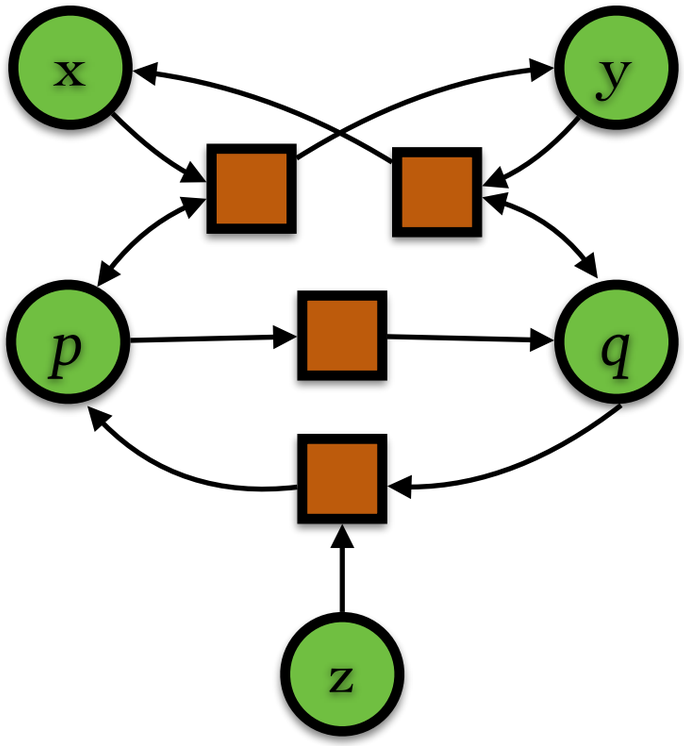
- vector addition systems with states:



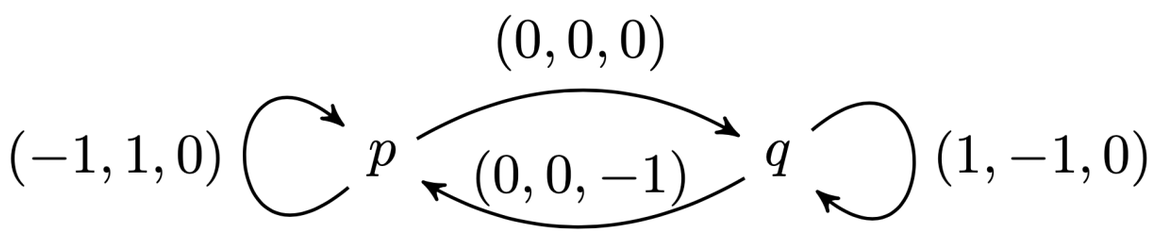
- vector addition systems
- counter automata **without 0-tests**
- multiset rewriting
- …

Many faces of Petri nets

- Petri nets:



- vector addition systems with states:



- counter programs **without 0-tests**:

```
1: loop
2:     loop
3:         x −= 1    y += 1
4:     loop
5:         x += 1    y −= 1
6:     z −= 1
```

- vector addition systems
- counter automata **without 0-tests**
- multiset rewriting
- …

Many faces of Petri nets

- counter programs **without 0-tests**:

```
1: loop
2:     loop
3:         x −= 1    y += 1
4:     loop
5:         x += 1    y −= 1
6:     z −= 1
```

Counter programs without zero tests

counters are nonnegative integer variables initially all equal zero

Counter programs without zero tests

counters are nonnegative integer variables initially all equal zero

Counter program = a sequence of commands of the form:

x $+= n$	(increment counter x by n)
x $-= n$	(decrement counter x by n)
goto L **or** L'	(jump to either line L or line L')

Counter programs without zero tests

counters are nonnegative integer variables initially all equal zero

Counter program = a sequence of commands of the form:

x $+= n$	(increment counter x by n)	
x $-= n$	(decrement counter x by n)	abort if x < n
goto L **or** L'	(jump to either line L or line L')	

Counter programs without zero tests

counters are nonnegative integer variables initially all equal zero

Counter program = a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= n$	(increment counter x by n)	
$\mathsf{x} \mathrel{-}= n$	(decrement counter x by n)	abort if x < n
goto L **or** L'	(jump to either line L or line L')	

except for the very last command which is of the form:

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)	otherwise abort

Counter programs without zero tests

counters are nonnegative integer variables initially all equal zero

Counter program = a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= n$	(increment counter x by n)	
$\mathsf{x} \mathrel{-}= n$	(decrement counter x by n)	abort if x < n
goto L **or** L'	(jump to either line L or line L')	

except for the very last command which is of the form:

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)	otherwise abort

Example:

initially: x' = x = y = 0

```
1: x′ += 100
2: goto 5 or 3
3: x += 1    x′ −= 1    y += 2
4: goto 2
5: halt if x′ = 0.
```

Counter programs without zero tests

counters are nonnegative integer variables initially all equal zero

Counter program = a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= n$	(increment counter x by n)	
$\mathsf{x} \mathrel{-}= n$	(decrement counter x by n)	abort if x < n
goto L **or** L'	(jump to either line L or line L')	

except for the very last command which is of the form:

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)	otherwise abort

Example:

initially: x' = x = y = 0

```
1: x′ += 100
2: goto 5 or 3
3: x += 1    x′ −= 1    y += 2
4: goto 2
5: halt if x′ = 0.
```

```
1: x′ += 100
2: loop
3:     x += 1    x′ −= 1    y += 2
4: halt if x′ = 0.
```

Counter programs without zero tests

counters are nonnegative integer variables initially all equal zero

Counter program = a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= n$	(increment counter x by n)	
$\mathsf{x} \mathrel{-}= n$	(decrement counter x by n)	abort if x < n
goto L **or** L'	(jump to either line L or line L')	

except for the very last command which is of the form:

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)	otherwise abort

Example:

1: $\mathsf{x}' \mathrel{+}= 100$
2: **goto** 5 **or** 3
3: $\mathsf{x} \mathrel{+}= 1$ $\quad \mathsf{x}' \mathrel{-}= 1$ $\quad \mathsf{y} \mathrel{+}= 2$
4: **goto** 2
5: **halt if** $\mathsf{x}' = 0$.

initially: x' = x = y = 0

finally: x' = 0 x = 100 y = 200

Counter programs without zero tests

counters are nonnegative integer variables initially all equal zero

Counter program = a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= n$	(increment counter x by n)
$\mathsf{x} \mathrel{-}= n$	(decrement counter x by n)
goto L **or** L'	(jump to either line L or line L')

abort if x < n

except for the very last command which is of the form:

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)

otherwise abort

Example:

1: $\mathsf{x}' \mathrel{+}= 100$
2: **goto** 5 **or** 3
3: $\mathsf{x} \mathrel{+}= 1$ $\quad \mathsf{x}' \mathrel{-}= 1$ $\quad \mathsf{y} \mathrel{+}= 2$
4: **goto** 2
5: **halt if** $\mathsf{x}' = 0$.

no zero tests

initially: x' = x = y = 0

finally: x' = 0 x = 100 y = 200

Reachability problem

Reachability problem: given a counter program **without zero tests**

```
1: x′ += 100
2: goto 5 or 3
3: x += 1    x′ −= 1    y += 2
4: goto 2
5: halt if x′ = 0.
```

can it halt? (successfully execute its halt command)

Reachability problem

Reachability problem: given a counter program **without zero tests**

```
1: x′ += 100
2: goto 5 or 3
3: x += 1    x′ −= 1    y += 2
4: goto 2
5: halt if x′ = 0.
```

can it halt? (successfully execute its halt command)

Coverability problem: given a counter program **without zero tests**
with trivial halt command

```
1: x′ += 100
2: goto 5 or 3
3: x += 1    x′ −= 1    y += 2
4: goto 2
5: halt.
```

can it halt?

1969 — decidability of coverability **[Karp, Miller]**

1970

1976 — EXPSPACE lower bound **[Lipton]**

1977 — (incomplete) decidability of reachability **[Sacerdote, Tenney]**

1978 — EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 — decidability of reachability **[Mayr]**

1982 — decidability of reachability - simplified proof **[Kosaraju]**

1990

1992 — decidability of reachability - refined data structure **[Lambert]**

2000

huge complexity gap!

2009 — decidability of reachability by Presburger invariants **[Leroux]**

2010

2011 — decidability of reachability without KLMST decomposition **[Leroux]**

2015 — first upper bound F_{ω^3} **[Leroux, Schmitz]**

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]**

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]**

2020

} **gap**

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]**

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]** } **gap**

2020

$$\left.2^{2^{2^{\cdots^2}}}\right\} n$$

2021

2022

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]**

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]** } **gap**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — "super"-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

2022

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]**

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]** } **gap**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — "super"-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

2021 — Ackermannian lower bound **[Czerwiński, Orlikowski] [Leroux]** **gap closed!**

2022

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]**

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]** } **gap**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — "super"-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

2021 — Ackermannian lower bound **[Czerwiński, Orlikowski] [Leroux]** **gap closed!**

2021 — improved Ackermannian lower bound **[L.]**

2022

Fast growing functions and induced complexity classes

$$A_1(n) = 2n$$

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ldots \circ A_i}_{n}(1) = A_i^n(1)$$

Fast growing functions and induced complexity classes

$$A_1(n) = 2n$$

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ \ldots \circ A_i}_{n}(1) = A_i^n(1)$$

$$A_1(n) = 2n$$

$$A_2(n) = 2^n$$

$$A_3(n) = 2^{2^{\cdots^2}} \Big\}n$$

$$A_4(n) = \ldots$$

Fast growing functions and induced complexity classes

$$A_1(n) = 2n$$

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ \ldots \circ A_i}_{n}(1) = A_i^n(1)$$

$$A_1(n) = 2n$$

$$A_2(n) = 2^n$$

$$A_3(n) = 2^{2^{\cdots^2}} \Big\}n$$

$$A_4(n) = \ldots$$

$$\mathsf{F}_i = \bigcup_{p \in \mathsf{FF}_{i-1}} \mathrm{DTIME}(A_i(p(n)))$$

Fast growing functions and induced complexity classes

$$A_1(n) = 2n$$

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ \ldots \circ A_i}_{n}(1) = A_i^n(1)$$

$$A_1(n) = 2n$$

$$A_2(n) = 2^n$$

$$A_3(n) = 2^{2^{\cdots^2}} \Big\}n$$

$$A_4(n) = \ldots$$

$$\mathsf{F}_i = \bigcup_{p \in \mathsf{FF}_{i-1}} \mathrm{DTIME}(A_i(p(n)))$$

$$\mathsf{FF}_i = \bigcup_{m} \mathrm{FDTIME}(A_i^m(n))$$

dimension = number of counters

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]**

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — super-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

2021 — Ackermannian lower bound **[Czerwiński, Orlikowski]** **[Leroux]**

2021 — improved Ackermannian lower bound **[L.]**

2022

dimension = number of counters

F_n -membership in dimension:

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]** *n-4*

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — super-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

2021 — Ackermannian lower bound **[Czerwiński, Orlikowski] [Leroux]**

2021 — improved Ackermannian lower bound **[L.]**

2022

dimension = number of counters

F_n -membership in dimension:

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]** *n-4*

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — super-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

F_n -hardness in dimension:

2021 — Ackermannian lower bound **[Czerwiński, Orlikowski]** **[Leroux]**

2021 — improved Ackermannian lower bound **[L.]**

2022

dimension = number of counters

F_n -membership in dimension:

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]** *n-4*

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — super-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

F_n -hardness in dimension:

2021 — Ackermannian lower bound **[Czerwiński, Orlikowski]** **[Leroux]** *6n*

2021 — improved Ackermannian lower bound **[L.]**

2022

dimension = number of counters

F_n -membership in dimension:

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]** *n-4*

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — super-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

F_n -hardness in dimension:

2021 — Ackermannian lower bound **[Czerwiński, Orlikowski]** **[Leroux]** *6n* *4n+5*

2021 — improved Ackermannian lower bound **[L.]**

2022

dimension = number of counters

F_n -membership in dimension:

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]** *n-4*

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — super-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

F_n -hardness in dimension:

2021	Ackermannian lower bound **[Czerwiński, Orlikowski] [Leroux]**	*6n*	*4n+5*
2021	improved Ackermannian lower bound **[L.]**	***3n+2***	

2022

dimension = number of counters

F_n -membership in dimension:

2019 — Ackermannian upper bound F_ω **[Leroux, Schmitz]** *n-4*

2019 — TOWER lower bound F_3 **[Czerwiński, L., Lazic, Leroux, Mazowiecki]**

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

2021

2021 — super-TOWER lower bound F_3 **[Czerwiński, L., Orlikowski]**

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

F_n -hardness in dimension:

2021	Ackermannian lower bound **[Czerwiński, Orlikowski]** **[Leroux]**	*6n*	*4n+5*
2021	improved Ackermannian lower bound **[L.]**	***3n+2***	
2021	further improved Ackermannian lower bound **[Leroux]**	*2n+4*	

2022

dimension = number of counters

F_n -membership in dimension:

2019 — Ackermannian upper bound F_ω [Leroux, Schmitz]

2019 — TOWER lower bound F_3 [Czerwiński, L., Lazic, Leroux, Mazowiecki]

2020

$$2^{2^{2^{\cdots^{2}}}\Big\}n}$$

n-4

still a gap!

2021 — super-TOWER lower bound F_3 [Czerwiński, L., Orlikowski]

$$2^{2^{2^{\cdots^{2}}}\Big\}2^n}$$

F_n -hardness in dimension:

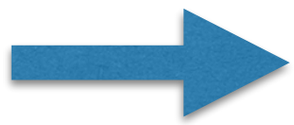
2021	Ackermannian lower bound [Czerwiński, Orlikowski] [Leroux]	*6n*	*4n+5*
2021	improved Ackermannian lower bound [L.]	***3n+2***	
2021	further improved Ackermannian lower bound [Leroux]	*2n+4*	

2022

k fixed

sketch of the proof of $\mathbf{F}_k$-hardness in dimension $3k+2$

does a counter program
with 0-tests of size n
have a halting run that
does $A_k(n)/2$ zero tests?



does a counter program
with $3k+2$ counters
without 0-tests halt?

The set computed by a counter program

initial valuation: all counters 0

```
1: x += 1    y += 1
2: loop
3:     x += 1    y += 1
4: for i := n down to 1 do
5:     loop
6:         x −= 1    z += 1
7:     loop
8:         x += i + 1    z −= i
9: loop
10:    x −= n + 1    y −= 1
11: halt if y = 0.
```

consider all runs
(nondeterminism)

the set of all valuations at successful halt

B-multiplier

B - fixed positive integer

initial valuation: all counters 0

```
1: x += 1    y += 1
2: loop
3:     x += 1    y += 1
4: for i := n down to 1 do
5:     loop
6:         x −= 1    z += 1
7:     loop
8:         x += i + 1    z −= i
9: loop
10:    x −= n + 1    y −= 1
11: halt if y = 0.
```

3 distinguished counters b, c, d

consider all runs (nondeterminism)

B-multiplier

B - fixed positive integer

initial valuation: all counters 0

```
1: x += 1    y += 1
2: loop
3:     x += 1    y += 1
4: for i := n down to 1 do
5:     loop
6:         x −= 1    z += 1
7:     loop
8:         x += i + 1    z −= i
9: loop
10:    x −= n + 1    y −= 1
11: halt if y = 0.
```

3 distinguished counters b, c, d

consider all runs (nondeterminism)

- b = B
- c > 0
- d = b · c
- all other counters 0

} RATIO(b, c, d, B)

B-multiplier

B - fixed positive integer

initial valuation: all counters 0

```
1: x += 1    y += 1
2: loop
3:     x += 1    y += 1
4: for i := n down to 1 do
5:     loop
6:         x −= 1    z += 1
7:     loop
8:         x += i + 1    z −= i
9: loop
10:    x −= n + 1    y −= 1
11: halt if y = 0.
```

3 distinguished counters b, c, d

consider all runs (nondeterminism)

- ~~b = B~~
- b > 0 ?
- c > 0
- d = b · c
- all other counters 0

} RATIO(b, c, d, B)

B-multiplier

B - fixed positive integer

initial valuation: all counters 0

```
1: x += 1    y += 1
2: loop
3:     x += 1    y += 1
4: for  i  :=  n  down to  1 do
5:     loop
6:         x −= 1    z += 1
7:     loop
8:         x += i + 1    z −= i
9: loop
10:    x −= n + 1    y −= 1
11: halt if y = 0.
```

3 distinguished counters b, c, d

consider all runs (nondeterminism)

- ~~b = B~~ • b > 0 ?
- c > 0
- d = b · c
- all other counters 0

} RATIO(b, c, d, B)

10th Hilbert's problem!

B-multiplier

B - fixed positive integer

initial valuation: all counters 0

```
1: b += B    d += B    c += 1
2: loop
3:     d += B    c += 1
11: halt if y = 0.
```

- b = B
- c > 0
- d = b · c
- all other counters 0

} RATIO(b, c, d, B)

B-multiplier

B - fixed positive integer

initial valuation: all counters 0

```
1: b += B    d += B    c += 1
2: loop
3:     d += B    c += 1
11: halt if y = 0.
```

- b = B
- c > 0
- d = b · c
- all other counters 0

} RATIO(b, c, d, B)

One can compute an **$A_k(n)$-multiplier** with $3k+2$ counters, in time polynomial in k, n

F_k-hardness in dimension $3k+2$

program of size n
with two **0-tested** counters:

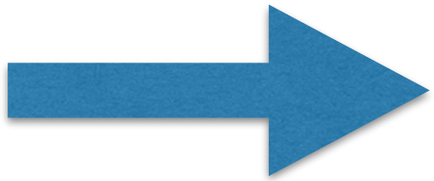
```
 1: i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
 2: loop
 3:     x += 1   y += 1   c += 1   d += 1
 4: loop
 5:     loop
 6:         c −= i   c′ +=
 7:         loop at most    time
 8:             x −= i                x′ += i + 1
 9:     loop
10:         b −= 1   b    = i + 1
11:     loop
12:         b′ −= 1   b += 1
13:     loop
14:         c′ −= 1   c += 1
15:         loop at most b times
16:             x′ −= 1   x += 1   d += 1
18: zero? i
20:         x −= i   y −= 1
21: halt if y = 0
```

P

does it have a halting run
that does $A_k(n)/2$ zero tests?

F_k-hardness in dimension $3k+2$

program of size n
with two **0-tested** counters:



program **without** 0-tests:

```
1: i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
2: loop
3:     x += 1   y += 1   c += 1   d += 1
4: loop
5:     loop
6:         c −= i   c′ +=
7:         loop at most    time
8:             x −= i                x += i + 1
9:     loop
10:        b −= 1   b    = i + 1
11:    loop
12:        b′ −= 1   b += 1
13:    loop
14:        c′ −= 1   c += 1
15:        loop at most b times
16:            x′ −= 1   x += 1   d += 1
18: zero? i
20:        x −= i   y −= 1
21: halt if y = 0
```

P

does it have a halting run
that does $A_k(n)/2$ zero tests?

does it halt?

F_k-hardness in dimension $3k+2$

program of size n
with two **0-tested** counters:

```
1: i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
2: loop
3:     x += 1   y += 1   c += 1   d += 1
4: loop
5:     loop
6:         c −= i   c′ +=
7:         loop at most    times
8:             x −= i          x += i+1
9:     loop
10:        b −= 1   b      = i+1
11:    loop
12:        b′ −= 1   b += 1
13:    loop
14:        c′ −= 1   c += 1
15:        loop at most b times
16:            x′ −= 1   x += 1   d += 1
17:    i += 1
18: zero? i
19: loop
20:        x −= i   y −= 1
21: halt if y = 0
```

P

does it have a halting run
that does $A_k(n)/2$ zero tests?

program **without** 0-tests:

```
1: x += 1   y += 1
2: loop
3:     x += 1   y += 1
4: for i := n down to 1 do
5:     loop
6:         x −= 1   z += 1
7:     loop
8:         x += i+1   z −= i
9: loop
10:    x −= n+1   y −= 1
11: halt if y = 0.
```

$A_k(n)$-multiplier M

RATIO(b, c, d, $A_k(n)$)

```
1: i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
2: loop
3:     x += 1   y += 1   c += 1   d += 1
4: loop
5:     loop
6:         c −= i   c′ +=
7:         loop at most    times
8:             x −= i   d      i          x += i+1
9:     loop
10:        b −= 1   b′      i+1
11:    loop
12:        b′ −= 1   b += 1
13:    loop
14:        c′ −= 1   c += 1
15:        loop at most b times
16:            x′ −= 1   x += 1   d += 1
17:    i += 1
18: zero? i
19: loop
20:    x −= i   y −= 1
21: halt if y = 0
```

P

instrumented

does it halt?

Instrumentation

- $b = A_k(n)$
- $c > 0$
- $d = b \cdot c$
- $x = y = 0$ **0-tested** counters

```
1:  i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
2:  loop
3:      x += 1   y += 1   c += 1   d += 1
4:  loop
5:      loop
6:          c −= i   c′ +=
7:          loop at most   tim
8:              x −= i   c   =   x′ += i + 1
9:      loop
10:         b −= 1   b   = i + 1
11:     loop
12:         b′ −= 1   b += 1
13:     loop
14:         c′ −= 1   c += 1
15:         loop at most b times
16:             x′ −= 1   x += 1   d += 1
17:     i += 1
18: zero? î
19: loop
20:     x −= i   y −= 1
21: halt if y = 0
```

P

instrumented

Aim:
simulate $A_k(n)/2$ zero tests

Instrumentation

- $b = A_k(n)$
- $c > 0$
- $d = b \cdot c$
- $x = y = 0$ **0-tested** counters

```
1:  i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
2:  loop
3:      x += 1   y += 1   c += 1   d += 1
4:  loop
5:      loop
6:          c −= i   c′ +=
7:          loop at most
8:              x −= i   c      x′ += i + 1
9:      loop
10:         b −= 1   b      i + 1
11:     loop
12:         b′ −= 1   b += 1
13:     loop
14:         c′ −= 1   c += 1
15:         loop at most b times
16:             x′ −= 1   x += 1   d += 1
17:     i += 1
18: zero? i
19: loop
20:     x −= i   y −= 1
21: halt if y = 0
```

P

instrumented

- **instrument increments and decrements:**

command	replaced by	
x += 1	x += 1	c −= 1
x −= 1	x −= 1	c += 1

put x, y on **budget** c

c + x + y constans

Aim:
simulate $A_k(n)/2$ zero tests

Instrumentation

- $b = A_k(n)$
- $c > 0$
- $d = b \cdot c$
- $x = y = 0$ **0-tested** counters

```
1:  i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
2:  loop
3:      x += 1   y += 1   c += 1   d += 1
4:  loop
5:      loop
6:          c −= i   c′ +=
7:          loop at most
8:              x −= i   d
9:      loop
10:         b −= 1   b
11:     loop
12:         b′ −= 1   b += 1
13:     loop
14:         c′ −= 1   c += 1
15:         loop at most b times
16:             x′ −= 1   x += 1   d += 1
17:     i += 1
18: zero? i
19: loop
20:     x −= i   y −= 1
21: halt if y = 0
```

P

instrumented

- **instrument increments and decrements:**

command	replaced by
x += 1	x += 1 c −= 1
x −= 1	x −= 1 c += 1

put x, y on **budget** c

- **replace zero? x by**

c + x + y constans

ZERO? x:

```
1: loop
2:     y −= 1   x += 1   d −= 1
3: loop
4:     c −= 1   y += 1   d −= 1
5: loop
6:     y −= 1   c += 1   d −= 1
7: loop
8:     x −= 1   y += 1   d −= 1
9: b −= 2
```

Aim:
simulate $A_k(n)/2$ zero tests

Instrumentation

- $b = A_k(n)$
- $c > 0$
- $d = b \cdot c$
- $x = y = 0$ **0-tested** counters

```
1:  i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
2:  loop
3:      x += 1   y += 1   c += 1   d += 1
4:  loop
5:      loop
6:          c −= i   c′ +=
7:          loop at most   tim
8:              x −= i   d      x′ += i + 1
9:      loop
10:         b −= 1   b      i + 1
11:     loop
12:         b′ −= 1   b += 1
13:     loop
14:         c′ −= 1   c += 1
15:         loop at most b times
16:             x′ −= 1   x += 1   d += 1
17:     i += 1
18: zero? i
19: loop
20:     x −= i   y −= 1
21: halt if y = 0
```

P

instrumented

Aim:
simulate $A_k(n)/2$ zero tests

- **instrument increments and decrements:**

command	replaced by	
x += 1	x += 1	c −= 1
x −= 1	x −= 1	c += 1

put x, y on **budget** c

- **replace zero? x by**

c + x + y constans

ZERO? x:

```
1: loop
2:     y −= 1   x += 1   d −= 1
3: loop
4:     c −= 1   y += 1   d −= 1
5: loop
6:     y −= 1   c += 1   d −= 1
7: loop
8:     x −= 1   y += 1   d −= 1
9: b −= 2
```

- **replace halt by**

halt if . . . , d = 0.

halt of M

Simulation of a zero test

$$d = b \cdot \frac{(c + x + y)}{\text{constans}}$$

ZERO? x:

```
1: loop
2:     y −= 1    x += 1    d −= 1
3: loop
4:     c −= 1    y += 1    d −= 1
5: loop
6:     y −= 1    c += 1    d −= 1
7: loop
8:     x −= 1    y += 1    d −= 1
9: b −= 2
```

Simulation of a zero test

put x, y on **budget** c

$$d = b \cdot \frac{(c + x + y)}{\text{constans}}$$

ZERO?x:

```
1: loop
2:     y −= 1    x += 1    d −= 1
3: loop
4:     c −= 1    y += 1    d −= 1
5: loop
6:     y −= 1    c += 1    d −= 1
7: loop
8:     x −= 1    y += 1    d −= 1
9: b −= 2
```

Simulation of a zero test

put x, y on **budget** c

$$d = b \cdot \frac{(c + x + y)}{\text{constans}}$$

ZERO? x:

```
1: loop
2:     y −= 1    x += 1    d −= 1
3: loop
4:     c −= 1    y += 1    d −= 1
5: loop
6:     y −= 1    c += 1    d −= 1
7: loop
8:     x −= 1    y += 1    d −= 1
9: b −= 2
```

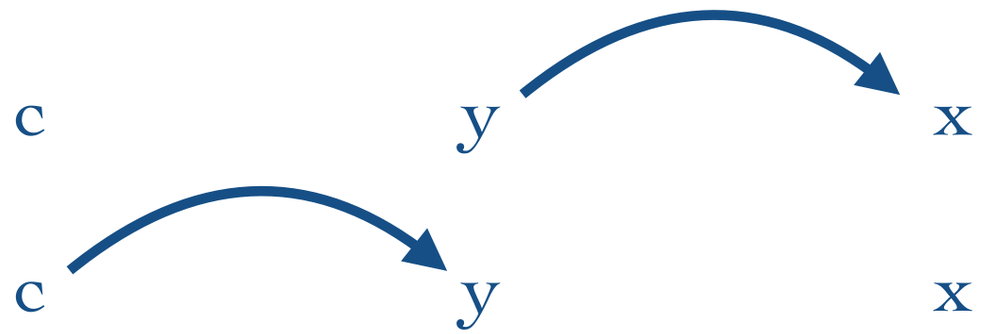
Simulation of a zero test

put x, y on **budget** c

$$d = b \cdot \frac{(c + x + y)}{\text{constans}}$$

ZERO?x:

```
1: loop
2:     y −= 1    x += 1    d −= 1
3: loop
4:     c −= 1    y += 1    d −= 1
5: loop
6:     y −= 1    c += 1    d −= 1
7: loop
8:     x −= 1    y += 1    d −= 1
9: b −= 2
```



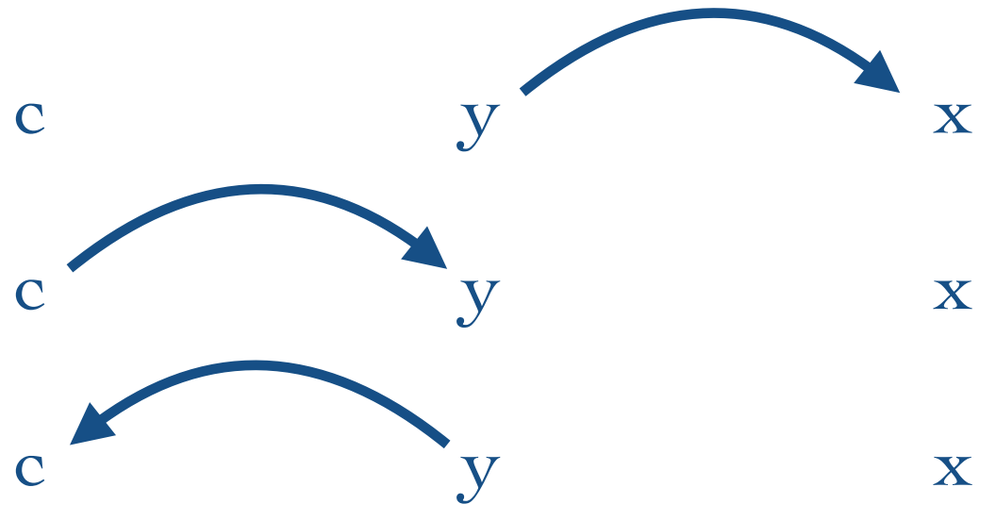
Simulation of a zero test

put x, y on **budget** c

$$d = b \cdot \frac{(c + x + y)}{\text{constans}}$$

ZERO? x:

```
1: loop
2:     y −= 1    x += 1    d −= 1
3: loop
4:     c −= 1    y += 1    d −= 1
5: loop
6:     y −= 1    c += 1    d −= 1
7: loop
8:     x −= 1    y += 1    d −= 1
9: b −= 2
```



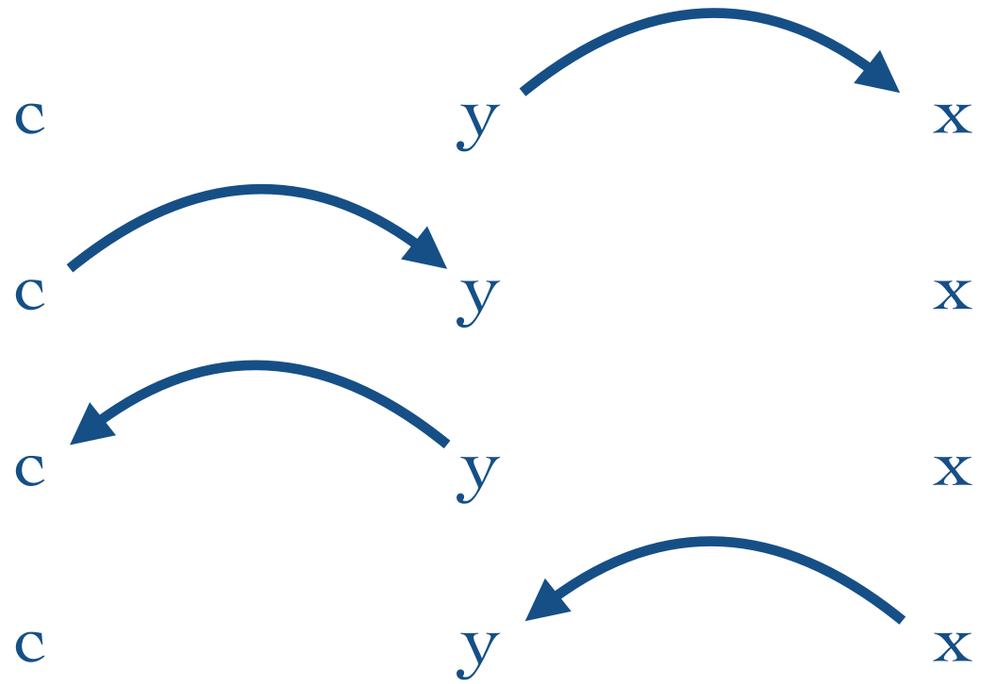
Simulation of a zero test

put x, y on **budget** c

$$d = b \cdot \frac{(c + x + y)}{\text{constans}}$$

ZERO? x:

```
1: loop
2:     y −= 1    x += 1    d −= 1
3: loop
4:     c −= 1    y += 1    d −= 1
5: loop
6:     y −= 1    c += 1    d −= 1
7: loop
8:     x −= 1    y += 1    d −= 1
9: b −= 2
```



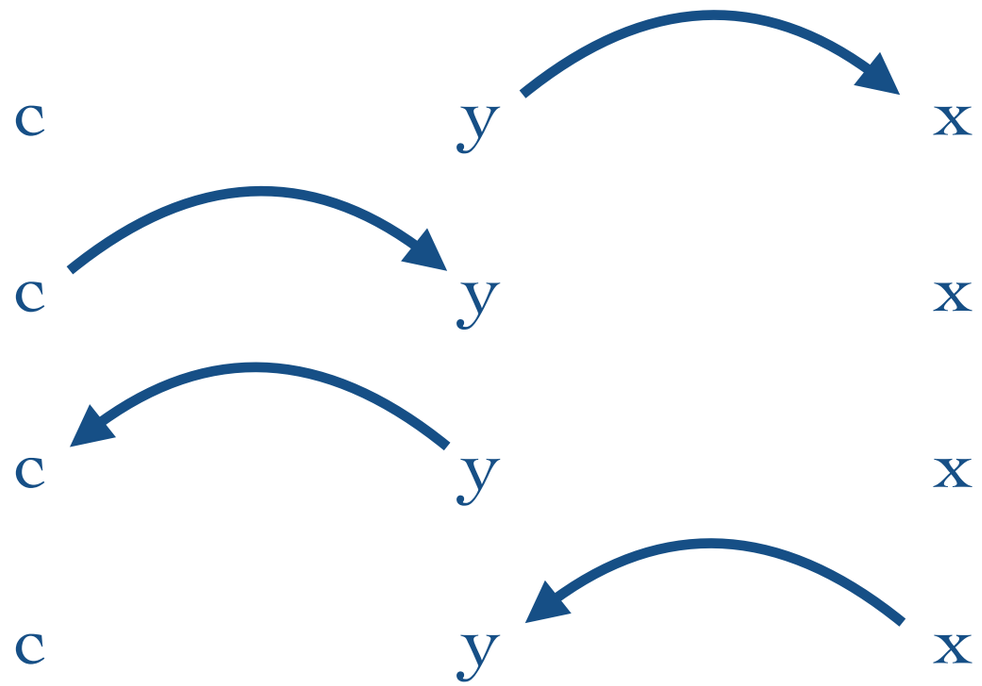
Simulation of a zero test

put x, y on **budget** c

$$d = b \cdot \frac{(c + x + y)}{\text{constans}}$$

ZERO?x:

```
1: loop
2:     y −= 1    x += 1    d −= 1
3: loop
4:     c −= 1    y += 1    d −= 1
5: loop
6:     y −= 1    c += 1    d −= 1
7: loop
8:     x −= 1    y += 1    d −= 1
9: b −= 2
```



d decreases by <= 2 · (c + x + y)
b decreases by 2

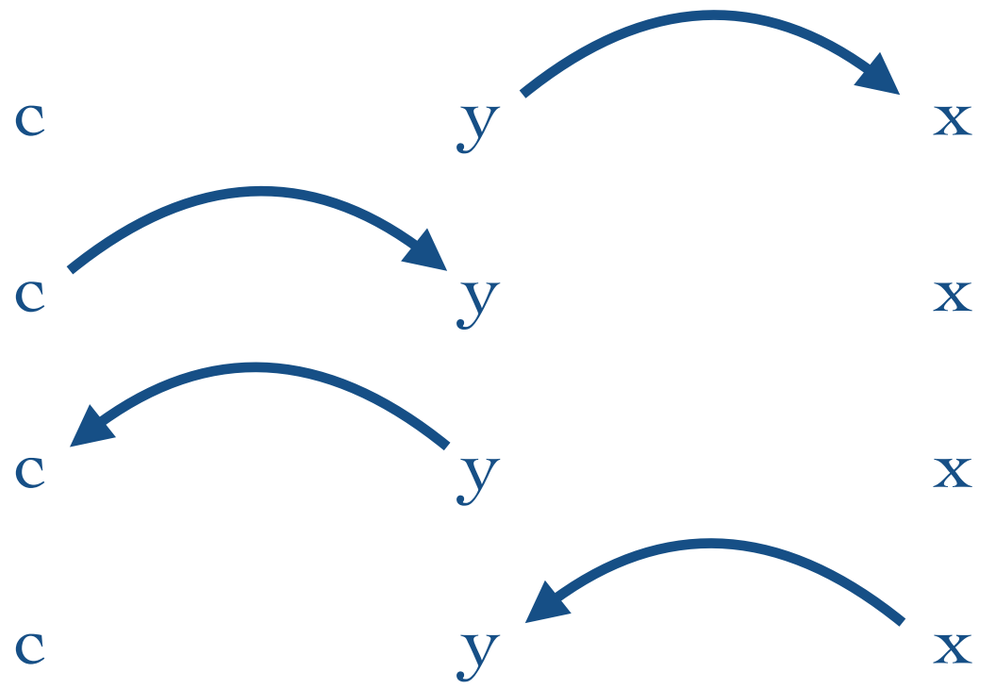
Simulation of a zero test

put x, y on **budget** c

$$d = b \cdot \frac{(c + x + y)}{\text{constans}}$$

ZERO? x:

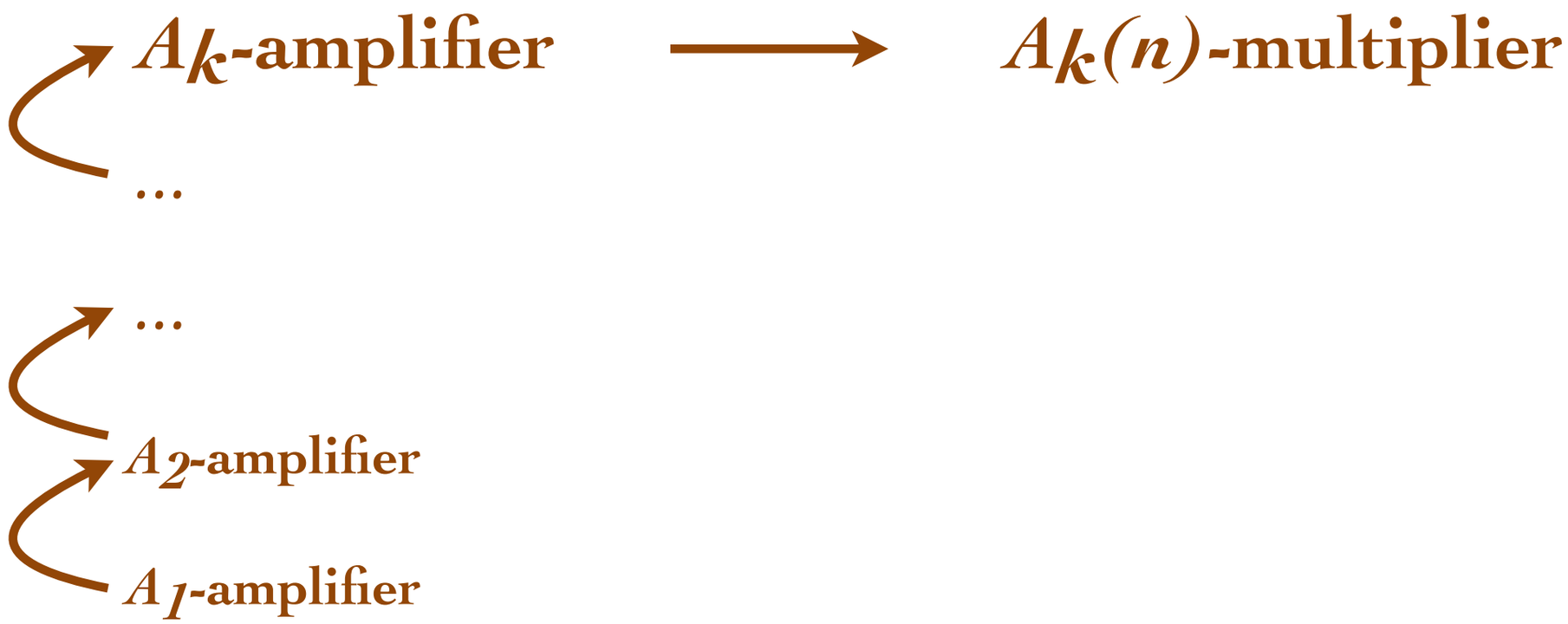
```
1: loop
2:     y −= 1    x += 1    d −= 1
3: loop
4:     c −= 1    y += 1    d −= 1
5: loop
6:     y −= 1    c += 1    d −= 1
7: loop
8:     x −= 1    y += 1    d −= 1
9: b −= 2
```



d decreases by <= 2 · (c + x + y)
b decreases by 2

- d decreases by 2 · (c + x + y) ⟶ x = 0 and y = c initially and finally
- d decreases by < 2 · (c + x + y) ⟶ **halt if** d = 0 will surely fail

One can compute an $A_k(n)$-multiplier with $3k+2$ counters, in time polynomial in k, n



The set computed by a counter program **from a set**

initial valuation: all counters 0

```
1: x += 1    y += 1
2: loop
3:     x += 1    y += 1
4: for i := n down to 1 do
5:     loop
6:         x −= 1    z += 1
7:     loop
8:         x += i + 1    z −= i
9: loop
10:    x −= n + 1    y −= 1
11: halt if y = 0.
```

consider all runs
(nondeterminism)

the set of all valuations at successful halt

The set computed by a counter program **from a set**

a set I of initial valuations

```
1: x += 1    y += 1
2: loop
3:     x += 1    y += 1
4: for i := n down to 1 do
5:     loop
6:         x −= 1    z += 1
7:     loop
8:         x += i+1    z −= i
9: loop
10:    x −= n+1    y −= 1
11: halt if y = 0.
```

consider all runs **starting in I** (nondeterminism)

the set of all valuations at successful halt

F-amplifier

RATIO(b, c, d, *B*)

- b = *B*
- c > 0
- d = b · c
- all other counters 0

For every fixed *B*:

RATIO(b, c, d, *B*)

```
1: x += 1    y += 1
2: loop
3:     x += 1    y += 1
4: for i = n down to 1 do
5:     loop
6:         x −= 1    z += 1
7:     loop
8:         x += i + 1    z −= i
9: loop
10:    x −= n + 1    y −= 1
11: halt if y=0
```

P(b, c, d, b', c', d')

consider all runs
(nondeterminism)

d=0

RATIO(b', c', d', *F*(*B*))

One can compute an ***A*$_k$-amplifier** with *3k+2* counters,
in time polynomial in *k*, *n*

F-amplifier

RATIO(b, c, d, B)

- b = B
- c > 0
- d = b · c
- all other counters 0

For every fixed B:

RATIO(b, c, d, B)

```
1: x += 1    y += 1
2: loop
3:     x += 1    y += 1
4: for i = n down to 1 do
5:     loop
6:         x −= 1    z += 1
7:     loop
8:         x += i + 1    z −= i
9: loop
10:    x −= d + 1    y −= 1
11: halt if y = 0
```

P(b, c, d, b', c', d')

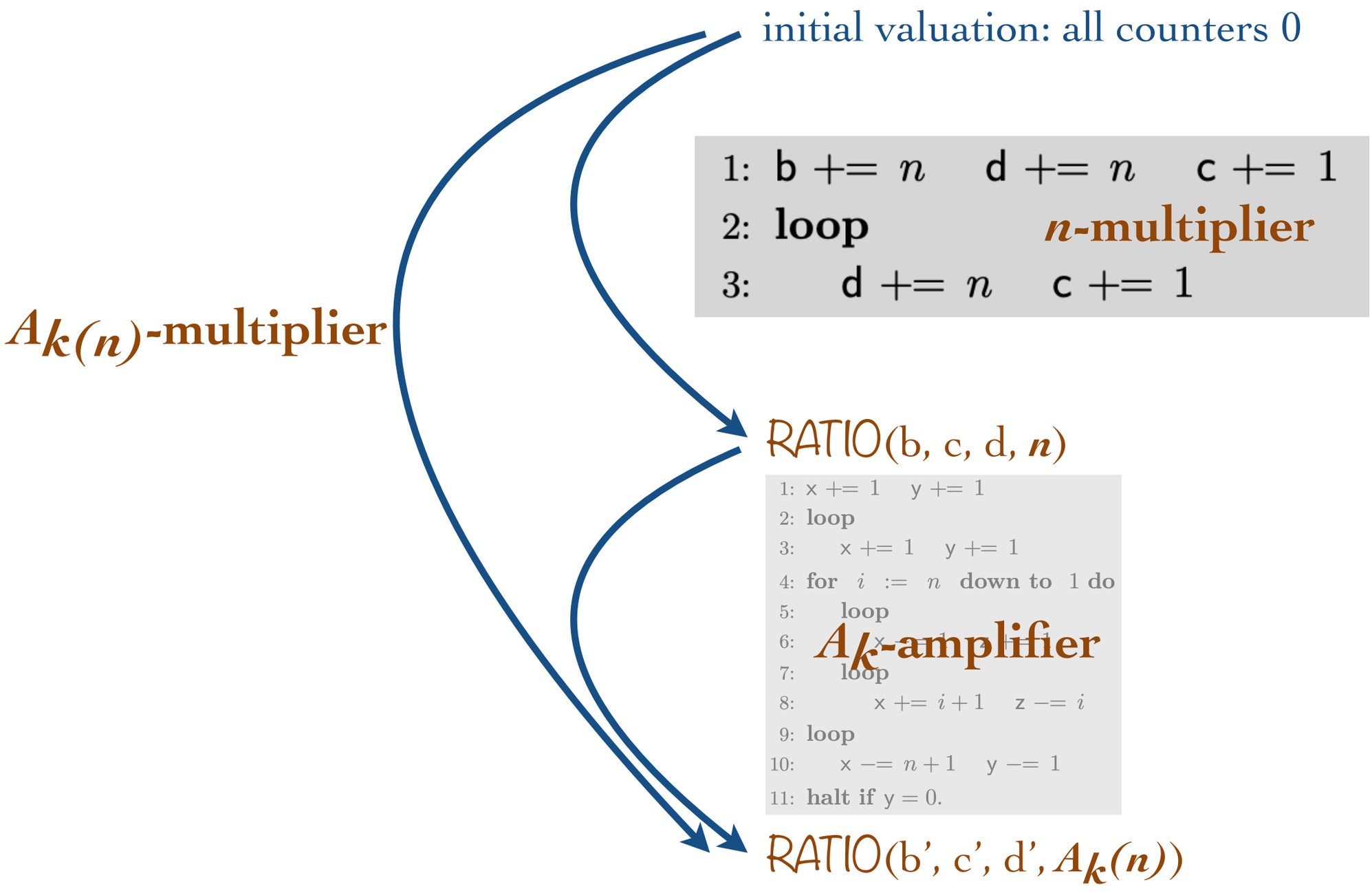
consider all runs (nondeterminism)

d=0

RATIO(b', c', d', $F(B)$)

One can compute an **A_k-amplifier** with $3k+2$ counters, in time polynomial in k, n

A_k-amplifier ⟶ $A_k(n)$-multiplier



Amplifier lifting

- ***A*1-amplifier:**

```
1: loop
2:     loop
3:         c −= 1   c′ += 1   d −= 1   d′ += 2
4:     loop
5:         c′ −= 1   c += 1   d −= 1   d′ += 2
6:     b −= 2   b′ += 4
7: loop
8:     c −= 1   c′ += 1   d −= 2   d′ += 4
9: b −= 2   b′ += 4
```

Amplifier lifting

- **A_1-amplifier:**

```
1: loop
2:     loop
3:         c −= 1   c′ += 1   d −= 1   d′ += 2
4:     loop
5:         c′ −= 1   c += 1   d −= 1   d′ += 2
6:     b −= 2   b′ += 4
7: loop
8:     c −= 1   c′ += 1   d −= 2   d′ += 4
9: b −= 2   b′ += 4
```

- **A_k-amplifier** ⟶ **A_{k+1}-amplifier**

Amplifier lifting

- **A_1-amplifier:**

```
1: loop
2:     loop
3:         c −= 1   c′ += 1   d −= 1   d′ += 2
4:     loop
5:         c′ −= 1   c += 1   d −= 1   d′ += 2
6:     b −= 2   b′ += 4
7: loop
8:     c −= 1   c′ += 1   d −= 2   d′ += 4
9: b −= 2   b′ += 4
```

- **A_k-amplifier** ⟶ **A_{k+1}-amplifier**

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ \ldots \ \circ A_i}_{n/4}(4) = A_i^{n/4}(4)$$

Amplifier lifting

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ \ldots \ \circ A_i}_{n/4}(4) = A_i^{n/4}(4)$$

A_k-amplifier

$\mathcal{P}$ $(b_1, c_1, d_1, b_2, c_2, d_2)$

Amplifier lifting

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ldots \circ A_i}_{n/4}(4) = A_i^{n/4}(4)$$

A_k-amplifier

$\mathcal{P}$ $(b_1, c_1, d_1, b_2, c_2, d_2)$

$\mathcal{M}$ 4-multiplier

Amplifier lifting

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ldots \circ A_i}_{n/4}(4) = A_i^{n/4}(4)$$

A_k-amplifier

$\mathcal{P}$ $(b_1, c_1, d_1, b_2, c_2, d_2)$

$\mathcal{M}$ 4-multiplier

$\mathcal{L}$ $(b_2, c_2, d_2, b_1, c_1, d_1)$ identity-amplifier

Amplifier lifting

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ \ldots \ \circ A_i(4)}_{n/4} = A_i^{n/4}(4)$$

A_k-amplifier

$\mathcal{P}$ $(b_1, c_1, d_1, b_2, c_2, d_2)$

$\mathcal{M}$ 4-multiplier

$\mathcal{L}$ $(b_2, c_2, d_2, b_1, c_1, d_1)$ identity-amplifier

```
1: M
2: loop
3:     P
4:     zero? d1
5:     L
6:     zero? d2
7: P
8: zero? d1
```

Amplifier lifting

$$A_{i+1}(n) = \underbrace{A_i \circ A_i \circ \ \ldots \ \circ A_i(4)}_{n/4} = A_i^{n/4}(4)$$

A_k-amplifier

$\mathcal{P}$ $(b_1, c_1, d_1, b_2, c_2, d_2)$

$\mathcal{M}$ 4-multiplier

$\mathcal{L}$ $(b_2, c_2, d_2, b_1, c_1, d_1)$ identity-amplifier

```
1: M
2: loop
3:     P
4:     zero? d1
5:     L
6:     zero? d2
7: P
8: zero? d1
```

instrumented

```
1: M
2: loop
3:     P
4:     zero? d1
5:     L
6:     zero? d2
7: P
8: zero? d1
```

(b, c, d, b_2, c_2, d_2)

A_{k+1}-amplifier

Open questions

- dimension-parametric complexity: **gap n-4 … 2n+4**
- low dimensions starting from 3
- extensions:
  - data Petri nets
  - pushdown Petri nets
  - branching VASS

Open questions

- dimension-parametric complexity: **gap n-4 … 2n+4**
- low dimensions starting from 3
- extensions:
  - data Petri nets
  - pushdown Petri nets
  - branching VASS

thank you!