Improved Ackermannian lower bound for the Petri nets reachability problem

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

Sławomir Lasota
University of Warsaw

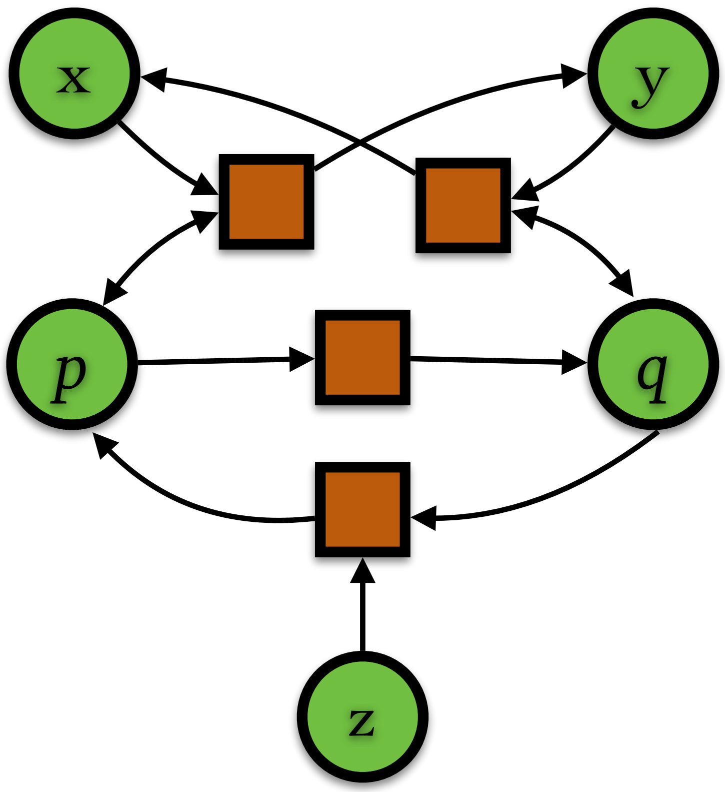
Jagiellonian TCS Seminar, 2022.01.26, online

Part I:

the reachability problem and its complexity

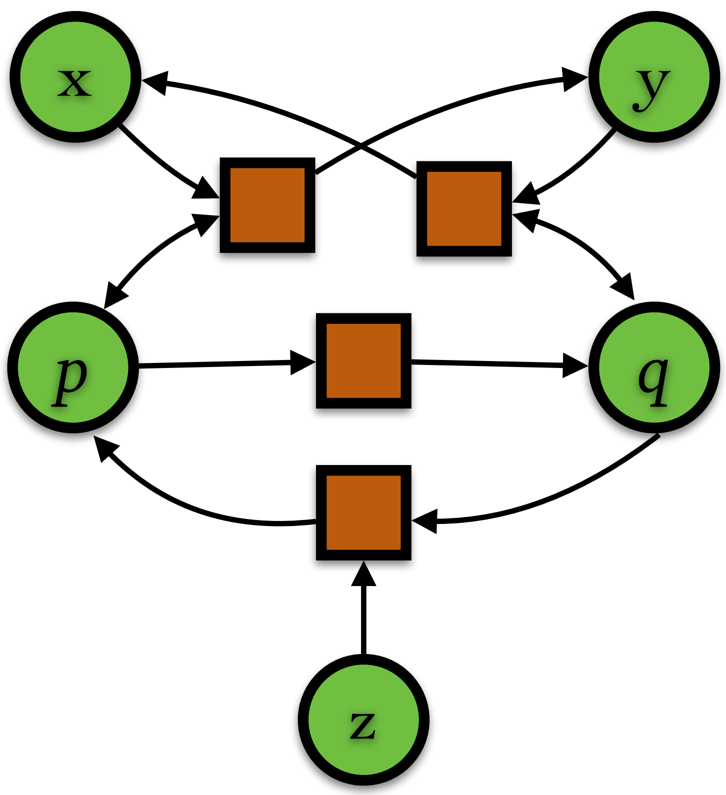
Many faces of Petri nets

- Petri nets:

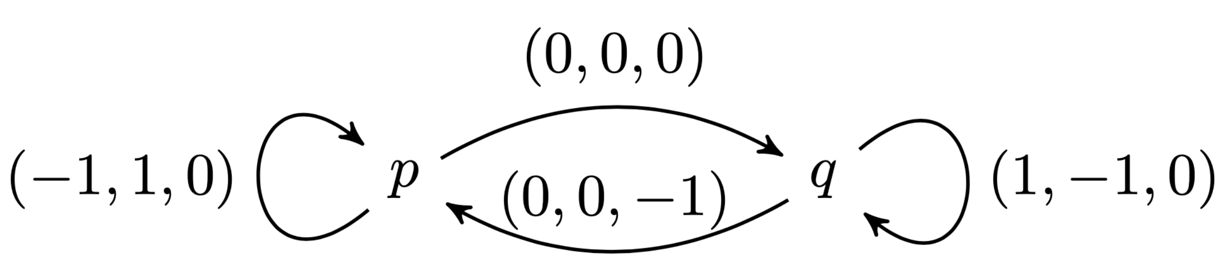


Many faces of Petri nets

- Petri nets:

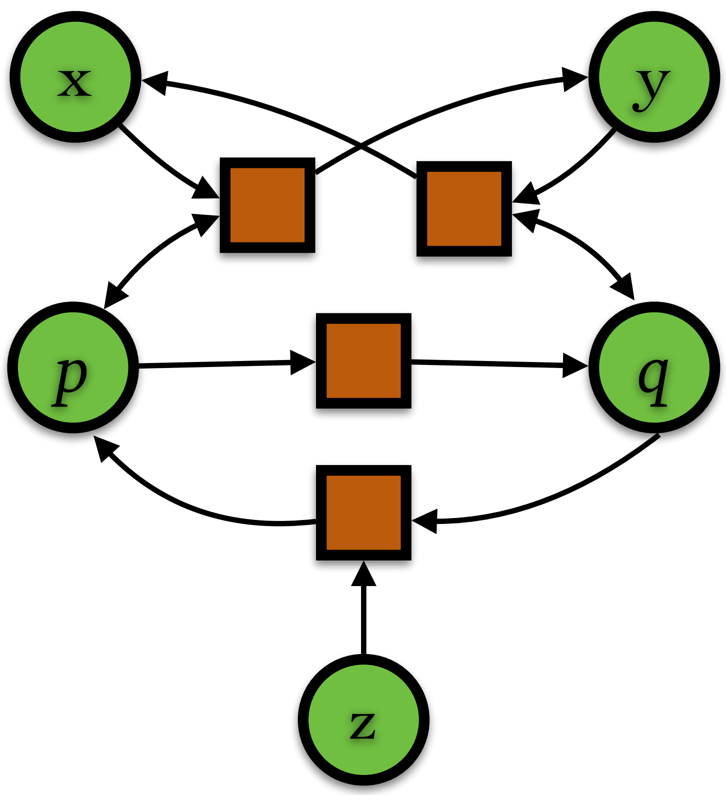


- vector addition systems with states:

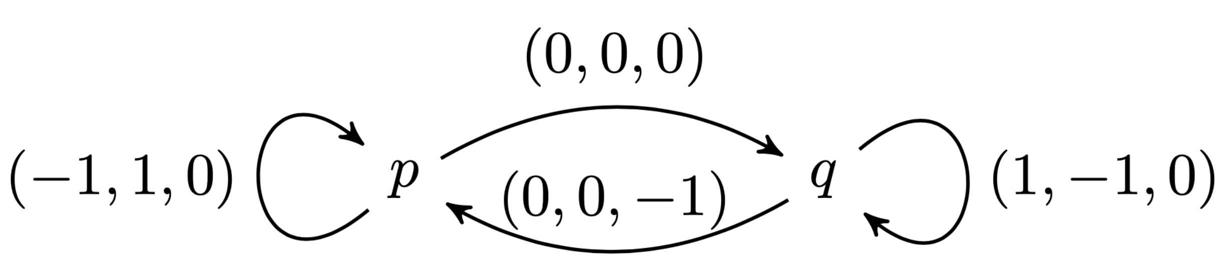


Many faces of Petri nets

- Petri nets:



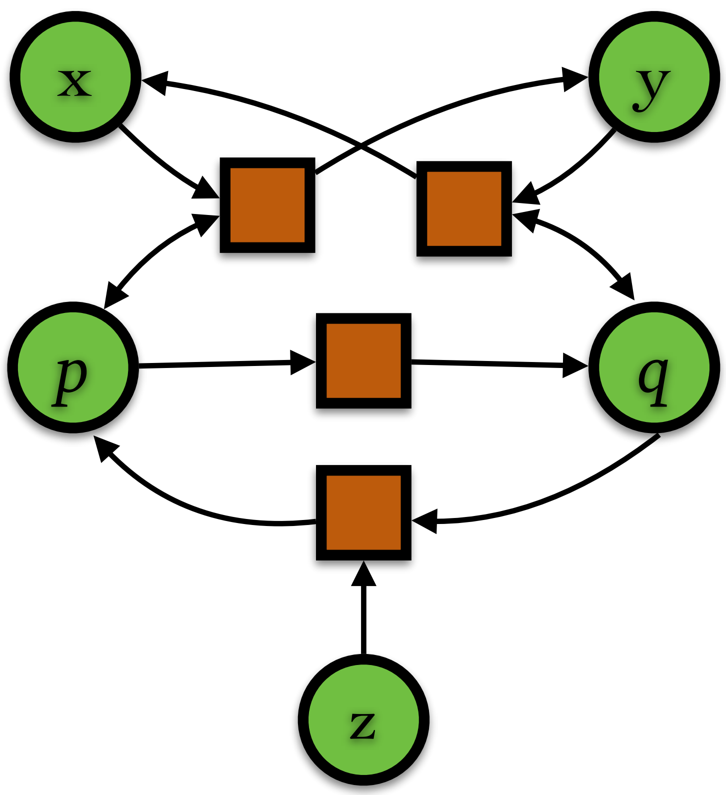
- vector addition systems with states:



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

Many faces of Petri nets

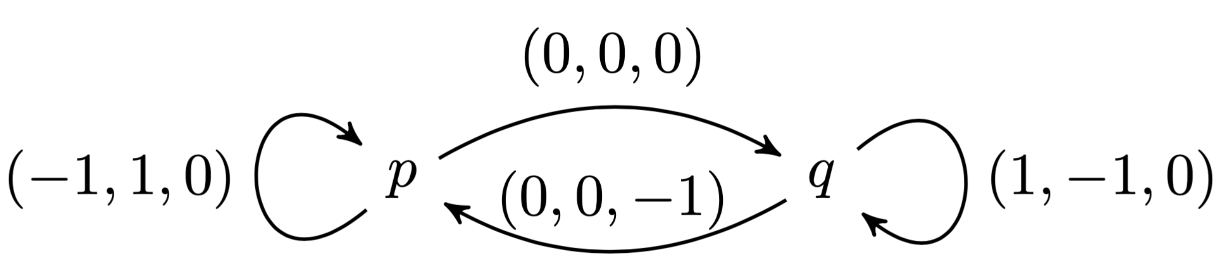
- Petri nets:



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

```
loop
    loop
        x −= 1    y += 1
    loop
        x += 1    y −= 1
    z −= 1
```

- vector addition systems with states:



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

Many faces of Petri nets

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

```
loop
    loop
        x −= 1    y += 1
    loop
        x += 1    y −= 1
    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:

$\mathsf{x} \mathrel{+}= n$	(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')

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 $x < n$

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 $\mathsf{x} < n$

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

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

initially: $x' = x = y = 0$

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

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

initially: $x' = x = y = 0$

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

initially: $x' = x = y = 0$

finally: $x' = 0 \quad x = 100 \quad y = 200$

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

can it halt? (successfully execute its halt command)

Reachability problem

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

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** $\mathsf{x}' = 0$.

why is it important?

can it halt? (successfully execute its halt command)

Reachability problem

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

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

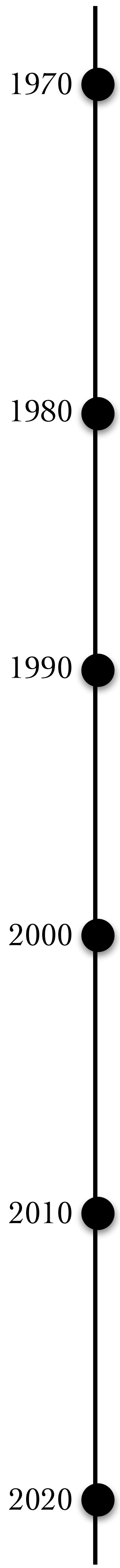
why is it important?

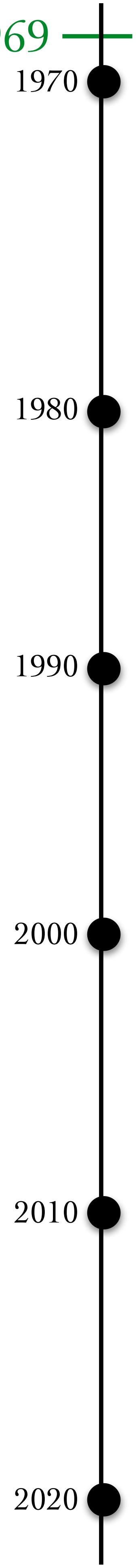
can it halt? (successfully execute its halt command)

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

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

can it halt?


1970
1980
1990
2000
2010
2020



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

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

1970

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

1980

1990

2000

2010

2020

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

1970

1976 EXPSPACE lower bound **[Lipton]**

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

1980

1990

2000

2010

2020

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

1990

2000

2010

2020

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]**

1990

2000

2010

2020

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

2000

2010

2020

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

2010

2020

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]**

KLMST decomposition

2000

2010

2020

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]**

KLMST decomposition

huge complexity gap!

2000

2010

2020

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]**

KLMST decomposition

1990

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

huge complexity gap!

2000

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

2010

2020

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]**

KLMST decomposition

1990

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

huge complexity gap!

2000

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

2010

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

2020

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]**

KLMST decomposition

huge complexity gap!

2000

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]**

2020

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]**

KLMST decomposition

huge complexity gap!

2000

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]**

2020

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]**

KLMST decomposition

1990

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

huge complexity gap!

2000

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

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]**

KLMST decomposition

huge complexity gap!

2000

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

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

2021

2022

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

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

gap

2020

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, n$$

2021

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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, 2^n$$

2022

2019 — Ackermannian upper bound F_ω [Leroux, Schmitz]

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

} gap

2020

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

2021

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

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

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

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

2021

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

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

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

$$A_1(n) = 2n$$

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

$$A_1(n) = 2n$$

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

$$A_3(n) = \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\right\}n$$

$$A_4(n) = \ldots$$

Fast growing functions and induced complexity classes

$$A_1(n) = 2n$$

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

$$A_1(n) = 2n$$

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

$$A_3(n) = \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\right\}n$$

$$A_4(n) = \dots$$

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

Fast growing functions and induced complexity classes

$$A_1(n) = 2n$$

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

$$A_1(n) = 2n$$

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

$$A_3(n) = \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\right\}n$$

$$A_4(n) = \ldots$$

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

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

dimension = number of counters

2019 Ackermannian upper bound F_ω [Leroux, Schmitz]

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

2020

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, n$$

2021

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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, 2^n$$

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

2021 improved Ackermannian lower bound [L.]

2022

dimension = number of counters

$\mathbf{F}_n$ -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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, n$$

2021

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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, 2^n$$

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

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

2022

dimension = number of counters

$\mathbf{F}_n$ -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

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

2021

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

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

$\mathbf{F}_n$ -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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Bigg\}\, n$$

2021

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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Bigg\}\, 2^n$$

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

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

2021

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

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

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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, n$$

2021

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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, 2^n$$

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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, n$$

2021

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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, 2^n$$

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] n-4

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

2020

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, n$$

still a gap!

2021

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

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}\, 2^n$$

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

Part II:

proof of the lower bound

F_k -hardness in dimension $3k+2$

The set computed by a counter program

initial valuation: all counters 0

```
x += 1    y += 1
loop
    x += 1    y += 1
for i := n down to 1 do
    loop
        x −= 1    z += 1
    loop
        x += i+1    z −= i
loop
    x −= n+1    y −= 1
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

```
x += 1    y += 1
loop
    x += 1    y += 1
for i := n down to 1 do
    loop
        x −= 1    z += 1
    loop
        x += i + 1    z −= i
loop
    x −= n + 1    y −= 1
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

```
x += 1   y += 1
loop
    x += 1   y += 1
for i := n down to 1 do
    loop
        x −= 1   z += 1
    loop
        x += i + 1   z −= i
loop
    x −= n + 1   y −= 1
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

```
x += 1    y += 1
loop
    x += 1    y += 1
for i := n down to 1 do
    loop
        x −= 1    z += 1
    loop
        x += i + 1    z −= i
loop
    x −= n + 1    y −= 1
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

```
x += 1   y += 1
loop
    x += 1   y += 1
for i := n down to 1 do
    loop
        x -= 1   z += 1
    loop
        x += i+1   z -= i
loop
    x -= n+1   y -= 1
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

$\Big\}$ RATIO(b, c, d, B)

10th Hilbert's problem!

B-multiplier

B - fixed positive integer

initial valuation: all counters 0

```
b += B    d += B    c += 1
loop
    d += B    c += 1
```

```
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

```
x += 1    y += 1
b += B    d += B    c += 1
loop
     d += B    c += 1
halt if y = 0.
```

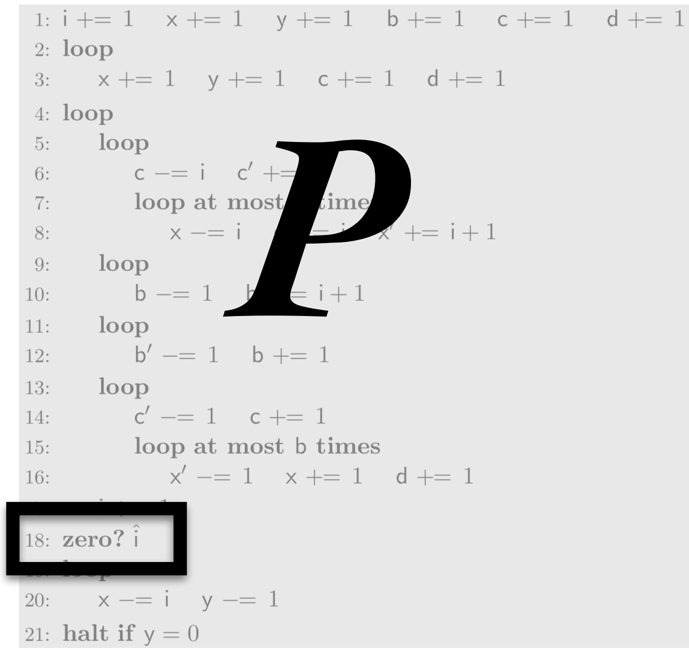
- b = B
- c > 0
- d = b · c
- all other counters 0

$\left.\begin{array}{l}\text{b} = B\\ \text{c} > 0\\ \text{d} = \text{b}\cdot\text{c}\\ \text{all other counters } 0\end{array}\right\}$ 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:



does it have a halting run
that does $(A_k(n)\text{-}1)/2$ zero tests?

F_k -hardness in dimension *3k+2*

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

```
i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
loop
    x += 1   y += 1   c += 1   d += 1
loop
    loop
        c −= i   c′ +=
        loop at most  time
            x −= i         x′ += i + 1
    loop
        b −= 1        i + 1
    loop
        b′ −= 1   b += 1
    loop
        c′ −= 1   c += 1
        loop at most b times
            x′ −= 1   x += 1   d += 1

zero? î

    x −= i   y −= 1
halt if y = 0
```

P

does it have a halting run
that does $(A_k(n)$-$1)/2$ zero tests?

program **without** 0-tests:

does it halt?

F_k-hardness in dimension $3k+2$

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

```
i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
loop
    x += 1   y += 1   c += 1   d += 1
loop
    loop
        c −= i   c′ +=
        loop at most   times
            x −= i        x += i+1
    loop
        b −= 1        i+1
    loop
        b′ −= 1   b += 1
    loop
        c′ −= 1   c += 1
        loop at most b times
            x′ −= 1   x += 1   d += 1

zero? î

    x −= i   y −= 1
halt if y = 0
```

P

does it have a halting run
that does $(A_k(n)\text{-}1)/2$ zero tests?

program **without** 0-tests:

```
x += 1   y += 1
loop
    x += 1   y += 1
for i := n down to 1 do
    loop
        x −= 1   z += 1
    loop
        x += i+1   z −= i
loop
    x −= n+1   y −= 1
halt if y = 0.
```

$A_k(n)$-multiplier M

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

```
i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
loop
    x += 1   y += 1   c += 1   d += 1
loop
    loop
        c −= i   c′ +=
        loop at most    mes
            x −= i   d        x += i+1
    loop
        b −= 1   b′        i+1
    loop
        b′ −= 1   b += 1
    loop
        loop at most b times
            x′ −= 1   x += 1   d += 1
    i += 1
zero? î
loop
    x −= i   y −= 1
halt if y = 0
```

P

instrumented

does it halt?

Instrumentation

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

```
i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
loop
    x += 1   y += 1   c += 1   d += 1
loop
    loop
        c −= i   c′ += [illegible]
        loop at most [illegible] tim[illegible]
            x −= i   d [illegible]   x′ += i + 1
    loop
        b −= 1   b′ [illegible] i + 1
    loop
        b′ −= 1   b += 1
    loop
        c′ −= 1   c += 1
        loop at most b times
            x′ −= 1   x += 1   d += 1
    i += 1
zero? î
loop
    x −= i   y −= 1
halt if y = 0
```

P
instrumented

Aim:
simulate $(A_k(n)\text{-}1)/2$ zero tests

Instrumentation

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

- **introduce fresh counters b, c, d**

```
i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
loop
    x += 1   y += 1   c += 1   d += 1
loop
    loop
        c −= i   c′ +=
        loop at most   tim
            x −= i   d −= i   x′ += i + 1
    loop
        b −= 1   b′ += i + 1
    loop
        b′ −= 1   b += 1
    loop
        c′ −= 1   c += 1
        loop at most b times
            x′ −= 1   x += 1   d += 1
    i += 1
zero? î
loop
    x −= i   y −= 1
halt if y = 0
```

P

- **instrumented**

Aim:
simulate $(A_k(n)\text{-}1)/2$ zero tests

Instrumentation

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

```
i += 1    x += 1    y += 1    b += 1    c += 1    d += 1
loop
    x += 1    y += 1    c += 1    d += 1
loop
    loop
        c −= i    c′ += [illegible]
        loop at most [illegible] times
            x −= i    d [illegible] i    x′ += i + 1
    loop
        b −= 1    b′ [illegible] i + 1
    loop
        b′ −= 1    b += 1
    loop
        c′ −= 1    c += 1
        loop at most b times
            x′ −= 1    x += 1    d += 1
    i += 1
zero? î
loop
    x −= i    y −= 1
halt if y = 0
```

P

instrumented

- **introduce fresh counters b, c, d**
- **instrument increments and decrements:**

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

Aim:
simulate *($A_k(n)$-1)/2* zero tests

Instrumentation

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

```
i += 1    x += 1    y += 1    b += 1    c += 1    d += 1
loop
    x += 1    y += 1    c += 1    d += 1
loop
    loop
        c −= i    c′ +=
        loop at most   times
            x −= i    d −= i    x′ += i + 1
    loop
        b −= 1    b′ += i + 1
    loop
        b′ −= 1    b += 1
    loop
        c′ −= 1    c += 1
        loop at most b times
            x′ −= 1    x += 1    d += 1
    i += 1
zero? î
loop
    x −= i    y −= 1
halt if y = 0
```

***P* instrumented**

- **introduce fresh counters b, c, d**
- **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 $(A_k(n)-1)/2$ zero tests

Instrumentation

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

```
i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
loop
    x += 1   y += 1   c += 1   d += 1
loop
    loop
        c −= i   c′ += [illegible]
        loop at most [illegible] time[illegible]
            x −= i   d[illegible] i   x′ += i + 1
    loop
        b −= 1   b[illegible] i + 1
    loop
        b′ −= 1   b += 1
    loop
        c′ −= 1   c += 1
        loop at most b times
            x′ −= 1   x += 1   d += 1
    i += 1
zero? i
loop
    x −= i   y −= 1
halt if y = 0
```

P

instrumented

Aim:
simulate $(A_k(n)\text{-}1)/2$ zero tests

- **introduce fresh counters b, c, d**
- **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

- **replace zero? x by**

```
ZERO? x:
loop
    y −= 1   x += 1   d −= 1
loop
    c −= 1   y += 1   d −= 1
loop
    y −= 1   c += 1   d −= 1
loop
    x −= 1   y += 1   d −= 1
b −= 2
```

Instrumentation

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

```
i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
loop
    x += 1   y += 1   c += 1   d += 1
loop
    loop
        c −= i   c′ +=
        loop at most    tim
            x −= i   d         x′ += i + 1
    loop
        b −= 1   b′      i + 1
    loop
        b′ −= 1   b += 1
    loop
        c′ −= 1   c += 1
        loop at most b times
            x′ −= 1   x += 1   d += 1
    i += 1
zero? î
loop
    x −= i   y −= 1
halt if y = 0
```

P

instrumented

Aim:
simulate $(A_k(n)\text{-}1)/2$ zero tests

- **introduce fresh counters b, c, d**
- **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

- **replace** **zero?** x **by**

```
ZERO? x:
loop
    y −= 1   x += 1   d −= 1
loop
    c −= 1   y += 1   d −= 1
loop
    y −= 1   c += 1   d −= 1
loop
    x −= 1   y += 1   d −= 1
b −= 2
```

- **replace** **halt** **by**

```
loop
    c −= 1   d −= 2
ZERO? c
halt if ..., d = 0.
```

merged halt of M and P

Simulation of a zero test

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

ZERO? x:

```
loop
    y -= 1    x += 1    d -= 1
loop
    c -= 1    y += 1    d -= 1
loop
    y -= 1    c += 1    d -= 1
loop
    x -= 1    y += 1    d -= 1
b -= 2
```

Simulation of a zero test

put x, y on
budget c

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

Zero? x:

1: **loop**
2: $\quad y \mathrel{-}= 1 \quad x \mathrel{+}= 1 \quad d \mathrel{-}= 1$
3: **loop**
4: $\quad c \mathrel{-}= 1 \quad y \mathrel{+}= 1 \quad d \mathrel{-}= 1$
5: **loop**
6: $\quad y \mathrel{-}= 1 \quad c \mathrel{+}= 1 \quad d \mathrel{-}= 1$
7: **loop**
8: $\quad x \mathrel{-}= 1 \quad y \mathrel{+}= 1 \quad d \mathrel{-}= 1$
9: $b \mathrel{-}= 2$

Simulation of a zero test

put x, y on **budget** c

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

Zero? x:

```
loop
    y -= 1    x += 1    d -= 1
loop
    c -= 1    y += 1    d -= 1
loop
    y -= 1    c += 1    d -= 1
loop
    x -= 1    y += 1    d -= 1
b -= 2
```

$$\begin{tikzcd} c & y \arrow[r, bend left] & x \end{tikzcd}$$

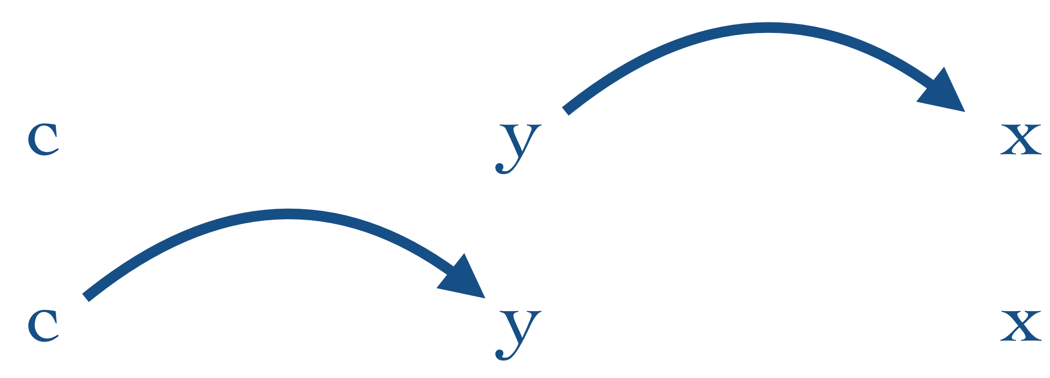
Simulation of a zero test

put x, y on **budget** c

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

<u>ZERO? x:</u>

```
loop
    y −= 1    x += 1    d −= 1
loop
    c −= 1    y += 1    d −= 1
loop
    y −= 1    c += 1    d −= 1
loop
    x −= 1    y += 1    d −= 1
b −= 2
```

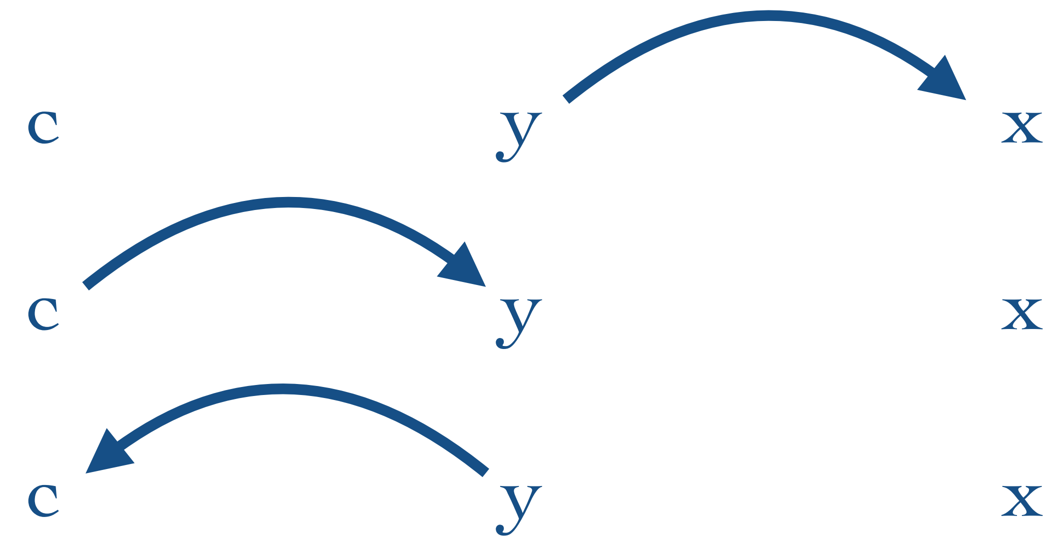


Simulation of a zero test

put x, y on **budget** c

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

```
ZERO? x:
loop
    y -= 1    x += 1    d -= 1
loop
    c -= 1    y += 1    d -= 1
loop
    y -= 1    c += 1    d -= 1
loop
    x -= 1    y += 1    d -= 1
b -= 2
```

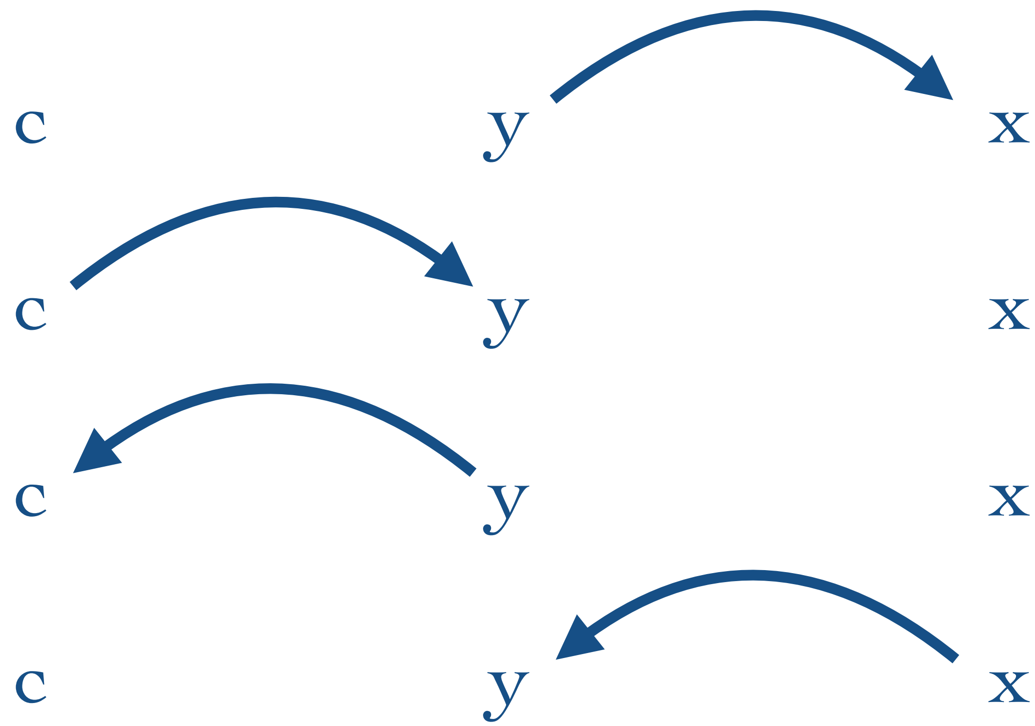


Simulation of a zero test

put x, y on
budget c

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

```
ZERO? x:
  loop
      y -= 1    x += 1    d -= 1
  loop
      c -= 1    y += 1    d -= 1
  loop
      y -= 1    c += 1    d -= 1
  loop
      x -= 1    y += 1    d -= 1
  b -= 2
```

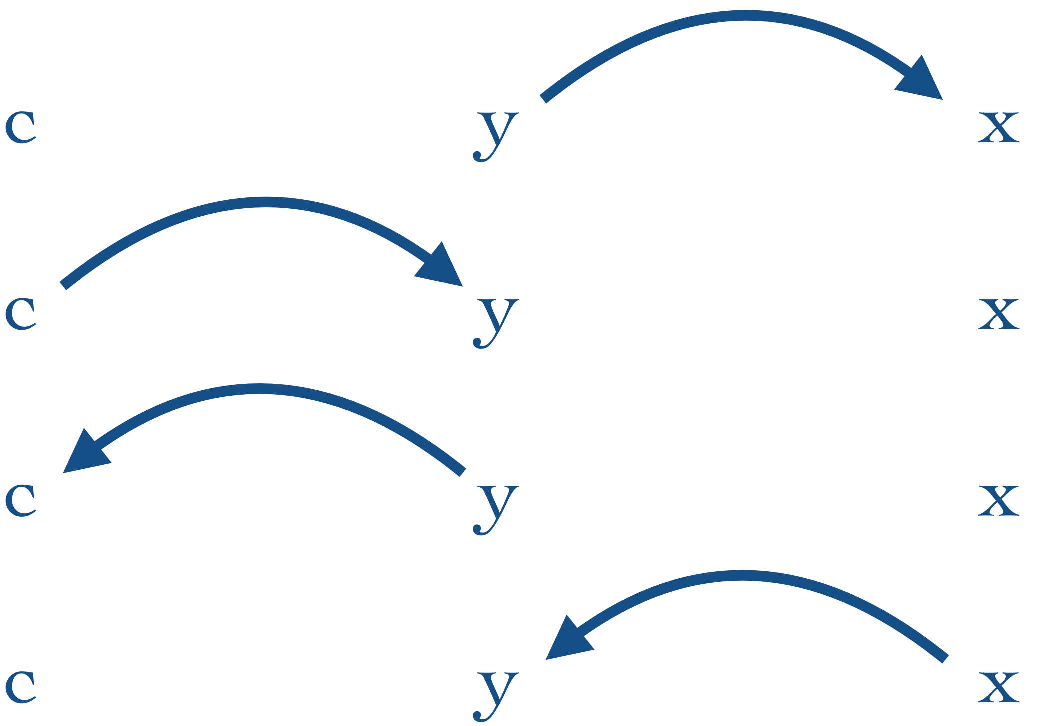


Simulation of a zero test

put x, y on
budget c

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

```
ZERO? x:
loop
      y -= 1    x += 1    d -= 1
loop
      c -= 1    y += 1    d -= 1
loop
      y -= 1    c += 1    d -= 1
loop
      x -= 1    y += 1    d -= 1
b -= 2
```



d decreases by <= $2 \cdot (c + x + y)$
b decreases by 2

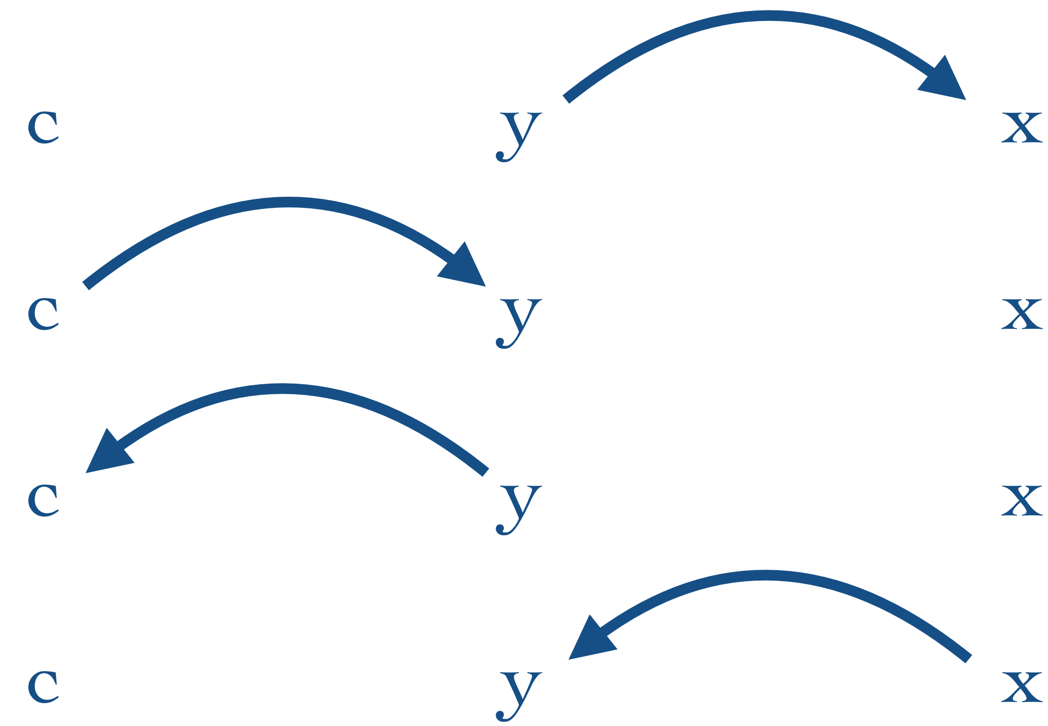
Simulation of a zero test

put x, y on **budget** c

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

$\underline{\textsc{Zero?}\,x:}$

```
loop
    y -= 1    x += 1    d -= 1
loop
    c -= 1    y += 1    d -= 1
loop
    y -= 1    c += 1    d -= 1
loop
    x -= 1    y += 1    d -= 1
b -= 2
```



d decreases by <= $2 \cdot (c + x + y)$
b decreases by 2

- d decreases by $2 \cdot (c + x + y)$ ⟶ x = 0 and y = c initially and finally
- d decreases by < $2 \cdot (c + x + y)$ ⟶ **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

A_k-amplifier $\longrightarrow$ **$A_k(n)$-multiplier**

…

…

A_2-amplifier

A_1-amplifier

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

initial valuation: all counters 0

```
x += 1   y += 1
loop
    x += 1   y += 1
for i := n down to 1 do
    loop
        x −= 1   z += 1
    loop
        x += i+1   z −= i
loop
    x −= n+1   y −= 1
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

```
x += 1   y += 1
loop
    x += 1   y += 1
for i := n down to 1 do
    loop
        x -= 1   z += 1
    loop
        x += i+1   z -= i
loop
    x -= n+1   y -= 1
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*)

```
x += 1    y += 1
loop
    x += 1    y += 1
for i := n down to 1 do
    loop
        x −= 1    z += 1
    loop
        x += i + 1    z −= i
loop
    x −= n + 1    y −= 1
halt if y = 0
```

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

d=0

consider all runs
(nondeterminism)

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) $\begin{cases} \bullet\ \text{b} = B \\ \bullet\ \text{c} > 0 \\ \bullet\ \text{d} = \text{b} \cdot \text{c} \\ \bullet\ \text{all other counters } 0 \end{cases}$

For every fixed B:

RATIO(b, c, d, B)

```
x += 1    y += 1
loop
    x += 1    += 1
for  i  =  n  down to  1 do
    loop
        x −= 1    z += 1
    loop
        x += i + 1    z −= i
loop
    x −=  1    y −= 1
halt if
```

$\boldsymbol{P}$**(b, c, d, b’, c’, d’)**

d=0

consider all runs
(nondeterminism)

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** $\longrightarrow$ $A_{k}(n)$**-multiplier**

initial valuation: all counters 0

```
b += n    d += n    c += 1
loop
    d += n    c += 1
```

n**-multiplier**

$A_{k(n)}$**-multiplier**

RATIO(b, c, d, n)

```
x += 1    y += 1
loop
    x += 1    y += 1
for i := n down to 1 do
    loop
        x −= 1    z += 1
    loop
        x += i+1    z −= i
loop
    x −= n+1    y −= 1
halt if y = 0.
```

A_k**-amplifier**

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

Amplifier lifting

- ***A₁*-amplifier:**

```
loop
    loop
        c −= 1    c′ += 1    d −= 1    d′ += 2
    loop
        c′ −= 1    c += 1    d −= 1    d′ += 2
    b −= 2    b′ += 4
loop
    c −= 1    c′ += 1    d −= 2    d′ += 4
b −= 2    b′ += 4
```

Amplifier lifting

- ***A₁*-amplifier:**

```
loop
    loop
        c −= 1    c′ += 1    d −= 1    d′ += 2
    loop
        c′ −= 1    c += 1    d −= 1    d′ += 2
    b −= 2    b′ += 4
loop
    c −= 1    c′ += 1    d −= 2    d′ += 4
b −= 2    b′ += 4
```

- ***A_k*-amplifier** $\longrightarrow$ ***A_{k+1}*-amplifier**

Amplifier lifting

- **A_1-amplifier:**

```
loop
    loop
        c −= 1    c′ += 1    d −= 1    d′ += 2
    loop
        c′ −= 1    c += 1    d −= 1    d′ += 2
    b −= 2    b′ += 4
loop
    c −= 1    c′ += 1    d −= 2    d′ += 4
b −= 2    b′ += 4
```

- **A_k-amplifier** $\longrightarrow$ **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

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

A_k-amplifier

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

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

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

A_k-amplifier

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

$\mathcal{M}$ 4-multiplier

```
M
loop
    P
    zero? d1
    L
    zero? d2
P
zero? d1
```

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

Amplifier lifting

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

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

```
M
loop
    P
    zero? d1
    L
    zero? d2
P
zero? d1
```

instrumented

```
M
loop
    P
    zero? d1
    L
    zero? d2
P
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!