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:

• vector addition systems with states:

• counter programs without 0-tests:

1: loop
2: loop
3: \( x \leftarrow 1 \quad y \leftarrow 1 \)
4: loop
5: \( x \leftarrow 1 \quad y \leftarrow 1 \)
6: \( z \leftarrow 1 \)

• vector addition systems
• counter automata without 0-tests
• multiset rewriting
• …
Many faces of Petri nets

- counter programs *without 0-tests*:

```plaintext
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:

\[
\begin{array}{ll}
\text{x } & \text{+= } n & \text{(increment counter x by n)} \\
\text{x } & \text{-= } n & \text{(decrement counter x by n)} \\
\text{goto } L \text{ or } L' & \text{(jump to either line L or line L')} \\
\end{array}
\]

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:

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

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

- **halt** if \( x_1, \ldots, x_l = 0 \) (terminate provided all the listed counters are zero)
- **abort** if \( x < n \)
- 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:

\[ x \; \text{+=} \; n \] (increment counter \( x \) by \( n \))
\[ x \; \text{-=} \; n \] (decrement counter \( x \) by \( n \))
\textbf{goto } L \text{ or } L' \quad \text{(jump to either line } L \text{ or line } L')

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

\textbf{halt if } x_1, \ldots, x_l = 0 \quad \text{(terminate provided all the listed counters are zero)}

\text{otherwise abort}

Example:

\begin{align*}
1: \ x' & \; \text{+=} \; 100 \\
2: \ \text{goto } 5 \text{ or } 3 \\
3: \ x & \; \text{+=} \; 1 \quad x' \; \text{-=} \; 1 \quad y \; \text{+=} \; 2 \\
4: \ \text{goto } 2 \\
5: \ \text{halt if } x' = 0.
\end{align*}

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:

\begin{align*}
&x += n & \text{(increment counter } x \text{ by } n) \\
&x -= n & \text{(decrement counter } x \text{ by } n) \\
&\text{goto } L \text{ or } L' & \text{(jump to either line } L \text{ or line } L')
\end{align*}

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

\begin{align*}
\text{halt if } x_1, \ldots, x_l = 0 & \quad \text{(terminate provided all} \\
& \text{the listed counters are zero)}
\end{align*}

\text{abort if } x < n

Example:

\begin{align*}
1: & \quad x' += 100 \\
2: & \quad \text{goto 5 or 3} \\
3: & \quad x += 1 \quad x' -= 1 \quad y += 2 \\
4: & \quad \text{goto 2} \\
5: & \quad \text{halt if } x' = 0.
\end{align*}

Initially: \(x' = x = y = 0\)

1: \(x' += 100\)

2: \text{loop}

3: \(x += 1 \quad x' -= 1 \quad y += 2\)

4: \text{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:

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

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

\textbf{halt if} \( x_1, \ldots, x_l = 0 \) (terminate provided all the listed counters are zero) otherwise abort

**Example:**

1: \( x' += 100 \)
2: \textbf{goto} 5 or 3
3: \( x += 1 \) \( x' -= 1 \) \( y += 2 \)
4: \textbf{goto} 2
5: \textbf{halt if} \( x' = 0 \).

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:

\[
\begin{align*}
x &\mathrel{+}= n & \text{(increment counter } x \text{ by } n) \\
x &\mathrel{-}= n & \text{(decrement counter } x \text{ by } n) \\
goto L \text{ or } L' & \text{ (jump to either line } L \text{ or line } L')
\end{align*}
\]

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

\[
\text{halt if } x_1, \ldots, x_L = 0 \quad \text{(terminate provided all the listed counters are zero)}
\]

otherwise abort

abort if \( x < n \)

Example:

\[
\begin{align*}
1: & \quad x' \mathrel{+}= 100 \\
2: & \quad \text{goto 5 or 3} \\
3: & \quad x += 1 \quad x' \mathrel{-}= 1 \quad y += 2 \\
4: & \quad \text{goto 2} \\
5: & \quad \text{halt if } x' = 0.
\end{align*}
\]

initially: \( x' = x = y = 0 \)

finally: \( x' = 0 \quad x = 100 \quad y = 200 \)
Reachability problem

Reachability problem: given a counter program without zero tests

1: \( x' += 100 \)
2: goto 5 or 3
3: \( x += 1 \quad x' -= 1 \quad y += 2 \)
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: \texttt{goto 5 or 3}
3: \( x += 1 \quad x' -= 1 \quad y += 2 \)
4: \texttt{goto 2}
5: \texttt{halt if } 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: \( x' += 100 \)
2: \texttt{goto 5 or 3} \hspace{1cm} \textbf{why is it important?}
3: \( x += 1 \quad x' -= 1 \quad y += 2 \)
4: \texttt{goto 2}
5: \texttt{halt if } x' = 0. \hspace{1cm} \text{(successfully execute its halt command)}

Can it halt?

Coverability problem: given a counter program without zero tests

1: \( x' += 100 \)
2: \texttt{goto 5 or 3} \hspace{1cm} \text{with trivial halt command}
3: \( x += 1 \quad x' -= 1 \quad y += 2 \)
4: \texttt{goto 2}
5: \texttt{halt.}

Can it halt?
decidability of coverability [Karp, Miller]

- 1969
- 1970
- 1980
- 1990
- 2000
- 2010
- 2020
1969 — decidability of coverability [Karp, Miller]

1976 — EXPSPACE lower bound [Lipton]
1969: decidability of coverability [Karp, Miller]

1976: EXPSPACE lower bound [Lipton]

1977: (incomplete) decidability of reachability [Sacerdote, Tenney]
decidability of coverability [Karp, Miller]

EXPSPACE lower bound [Lipton]

(incomplete) decidability of reachability [Sacerdote, Tenney]

EXPSPACE algorithm for coverability [Rackoff]
1969 — decidability of coverability [Karp, Miller]

1976 — EXPSPACE lower bound [Lipton]

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

1978 — EXPSPACE algorithm for coverability [Rackoff]

1981 — decidability of reachability [Mayr]
1969  decidability of coverability [Karp, Miller]

1976  EXPSPACE lower bound [Lipton]
1977  (incomplete) decidability of reachability [Sacerdote, Tenney]
1978  EXPSPACE algorithm for coverability [Rackoff]

1980  decadibility of reachability [Mayr]
1982  decidability of reachability - simplified proof [Kosaraju]
1969 — decidability of coverability [Karp, Miller]

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]

1976 — EXPSPACE lower bound [Lipton]
1977 — (incomplete) decidability of reachability [Sacerdote, Tenney]
1978 — EXPSPACE algorithm for coverability [Rackoff]

1981 — decidability of reachability [Mayr]
1982 — decidability of reachability - simplified proof [Kosaraju]

1990 — KLMST decomposition
1992 — decidability of reachability - refined data structure [Lambert]
1969 — decidability of coverability [Karp, Miller]

1976 — EXPSPACE lower bound [Lipton]

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

1978 — EXPSPACE algorithm for coverability [Rackoff]

1981 — decidability of reachability [Mayr]

1982 — decidability of reachability - simplified proof [Kosaraju]

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

KLMST decomposition

huge complexity gap!
<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1969</td>
<td>decidability of coverability</td>
<td>[Karp, Miller]</td>
</tr>
<tr>
<td>1976</td>
<td>EXPSPACE lower bound</td>
<td>[Lipton]</td>
</tr>
<tr>
<td>1977</td>
<td>(incomplete) decidability of reachability</td>
<td>[Sacerdote, Tenney]</td>
</tr>
<tr>
<td>1978</td>
<td>EXPSPACE algorithm for coverability</td>
<td>[Rackoff]</td>
</tr>
<tr>
<td>1981</td>
<td>decidability of reachability</td>
<td>[Mayr]</td>
</tr>
<tr>
<td>1982</td>
<td>decidability of reachability - simplified proof</td>
<td>[Kosaraju]</td>
</tr>
<tr>
<td>1992</td>
<td>decidability of reachability - refined data structure</td>
<td>[Lambert]</td>
</tr>
<tr>
<td>2009</td>
<td>decidability of reachability by Presburger invariants</td>
<td>[Leroux]</td>
</tr>
</tbody>
</table>

KLMST decomposition

huge complexity gap!
decidability of coverability [Karp, Miller]

EXPSPACE lower bound [Lipton]

(incomplete) decidability of reachability [Sacerdote, Tenney]

EXPSPACE algorithm for coverability [Rackoff]

decidability of reachability [Mayr]

decidability of reachability - simplified proof [Kosaraju]

decidability of reachability - refined data structure [Lambert]

decidability of reachability by Presburger invariants [Leroux]

decidability of reachability without KLMST decomposition [Leroux]

huge complexity gap!
1969 — decidability of coverability \[\text{[Karp, Miller]}\]

1976 — EXPSPACE lower bound \[\text{[Lipton]}\]
1977 — (incomplete) decidability of reachability \[\text{[Sacerdote, Tenney]}\]
1978 — EXPSPACE algorithm for coverability \[\text{[Rackoff]}\]

1981 — decidability of reachability \[\text{[Mayr]}\]
1982 — decidability of reachability - simplified proof \[\text{[Kosaraju]}\]

1990 —

1992 — decidability of reachability - refined data structure \[\text{[Lambert]}\]

1990 —

2000 —

2009 — decidability of reachability by Presburger invariants \[\text{[Leroux]}\]
2010 —
2011 — decidability of reachability without KLMST decomposition \[\text{[Leroux]}\]

2015 — first upper bound \(F_{\omega^3}\) \[\text{[Leroux, Schmitz]}\]
2019 — Ackermannian upper bound \(F_{\omega}\) \[\text{[Leroux, Schmitz]}\]

2020 —

huge complexity gap!
1969  decidability of coverability [Karp, Miller]

1976  EXPSPACE lower bound [Lipton]
1977  (incomplete) decidability of reachability [Sacerdote, Tenney]
1978  EXPSPACE algorithm for coverability [Rackoff]

1981  decidability of reachability [Mayr]
1982  decidability of reachability - simplified proof [Kosaraju]

1990

1992  decidability of reachability - refined data structure [Lambert]

1992  KLMST decomposition

1992  huge complexity gap!

2009  decidability of reachability by Presburger invariants [Leroux]
2010  decidability of reachability without KLMST decomposition [Leroux]
2015  first upper bound $F^{\omega_3}$ [Leroux, Schmitz]
2019  Ackermannian upper bound $F^\omega$ [Leroux, Schmitz]
2019  TOWER lower bound $F_3$ [Czerwiński, L., Lazic, Leroux, Mazowiecki]
1969  
1970 — decidability of coverability [Karp, Miller]  

1976  
1977 — 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]  

1992  
1999  
2000  
2009  
2010 — decidability of reachability by Presburger invariants [Leroux]  
2011 — decidability of reachability without KLMST decomposition [Leroux]  
2015 — first upper bound $F_{\omega^3}$ [Leroux, Schmitz]  
2019 — Ackermannian upper bound $F_\omega$ [Leroux, Schmitz]  
2019  
2020 — TOWER lower bound $F_3$ [Czerwiński, L., Lazic, Leroux, Mazowiecki]  

huge complexity gap!
2019 — Ackermannian upper bound $F_\omega$ [Leroux, Schmitz]

2019 — TOWER lower bound $F_3$ [Czerwiński, L., Lazic, Leroux, Mazowiecki]
Ackermannian upper bound $F_\omega$ [Leroux, Schmitz]

Tower lower bound $F_3$ [Czerwiński, L., Lazic, Leroux, Mazowiecki]

\[ 2^{2^2\cdots2} \begin{array} \end{array} n \]

“Super”-Tower lower bound $F_3$ [Czerwiński, L., Orlikowski]

\[ 2^{2^2\cdots2} \begin{array} \end{array} 2^n \]
2019 — Ackermannian upper bound $F_\omega$ \cite{Leroux, Schmitz} \] gap

2019 — TOWER lower bound $F_3$ \cite{Czerwinski, L., Lazic, Leroux, Mazowiecki} \] gap

\[
\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{n}\]

2020

2021 — “super”-TOWER lower bound $F_3$ \cite{Czerwinski, L., Orlikowski}

\[
\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{n} \}

2^{2n}

2021 — Ackermannian lower bound \cite{Czerwinski, Orlikowski} \cite{Leroux} \] gap closed!
Ackermannian upper bound $F_\omega$ [Leroux, Schmitz]

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

\[
\left\{ \begin{array}{c} 2^2 \cdot 2 \cdots 2 \end{array} \right\}^n
\]

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

\[
\left\{ \begin{array}{c} 2^2 \cdot 2 \cdots 2 \end{array} \right\} 2^n
\]

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

improved Ackermannian lower bound [L.]
Fast growing functions and induced complexity classes

\[ A_1(n) = 2n \]

\[ A_{i+1}(n) = A_i \circ A_i \circ \ldots \circ A_i(1) = A_i^n(1) \]
Fast growing functions and induced complexity classes

\[ A_1(n) = 2n \]

\[ A_{i+1}(n) = A_i \circ A_i \circ \ldots \circ A_i(1) = A_i^n(1) \]

\[ A_1(n) = 2n \]

\[ A_2(n) = 2^n \]

\[ A_3(n) = 2^{2^n} \]

\[ A_4(n) = \ldots \]
Fast growing functions and induced complexity classes

\[ A_1(n) = 2n \]

\[ A_{i+1}(n) = A_i \circ A_i \circ \ldots \circ A_i(1) = A_i^n(1) \]

\[ FF_i = \bigcup_m \text{FDTIME}(A_i^m(n)) \]

\[ A_1(n) = 2n \]
\[ A_2(n) = 2^n \]
\[ A_3(n) = 2^{2^n} \]
\[ A_4(n) = \ldots \]
Fast growing functions and induced complexity classes

\[ A_1(n) = 2n \]

\[ A_{i+1}(n) = A_i \circ A_i \circ \ldots \circ A_i(1) = A_i^n(1) \]

\[ n \]

\[ FF_i = \bigcup_{m} \text{FDTIME}(A_i^m(n)) \]

\[ F_i = \bigcup_{p \in FF_{i-1}} \text{DTIME}(A_i(p(n))) \]
2019 — Ackermannian upper bound $F_\omega$ \cite{Leroux, Schmitz}

2019 — TOWER lower bound $F_3$ \cite{Czerwinski, L., Lazic, Leroux, Mazowiecki}

2020

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

2021 — super-TOWER lower bound $F_3$ \cite{Czerwinski, L., Orlikowski}

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

2021 — Ackermannian lower bound \cite{Czerwinski, Orlikowski, Leroux}

2021 — improved Ackermannian lower bound \cite{L.}
\( F_n \) -membership in dimension:

- **2019** — Ackermannian upper bound \( F_\omega \) [Leroux, Schmitz]
- **2019** — TOWER lower bound \( F_3 \) [Czerwiński, L., Lazic, Leroux, Mazowiecki]

\[
\left\{ 2^2 2^2 \cdots 2^2 \right\}^n
\]

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

\[
\left\{ 2^2 2^2 \cdots 2^2 \right\} 2^n
\]

- **2021** — Ackermannian lower bound [Czerwiński, Orlikowski] [Leroux]
- **2021** — improved Ackermannian lower bound [L.]

**Dimension = number of counters**
$F_n$ -membership in dimension:

$F_n$ -hardness in dimension:

2019 — Ackermannian upper bound $F_\omega$ [Leroux, Schmitz]

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

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

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

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

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

2021 — improved Ackermannian lower bound [L.]

dimension = number of counters
Dimension = number of counters

- **Fₙ** membership in dimension: $F \omega$
  - 2019: Ackermannian upper bound $F \omega$ [Leroux, Schmitz]
  - 2019: TOWER lower bound $F_3$ [Czerwiński, L., Lazic, Leroux, Mazowiecki]

- **Fₙ** hardness in dimension:
  - 2021: Ackermannian lower bound [Czerwiński, Orlikowski] [Leroux]
  - 2021: Improved Ackermannian lower bound [L.]

\[
\begin{array}{c}
2^2 2^2 \cdots 2^n \\
2^n
\end{array}
\]
dimension = number of counters

\[ F_n \text{-membership in dimension:} \]

2019 — Ackermannian upper bound \( F_\omega \) [Leroux, Schmitz]

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

\[
\left\{ 2^{2^2 \cdots 2} \right\}^n
\]

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

\[
\left\{ 2^{2^2 \cdots 2} \right\} 2^n
\]

\[ F_n \text{-hardness in dimension:} \]

2021 — Ackermannian lower bound [Czerwiński, Orlikowski] [Leroux] \( 6n \quad 4n+5 \)

2021 — improved Ackermannian lower bound [L.]
$F_n$ - membership in dimension:

- 2019: Ackermannian upper bound $F_\omega$ [Leroux, Schmitz]
- 2019: TOWER lower bound $F_3$ [Czerwiński, L., Lazic, Leroux, Mazowiecki]

$$2^2^2^\ldots^2^n$$

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

$$2^2^2^\ldots^2^n$$

$F_n$ - hardness in dimension:

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

dimension = number of counters
Ackermannian upper bound $F_\omega$ [Leroux, Schmitz]

2019

TORER lower bound $F_3$ [Czerwiński, L., Lazic, Leroux, Mazowiecki]

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

2019

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

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

2021

Acknowledgment in dimension:

$F_n$-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
dimension = number of counters

$F_n$ -membership in dimension:

2019  Ackermannian upper bound $F_{\omega}$ [Leroux, Schmitz]

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

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

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

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

still a gap!

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

2021  improved Ackermannian lower bound [L.]

2021  further improved Ackermannian lower bound [Leroux]

$F_n$ -hardness in dimension:

2021  $6n \quad 4n+5$

2021  $3n+2$

2022  $2n+4$
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

```plaintext
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: $x \leftarrow x + 1 \quad y \leftarrow y + 1$
2: loop
3: $x \leftarrow x + 1 \quad y \leftarrow y + 1$
4: for $i := n$ down to $1$ do
5:  loop
6:  $x \leftarrow x + 1 \quad z \leftarrow z + 1$
7:  loop
8:  $x \leftarrow x + i + 1 \quad z \leftarrow z - i$
9:  loop
10: $x \leftarrow n + 1 \quad y \leftarrow y + 1$
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 loop
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)

- $b = B$
- $c > 0$
- $d = b \cdot c$
- all other counters 0

$\text{RATIO}(b, c, d, B)$
**$B$-multiplier**

$B$ - fixed positive integer

initial valuation: all counters 0

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>$x := x + 1$</td>
</tr>
<tr>
<td>2:</td>
<td>loop</td>
</tr>
<tr>
<td>3:</td>
<td>$x := x + 1$</td>
</tr>
<tr>
<td>4:</td>
<td>for $i := n$</td>
</tr>
<tr>
<td>5:</td>
<td>loop</td>
</tr>
<tr>
<td>6:</td>
<td>$x := x - 1$</td>
</tr>
<tr>
<td>7:</td>
<td>loop</td>
</tr>
<tr>
<td>8:</td>
<td>$x := x + 1$</td>
</tr>
<tr>
<td>9:</td>
<td>loop</td>
</tr>
<tr>
<td>10:</td>
<td>$x := n + 1$</td>
</tr>
<tr>
<td>11:</td>
<td>halt if $y = 0$.</td>
</tr>
</tbody>
</table>

- $b = B$
- $b > 0$?
- $c > 0$
- $d = b \cdot c$
- all other counters 0

3 distinguished counters $b, c, d$

consider all runs (nondeterminism)

$\text{RATIO}(b, c, d, B)$
**B-multiplier**

*B - fixed positive integer*

**initial valuation:** all counters 0

```plaintext
1: x += 1  y += 1
2: loop
3:  x += 1  y += 1
4: for i := n down to 1 do loop
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.
```

- \( b = B \)
- \( b > 0 \)?
- \( c > 0 \)
- \( d = b \cdot c \)
- all other counters 0

3 distinguished counters \( b, c, d \)

consider all runs (nondeterminism)

\[ \text{RATIO}(b, c, d, B) \]

10th Hilbert’s problem!
**B-multiplier**

*B - fixed positive integer*

Initial valuation: all counters 0

\begin{align*}
1: & \quad b += B \quad d += B \quad c += 1 \\
2: & \quad \text{loop} \\
3: & \quad d += B \quad c += 1 \\
\end{align*}

\begin{itemize}
  \item b = B \\
  \item c > 0 \\
  \item d = b \cdot c \\
  \item all other counters 0
\end{itemize}

\[ \text{RATIO}(b, c, d, B) \]
**B-multiplier**

*B*- fixed positive integer

1. initial valuation: all counters 0
   - $b = B$
   - $c > 0$
   - $d = b \cdot c$
   - all other counters 0

2. loop

3. \[ \text{RATIO}(b, c, d, B) \]

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 \textbf{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' := i
7:       loop at most \( i \) times
8:         x := i
9:     loop
10:       b := 1  i := 1  i := 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: zero? i
18:   x := 1  y := 1
19:   halt if \( y = 0 \)
```

does it have a halting run
that does \((A_k(n)-1)/2\) zero tests?
\( F_k \)-hardness in dimension \( 3k+2 \)

program of size \( n \) with two 0-tested counters:

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

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:

```
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 := i + 1
7:     loop at most \( i \) times
8:     x := i  c := i + 1
9:   loop
10:   b := 1  /  i := i + 1
11: loop
12:   b' := i  b := 1
13: loop
14:   c' := i  c := 1
15:   loop at most \( i \) times
16: x' := 1  x := 1  d := 1
17: zero? i
18: x := 1  y := 1
19: halt if \( y = 0 \)
```

does it have a halting run that does \((A_k(n) - 1)/2\) 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:   x := 1  z := i
6:   loop
7:     x := i + 1  z := i
8:   loop
9:   x := n + 1  y := 1
10: halt if \( y = 0 \).
```

\( A_k(n) \)-multiplier \( M \)

\( \text{RATIO}(b, c, d, A_k(n)) \)

```
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 := i + 1
7:     loop at most \( i \) times
8:     x := i  d := i + 1
9:   loop
10:   b := 1  b := 1
11: loop
12:   b' := i  b := 1
13: loop
14:   c' := i  c := 1
15:   loop at most \( i \) times
16: x' := 1  x := 1  d := 1
17: i := 1
18: zero? i
19: loop
20:   x := i  y := 1
21: halt if \( y = 0 \)
```

does it halt?
Instrumentation

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

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

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

```plaintext
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' += 1
7: loop at most $b$ times
8: x' = i  x = i + 1
9: loop
10: b -= i  b = i + 1
11: loop
12: c' -= 1  c += 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 $y = 0$
```

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

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

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

<table>
<thead>
<tr>
<th>command</th>
<th>replaced by</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x += 1$</td>
<td>$x += 1$  $c -= 1$</td>
</tr>
<tr>
<td>$x -= 1$</td>
<td>$x -= 1$  $c += 1$</td>
</tr>
</tbody>
</table>

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:     $c -= i$  $c' += 1$
7:      loop at most $i$ times
8:       $x -= i$  $x' += i + 1$
9:     loop
10:    $b -= 1$  $b += 1$
11: loop
12:    $c' -= 1$  $c += 1$
13: loop
14:       $c' -= 1$  $c += 1$
15:      loop at most $b$ times
16:       $x' -= 1$  $x += 1$  $d += 1$
17: $i += 1$
18: $\text{zero?} \; i$
19: loop
20:   $x -= i$  $y -= 1$
21: $\text{halt if } y = 0$
**Instrumentation**

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

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

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

<table>
<thead>
<tr>
<th>command</th>
<th>replaced by</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x += 1$</td>
<td>$x += 1$ $c -= 1$</td>
</tr>
<tr>
<td>$x -= 1$</td>
<td>$x -= 1$ $c += 1$</td>
</tr>
</tbody>
</table>

**put $x$, $y$ on budget $c$**

$c + x + y$ constans
Instrumentation

- \( b = A^k(n) \)
- \( c > 0 \)
- \( d = b \cdot c \)
- \( x = y = 0 \) **0-tested** counters

Aim:
simulate \((A^k(n)-1)/2\) zero tests

- introduce fresh counters b, c, d
- instrument increments and decrements:
  
<table>
<thead>
<tr>
<th>command</th>
<th>replaced by</th>
</tr>
</thead>
<tbody>
<tr>
<td>x ++ 1</td>
<td>x ++ 1 c -- 1</td>
</tr>
<tr>
<td>x -- 1</td>
<td>x -- 1 c ++ 1</td>
</tr>
</tbody>
</table>

  put x, y on budget c

  \( c + x + y \) constans

- replace zero? x by

```
P instrumented

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' += i
7: loop at most \( b \) times
8: x -= 1 c' += 1 x' += i + 1
9: loop
10: b -= 1 c' += i + 1
11: loop
12: c' -= i c += 1
13: loop at most \( b \) times
14: x' -= 1 x += 1 d += 1
15: i += 1
16: zero? i
17: loop
18: x -= i y -= 1
19: halt if y = 0

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
```
Instrumentation

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

\begin{verbatim}
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' += 1
7: loop at most b times
8:  x -= 1  x' += i + 1
9: loop
10: b -= 1  b' += i + 1
11: loop
12: c' -= 1  c += 1
13: loop at most b times
14: x' -= 1  x += 1  d += 1
15: i += 1
16: zero? i
17: loop
18: x -= i  y -= 1
19: halt if y = 0
20: halt
\end{verbatim}

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

- introduce fresh counters $b$, $c$, $d$
- \textbf{instrument} increments and decrements:

<table>
<thead>
<tr>
<th>command</th>
<th>replaced by</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x += 1$</td>
<td>$x += 1$ $c -= 1$</td>
</tr>
<tr>
<td>$x -= 1$</td>
<td>$x -= 1$ $c += 1$</td>
</tr>
</tbody>
</table>

- replace \textbf{zero? x by}

\begin{verbatim}
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
\end{verbatim}

- replace \textbf{halt by}

\begin{verbatim}
1: loop
2:  c -= 1  d -= 2
3: ZERO? c
4: halt if \ldots, d = 0.
\end{verbatim}

\textbf{merged} halt of $M$ and $P$
Simulation of a zero test

\[ d = b \cdot (c + x + y) \]

constans

**ZERO? x:**

1: loop
2: \( y -= 1 \quad x += 1 \quad d -= 1 \)
3: loop
4: \( c -= 1 \quad y += 1 \quad d -= 1 \)
5: loop
6: \( y -= 1 \quad c += 1 \quad d -= 1 \)
7: loop
8: \( x -= 1 \quad y += 1 \quad d -= 1 \)
9: \( b -= 2 \)
Simulation of a zero test

\[ d = b \cdot (c + x + y) \]

**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: \( b -= 2 \)

**put x, y on budget c**
Simulation of a zero test

\[ d = b \cdot (c + x + y) \]

**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: \( b \ -= \ 2 \)

put x, y on budget c
Simulation of a zero test

\[ d = b \cdot (c + x + y) \]

**ZERO? x:**

1: **loop**
2: \( y \leftarrow 1 \quad x \rightarrow= 1 \quad d \leftarrow= 1 \)
3: **loop**
4: \( c \leftarrow= 1 \quad y \rightarrow= 1 \quad d \leftarrow= 1 \)
5: **loop**
6: \( y \leftarrow= 1 \quad c \rightarrow= 1 \quad d \leftarrow= 1 \)
7: **loop**
8: \( x \leftarrow= 1 \quad y \rightarrow= 1 \quad d \leftarrow= 1 \)
9: \( b \leftarrow= 2 \)

Put \( x, y \) on budget \( c \)
Simulation of a zero test

\[ d = b \cdot (c + x + y) \]

\textbf{ZERO? x:}

1: \textbf{loop}
2: \hspace{1em} y := 1 \quad x := 1 \quad d := 1
3: \textbf{loop}
4: \hspace{1em} c := 1 \quad y := 1 \quad d := 1
5: \textbf{loop}
6: \hspace{1em} y := 1 \quad c := 1 \quad d := 1
7: \textbf{loop}
8: \hspace{1em} x := 1 \quad y := 1 \quad d := 1
9: \hspace{1em} b := 2

\textbf{put x, y on budget c}
Simulation of a zero test

\[ d = b \cdot (c + x + y) \]

\[ \text{constans} \]

**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

\[ d = b \cdot (c + x + y) \]

\textbf{ZERO? x:}

1: \textbf{loop}
2: \quad y \leftarrow 1 \quad x \leftarrow 1 \quad d \leftarrow 1
3: \textbf{loop}
4: \quad c \leftarrow 1 \quad y \leftarrow 1 \quad d \leftarrow 1
5: \textbf{loop}
6: \quad y \leftarrow 1 \quad c \leftarrow 1 \quad d \leftarrow 1
7: \textbf{loop}
8: \quad x \leftarrow 1 \quad y \leftarrow 1 \quad d \leftarrow 1
9: \quad b \leftarrow 2

\text{d decreases by } \leq 2 \cdot (c + x + y)

\text{b decreases by 2}
Simulation of a zero test

\[ d = b \cdot (c + x + y) \]

**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 \( 2 \cdot (c + x + y) \)
- \( b \) decreases by 2

\( x = 0 \) and \( y = c \) initially and finally

**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

Consider all runs starting in \( I \) (nondeterminism)

**a set \( I \) of initial valuations**

```plaintext
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.
```

the set of all valuations at successful halt
For every fixed $B$:

\[ \text{RATIO}(b, c, d, B) \]

consider all runs (nondeterminism)

\[ \text{P}(b, c, d, b', c', d') \]

One can compute an $A_k$-amplifier with $3k+2$ counters, in time polynomial in $k, n$
For every fixed $B$: 

\[
\text{RATIO}(b, c, d, B) \}
\begin{align*}
& \cdot \ b = B \\
& \cdot \ c > 0 \\
& \cdot \ d = b \cdot c \\
& \cdot \text{all other counters} \ 0
\end{align*}
\]

One can compute an $A_k$-amplifier with $3k+2$ counters, in time polynomial in $k$, $n$
\[ A_k \text{-amplifier} \quad \rightarrow \quad A_k(n) \text{-multiplier} \]

Initial valuation: all counters 0

\begin{align*}
1: & \quad b \mathrel{+}= n \quad d \mathrel{+}= n \quad c \mathrel{+}= 1 \\
2: & \quad \textbf{loop} \quad \text{n-multiplier} \\
3: & \quad d \mathrel{+}= n \quad c \mathrel{+}= 1
\end{align*}

RATIO(b, c, d, n)

\begin{align*}
1: & \quad x \mathrel{+}= 1 \quad y \mathrel{+}= 1 \\
2: & \quad \textbf{loop} \\
3: & \quad x \mathrel{+}= 1 \quad y \mathrel{+}= 1 \\
4: & \quad \textbf{for} \; i := n \; \text{down to} \; 1 \; \text{do} \\
5: & \quad \textbf{loop} \\
6: & \quad x \mathrel{+}= 1 \\
7: & \quad \textbf{loop} \\
8: & \quad x \mathrel{+}= i + 1 \quad z \mathrel{=} i \\
9: & \quad \textbf{loop} \\
10: & \quad x \mathrel{=} n + 1 \quad y \mathrel{=} 1 \\
11: & \quad \text{halt if} \; y = 0.
\end{align*}

RATIO(b', c', d', A_k(n))
Amplifier lifting

- $A_I$-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

- $A_I$-amplifier:

```plaintext
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 $\rightarrow$ $A_{K+I}$-amplifier
Amplifier lifting

- $A_I$-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 $\rightarrow$ $A_{k+1}$-amplifier

\[ A_{i+1}(n) = A_i \circ A_i \circ \ldots \circ A_i(4) = A_i^{n/4}(4) \]
Amplifier lifting

$A_{\mathcal{k}}$-amplifier

$P (b_1, c_1, d_1, b_2, c_2, d_2)$

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

$A_k$-amplifier

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

$\mathcal{M} \ 4$-multiplier

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

$n/4$
Amplifier lifting

$A_k$-amplifier

$P \quad (b_1, c_1, d_1, b_2, c_2, d_2)$

$M \quad 4$-multiplier

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

\[ A_{i+1}(n) = A_i \circ A_i \circ \ldots \circ A_i(4) = A_i^{n/4}(4) \]
Amplifier lifting

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

$A_k$-amplifier

$\mathcal{P}$ (b₁, c₁, d₁, b₂, c₂, d₂)

$\mathcal{M}$ 4-multiplier

$\mathcal{L}$ (b₂, c₂, d₂, b₁, c₁, d₁) identity-amplifier
Amplifier lifting

\[ A_{i+1}(n) = A_i \circ A_i \circ \ldots \circ A_i(4) = A_i^{\frac{n}{4}}(4) \]

\( n/4 \)

**A\_k-amplifier**

\( \mathcal{P} \) (b\_1, c\_1, d\_1, b\_2, c\_2, d\_2)

**M** 4-multiplier

**L** (b\_2, c\_2, d\_2, b\_1, c\_1, d\_1) identity-amplifier

**A\_k+1-amplifier**

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

instrumented
Open questions

• dimension-parametric complexity: $\text{gap } n-4 \ldots 2n+4$

• low dimensions starting from 3

• extensions:
  • data Petri nets
  • pushdown Petri nets
  • branching VASS
Open questions

• dimension-parametric complexity: gap $n-4 \ldots 2n+4$

• low dimensions starting from 3

• extensions:
  • data Petri nets
  • pushdown Petri nets
  • branching VASS

thank you!