The Reachability Problem for Petri Nets is Not Elementary

Wojciech Czerwiński[1], Sławomir Lasota[1], Ranko Lazić[2],
Jérôme Leroux[3] and Filip Mazowiecki[3]

[1]University of Warsaw

[2]University of Warwick

[3]University of Bordeaux

STOC 2019

The Reachability Problem for Petri Nets is Not Elementary

Wojciech Czerwiński[1], Sławomir Lasota[1], Ranko Lazić[2], Jérôme Leroux[3] and Filip Mazowiecki[3]

[1]University of Warsaw

[2]University of Warwick

[3]University of Bordeaux

STOC 2019

Petri nets and the Reachability problem

Petri net (d, T): d – dimension, $T \subseteq \mathbb{Z}^d$

Petri nets and the Reachability problem

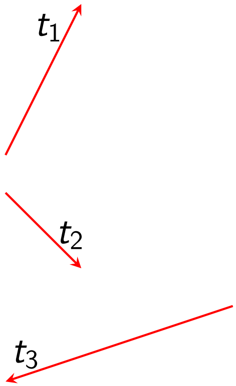
Petri net (d, T): d – dimension, $T \subseteq \mathbb{Z}^d$

Example: $d = 2,\ T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$



Petri nets and the Reachability problem

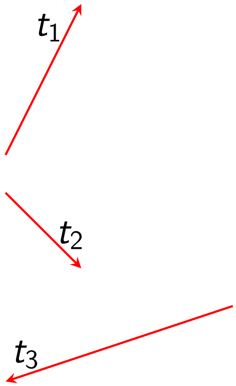
Petri net (d, T): d – dimension, $T \subseteq \mathbb{Z}^d$

Example: $d = 2,\ T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$



Reachability: given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}^d$ is there a walk from $\boldsymbol{a}$ to $\boldsymbol{b}$ within $\mathbb{N}^d$?

Petri nets and the Reachability problem

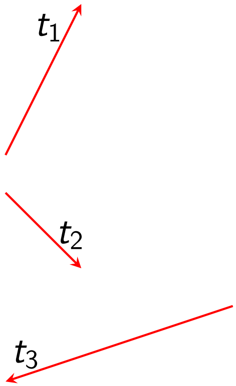
Petri net (d, T): d – dimension, $T \subseteq \mathbb{Z}^d$

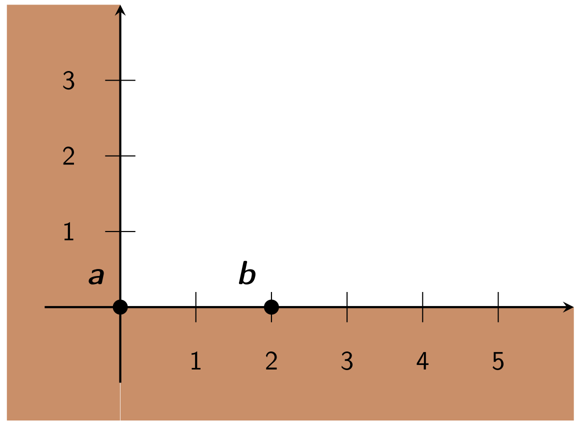
Example: $d = 2$, $T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$





Reachability: given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}^d$ is there a walk from $\boldsymbol{a}$ to $\boldsymbol{b}$ within $\mathbb{N}^d$?

Petri nets and the Reachability problem

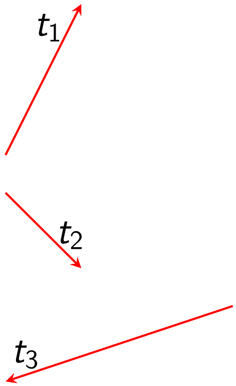
Petri net (d, T): d – dimension, $T \subseteq \mathbb{Z}^d$

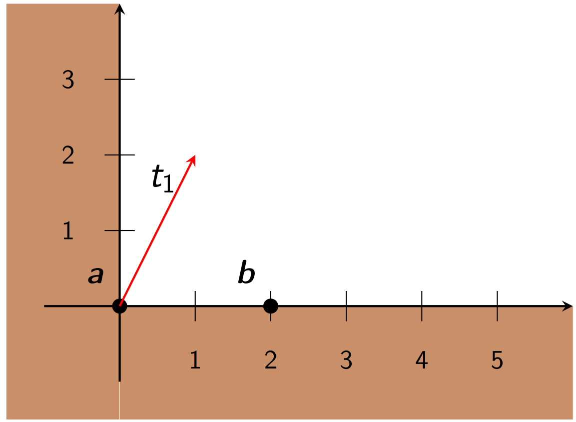
Example: $d = 2$, $T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$





Reachability: given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}^d$ is there a walk from $\boldsymbol{a}$ to $\boldsymbol{b}$ within $\mathbb{N}^d$?

Petri nets and the Reachability problem

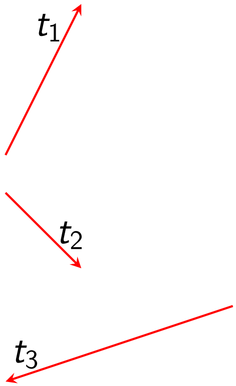
Petri net (d, T): d – dimension, $T \subseteq \mathbb{Z}^d$

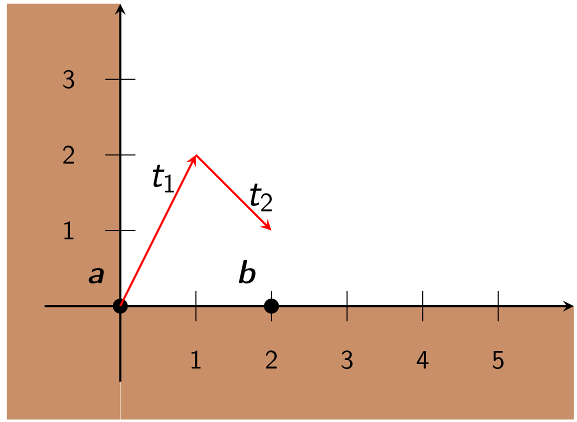
Example: $d = 2,\ T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$





Reachability: given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}^d$ is there a walk from $\boldsymbol{a}$ to $\boldsymbol{b}$ within $\mathbb{N}^d$?

Petri nets and the Reachability problem

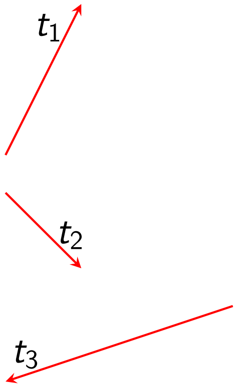
Petri net (d, T): d – dimension, $T \subseteq \mathbb{Z}^d$

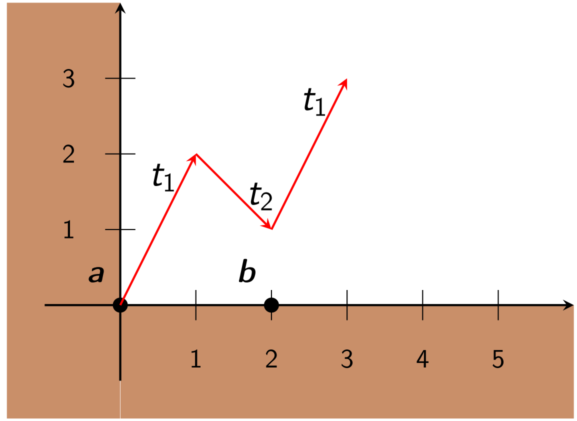
Example: $d = 2$, $T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$





Reachability: given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}^d$ is there a walk from $\boldsymbol{a}$ to $\boldsymbol{b}$ within $\mathbb{N}^d$?

Petri nets and the Reachability problem

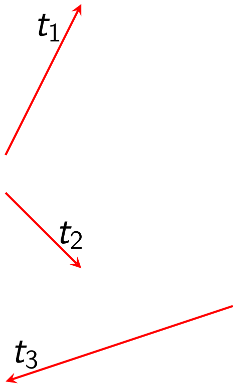
Petri net (d, T): d – dimension, $T \subseteq \mathbb{Z}^d$

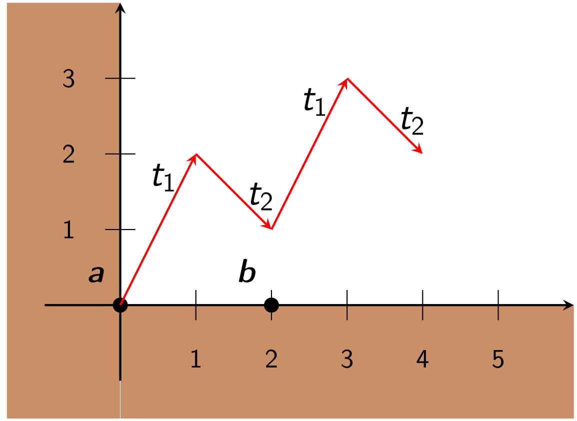
Example: $d = 2$, $T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$





Reachability: given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}^d$ is there a walk from $\boldsymbol{a}$ to $\boldsymbol{b}$ within $\mathbb{N}^d$?

Petri nets and the Reachability problem

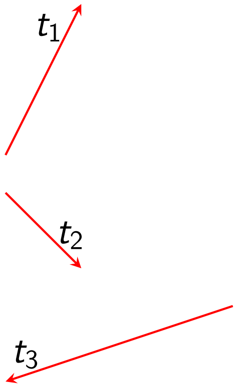
Petri net (d, T): $\quad d$ – dimension, $\quad T \subseteq \mathbb{Z}^d$

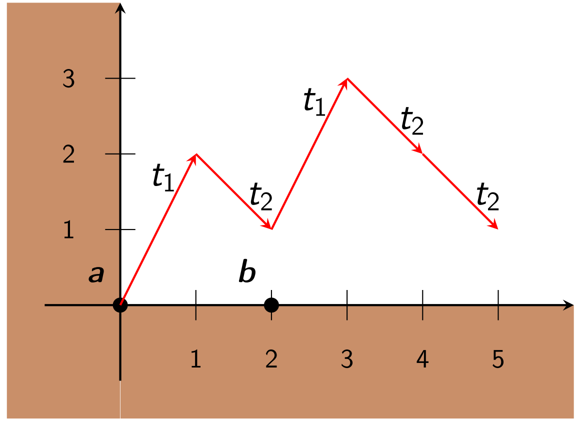
Example: $d = 2,\ T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$





Reachability: given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}^d$ is there a walk from $\boldsymbol{a}$ to $\boldsymbol{b}$ within $\mathbb{N}^d$?

Petri nets and the Reachability problem

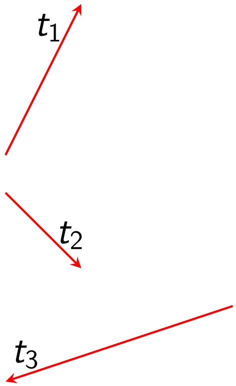
Petri net (d, T): $\quad d$ – dimension, $\quad T \subseteq \mathbb{Z}^d$

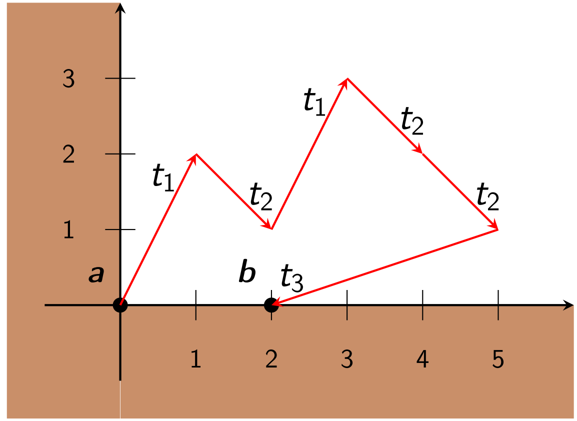
Example: $d = 2,\ T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$





Reachability: given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}^d$ is there a walk from $\boldsymbol{a}$ to $\boldsymbol{b}$ within $\mathbb{N}^d$?

Petri nets and the Reachability problem

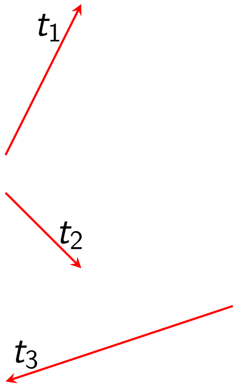
Petri net (d, T): d – dimension, $T \subseteq \mathbb{Z}^d$

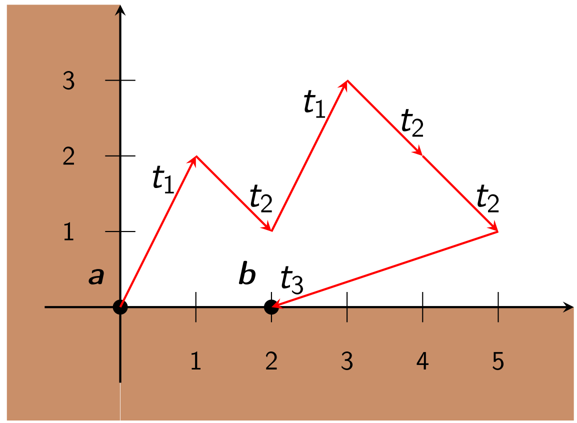
Example: $d = 2,\ T = \{t_1, t_2, t_3\}$

$t_1 = (1, 2)$

$t_2 = (1, -1)$

$t_3 = (-3, -1)$





Reachability: given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{N}^d$ is there a walk from $\boldsymbol{a}$ to $\boldsymbol{b}$ within $\mathbb{N}^d$?

One of the fundamental models and problems in formal verification

Verification of concurrent programs

1: x = True
2: **if** x **then goto** 3 **else goto** 1
3: x = False
4: # critical section
5: **exit**

Verification of concurrent programs

```
1: x = True
2: if x then goto 3 else goto 1
3: x = False
4: # critical section
5: exit
```

Input: n processes in parallel

Output: can two processes be at once in critical section?

Verification of concurrent programs

```
1: x = True
2: if x then goto 3 else goto 1
3: x = False
4: # critical section
5: exit
```

Input: n processes in parallel

Output: can two processes be at once in critical section?

Model with Petri nets (d, T):

- $d = 5$ (one per program command), configurations: vectors in $\mathbb{N}^5$

Verification of concurrent programs

```
1: x = True
2: if x then goto 3 else goto 1
3: x = False
4: # critical section
5: exit
```

Input: n processes in parallel

Output: can two processes be at once in critical section?

Model with Petri nets (d, T):

- $d = 5$ (one per program command), configurations: vectors in $\mathbb{N}^5$
- Transitions: vectors in $\mathbb{Z}^5$ e.g. $(-1, 1, 0, 0, 0)$

Verification of concurrent programs

1: x = True
2: **if** x **then goto** 3 **else goto** 1
3: x = False
4: # critical section
5: **exit**

Input: n processes in parallel

Output: can two processes be at once in critical section?

Model with Petri nets (d, T):

- $d = 5$ (one per program command), configurations: vectors in $\mathbb{N}^5$
- Transitions: vectors in $\mathbb{Z}^5$ e.g. $(-1, 1, 0, 0, 0)$
- Given two configurations $\boldsymbol{a} = (n, 0, 0, 0, 0)$ and $\boldsymbol{b} = (0, 0, 0, 2, 0)$

Verification of concurrent programs

1: x = True
2: **if** x **then goto** 3 **else goto** 1
3: x = False
4: # critical section
5: **exit**

Input: n processes in parallel

Output: can two processes be at once in critical section?

Model with Petri nets (d, T):

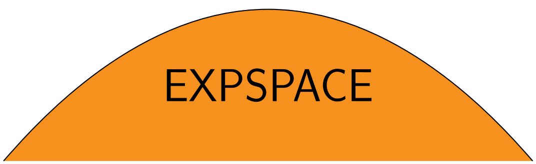
- $d = 5$ (one per program command), configurations: vectors in $\mathbb{N}^5$
- Transitions: vectors in $\mathbb{Z}^5$ e.g. $(-1, 1, 0, 0, 0)$
- Given two configurations $\boldsymbol{a} = (n, 0, 0, 0, 0)$ and $\boldsymbol{b} = (0, 0, 0, 2, 0)$

$\boldsymbol{b}$ is reachable from $\boldsymbol{a}$ $\iff$ two processes at once in critical section

Reachability for Petri nets state of art

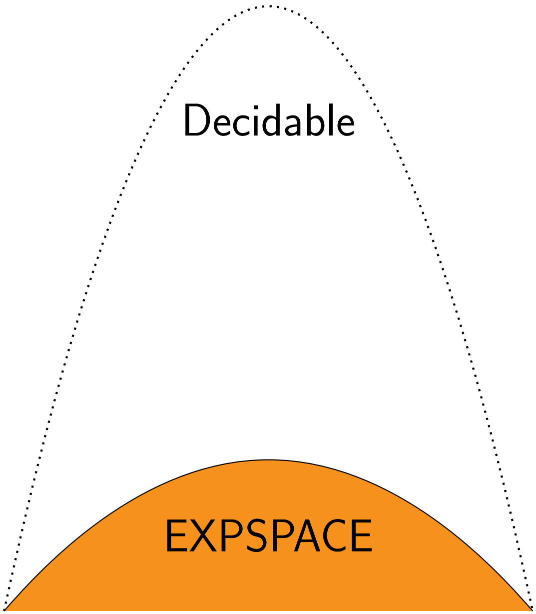
What is the complexity?

1976 EXPSPACE-hard (Lipton)



Reachability for Petri nets state of art

What is the complexity?



1976 — EXPSPACE-hard (Lipton)

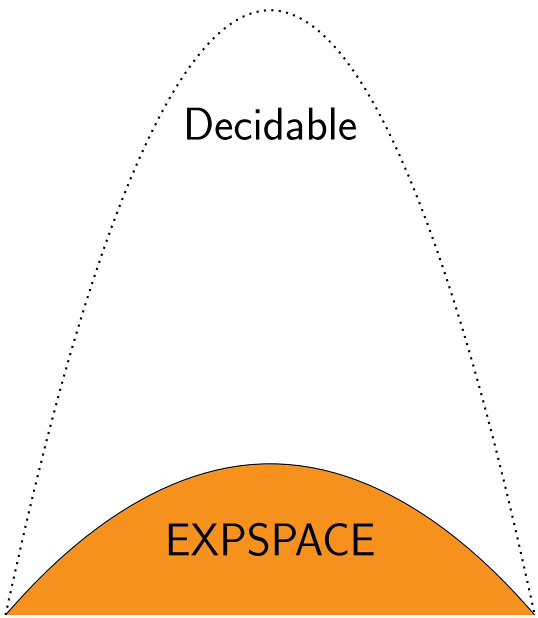
1981 — Decidable (Mayr)

1982 — Decidable (Kosaraju)

1992 — Decidable (Lambert)

Reachability for Petri nets state of art

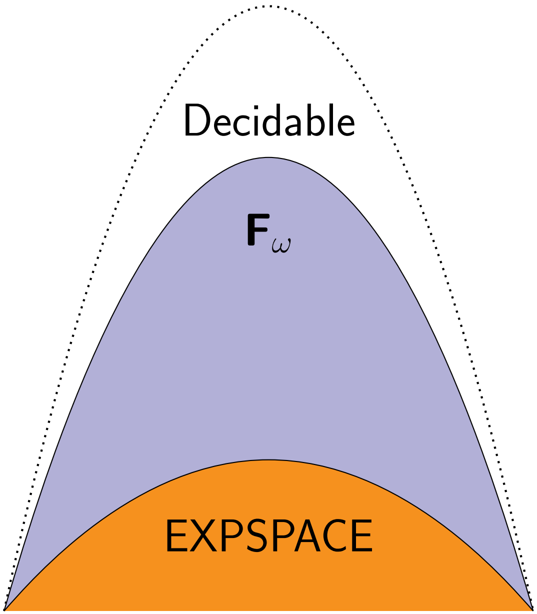
What is the complexity?



1976	EXPSPACE-hard (Lipton)
1981	Decidable (Mayr)
1982	Decidable (Kosaraju)
1992	Decidable (Lambert)
2009	Decidable (Leroux)
2011	Decidable (Leroux)
2012	Decidable (Leroux)

Reachability for Petri nets state of art

What is the complexity?

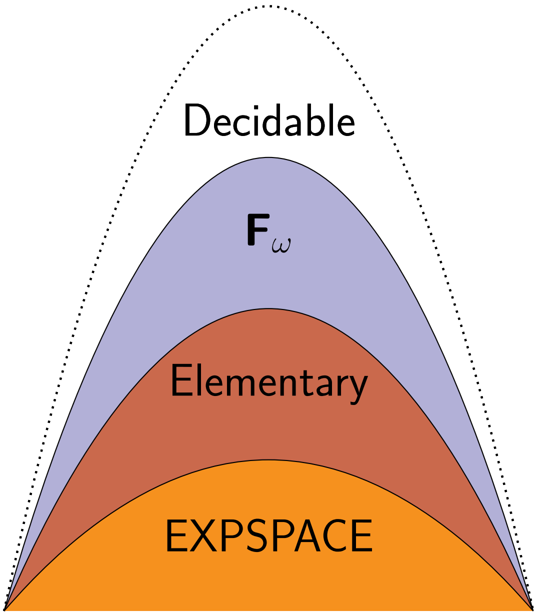


1976	EXPSPACE-hard (Lipton)
1981	Decidable (Mayr)
1982	Decidable (Kosaraju)
1992	Decidable (Lambert)
2009	Decidable (Leroux)
2011	Decidable (Leroux)
2012	Decidable (Leroux)
2015	In $\mathbf{F}_{\omega^3}$ (Leroux and Schmitz)
2019	In $\mathbf{F}_\omega$ (Leroux and Schmitz)

Reachability for Petri nets state of art

What is the complexity?

the first improvement of the lower bound

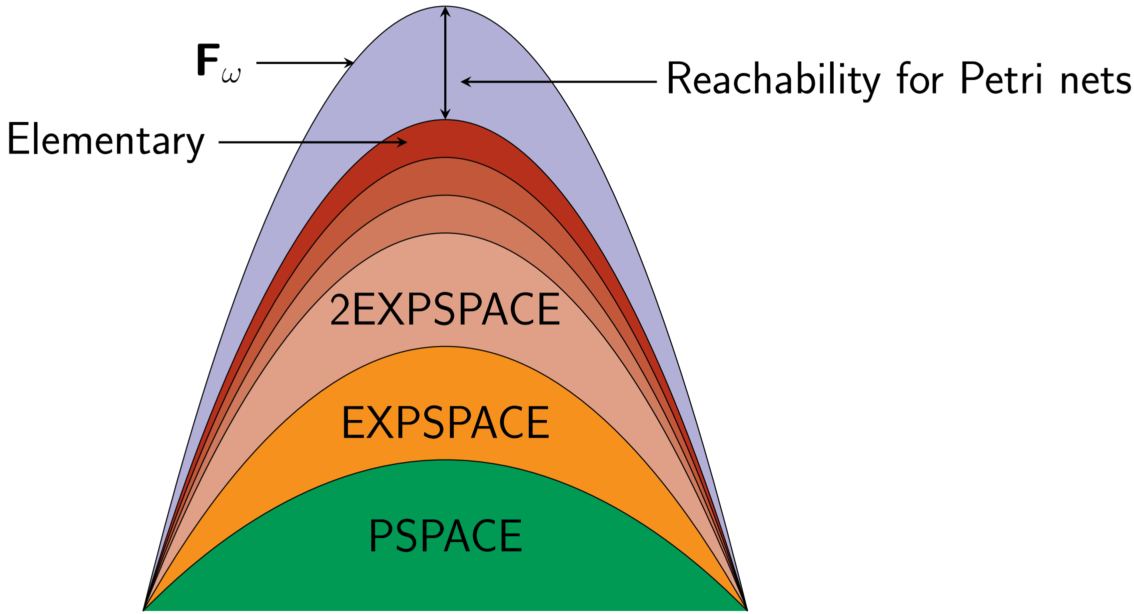


1976	EXPSPACE-hard (Lipton)
1981	Decidable (Mayr)
1982	Decidable (Kosaraju)
1992	Decidable (Lambert)
2009	Decidable (Leroux)
2011	Decidable (Leroux)
2012	Decidable (Leroux)
2015	In $\mathbf{F}_{\omega^3}$ (Leroux and Schmitz)
2019	In $\mathbf{F}_\omega$ (Leroux and Schmitz)
2019	Not elementary (this result)

Implications of the Not elementary result

- Disproves the EXPSPACE-complete conjecture

- Many problems are now not elementary, in: logic, databases, ...



Petri nets as counter programs

$\mathrm{x} \mathrel{+}= \mathrm{m}$	(add m to variable x)
$\mathrm{x} \mathrel{-}= \mathrm{m}$	(subtract m from variable x)
halt	(terminate)
halt if $\mathrm{x}_1, \ldots, \mathrm{x}_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Petri nets as counter programs

$\mathsf{x} \mathrel{+}= \mathsf{m}$	(add *m* to variable x)
$\mathsf{x} \mathrel{-}= \mathsf{m}$	(subtract *m* from variable x)
halt	(terminate)
halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Variables initialised to 0, never become negative. Terminate executing **halt**

Petri nets as counter programs

$\mathsf{x} \mathrel{+}= \mathsf{m}$	(add m to variable x)
$\mathsf{x} \mathrel{-}= \mathsf{m}$	(subtract m from variable x)
halt	(terminate)
halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Variables initialised to 0, never become negative. Terminate executing **halt**

Typically a loop would be **loop until** {condition}

Petri nets as counter programs

$x \mathrel{+}= m$	(add *m* to variable x)
$x \mathrel{-}= m$	(subtract *m* from variable x)
halt	(terminate)
halt if $x_1, \ldots, x_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Variables initialised to 0, never become negative. Terminate executing **halt**

Typically a loop would be **loop until** {condition}

Example

$x_1 \mathrel{+}= 3$
loop
  $x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
until $x_1 = 0$
halt.

Petri nets as counter programs

$x \mathrel{+}= m$	(add m to variable x)
$x \mathrel{-}= m$	(subtract m from variable x)
halt	(terminate)
halt if $x_1, \ldots, x_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Variables initialised to 0, never become negative. Terminate executing **halt**

Typically a loop would be **loop until** $\{$condition$\}$

Example

$x_1 \mathrel{+}= 3$

loop

    $x_1 \mathrel{-}= 1$ $x_2 \mathrel{+}= 2$

until $x_1 = 0$

halt.

x_1	0
x_2	0

Petri nets as counter programs

x += m	(add *m* to variable x)
x −= m	(subtract *m* from variable x)
halt	(terminate)
halt if $x_1, \ldots, x_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Variables initialised to 0, never become negative. Terminate executing **halt**

Typically a loop would be **loop until** {condition}

Example

→ x_1 += 3
loop
  x_1 −= 1 x_2 += 2
until $x_1 = 0$
halt.

x_1	0	3
x_2	0	0

Petri nets as counter programs

x $+=$ m	(add m to variable x)
x $-=$ m	(subtract m from variable x)
halt	(terminate)
halt if $x_1, \ldots, x_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Variables initialised to 0, never become negative. Terminate executing **halt**

Typically a loop would be **loop until** $\{$condition$\}$

Example

$x_1 \mathrel{+}= 3$
loop
→ $x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
until $x_1 = 0$
halt.

x_1	0	3	2
x_2	0	0	2

Petri nets as counter programs

x += m	(add *m* to variable x)
x −= m	(subtract *m* from variable x)
halt	(terminate)
halt if $x_1, \ldots, x_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Variables initialised to 0, never become negative. Terminate executing **halt**

Typically a loop would be **loop until** {condition}

Example

x_1 += 3
loop
→ x_1 −= 1 x_2 += 2
until $x_1 = 0$
halt.

x_1	0	3	2	1
x_2	0	0	2	4

Petri nets as counter programs

x += m	(add *m* to variable x)
x −= m	(subtract *m* from variable x)
halt	(terminate)
halt if $x_1, \ldots, x_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Variables initialised to 0, never become negative. Terminate executing **halt**

Typically a loop would be **loop until** {condition}

Example

x_1 += 3
loop
  x_1 −= 1 x_2 += 2
→ **until** $x_1 = 0$
halt.

x_1	0	3	2	1	0
x_2	0	0	2	4	6

Petri nets as counter programs

$x \mathrel{+}= m$	(add m to variable x)
$x \mathrel{-}= m$	(subtract m from variable x)
halt	(terminate)
halt if $x_1, \ldots, x_l = 0$	(terminate if listed variables are zero)
loop	(to be explained)

Variables initialised to 0, never become negative. Terminate executing **halt**

Typically a loop would be **loop until** {condition}

Example

$x_1 \mathrel{+}= 3$
loop
  $x_1 \mathrel{-}= 1$ $x_2 \mathrel{+}= 2$
until $x_1 = 0$
→ **halt**.

in the end $x_2 = 3 \cdot 2 = 6$
"double the value"

x_1	0	3	2	1	0
x_2	0	0	2	4	6

Petri nets as counter programs

What is our **loop**

Petri nets as counter programs

What is our **loop**

Previous example

$x_1 \mathrel{+}= 3$
loop
$\quad x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
until $x_1 = 0$
halt.
$\{x_2 = 6\}$

Petri nets as counter programs

What is our **loop**

Previous example

$x_1 \mathrel{+}= 3$
loop
  $x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
~~**until** $x_1 = 0$~~
halt.

$\{x_2 \in \{0, 2, 4, 6\}\}$

Petri nets as counter programs

What is our **loop**

Previous example

x_1 += 3
loop
    x_1 −= 1 x_2 += 2
~~**until** $x_1 = 0$~~
halt.

$\{x_2 \in \{0, 2, 4, 6\}\}$

x_1 += 3
loop
    x_1 −= 1 x_2 += 2
halt if $x_1 = 0$.

$\{x_2 = 6\}$

Petri nets as counter programs

What is our **loop**

Previous example

x_1 += 3
loop
  x_1 −= 1 x_2 += 2
~~**until** $x_1 = 0$~~
halt.

$\{x_2 \in \{0, 2, 4, 6\}\}$

x_1 += 3
loop
  x_1 −= 1 x_2 += 2
halt if $x_1 = 0$.

$\{x_2 = 6\}$

loop creates many runs

loop
  x += 1
halt

$\{x \in \mathbb{N}\}$

Petri nets as counter programs

What is our **loop**

Previous example

$x_1 \mathrel{+}= 3$
loop
  $x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
~~**until** $x_1 = 0$~~
halt.

$\{x_2 \in \{0, 2, 4, 6\}\}$

$x_1 \mathrel{+}= 3$
loop
  $x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
halt if $x_1 = 0$.

$\{x_2 = 6\}$

loop creates many runs

loop
  $x \mathrel{+}= 1$
halt

$\{x \in \mathbb{N}\}$

Is there a run executing **halt**? $\iff$ reachability for Petri nets

Programs with normal loops

Additional command: **test** $x = 0$

Programs with normal loops

Additional command: **test** $\mathrm{x} = 0$

- More powerful and more readable

Programs with normal loops

Additional command: **test** $x = 0$

- More powerful and more readable

Previous example: $n \to 2n$

$x_1 \mathrel{+}= n$
loop
  $x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
until $x_1 = 0$
halt.

$x_1 \mathrel{+}= n$
loop
  $x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
halt if $x_1 = 0$.

Programs with normal loops

Additional command: **test** $x = 0$

- More powerful and more readable

Previous example: $n \to 2n$

```
x1 += n
loop
    x1 -= 1    x2 += 2
until x1 = 0
halt.
```

```
x1 += n
loop
    x1 -= 1    x2 += 2
halt if x1 = 0.
```

- But too powerful: reachability becomes undecidable

Programs with normal loops

Additional command: **test** $x = 0$

- More powerful and more readable

Previous example: $n \to 2n$

```
x1 += n
loop
    x1 -= 1    x2 += 2
until x1 = 0
halt.
```

```
x1 += n
loop
    x1 -= 1    x2 += 2
halt if x1 = 0.
```

- But too powerful: reachability becomes undecidable

How much can Petri nets simulate?

High level ideas to prove hardness

Key ingredient: compute a big value in some counter

High level ideas to prove hardness

Key ingredient: compute a big value in some counter

by iteratively reusing some program

- Previous program "double the value"

  take $\mathrm{x}_1, \ldots, \mathrm{x}_n$

High level ideas to prove hardness

Key ingredient: compute a big value in some counter

by iteratively reusing some program

- Previous program "double the value"

take $x_1, \ldots, x_n$

$$
\begin{array}{l}
x_1 \mathrel{+}= 2 \\
\textbf{loop} \\
\quad x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2 \\
\textbf{loop} \\
\quad x_2 \mathrel{-}= 1 \quad x_3 \mathrel{+}= 2 \\
\vdots \\
\textbf{loop} \\
\quad x_{n-1} \mathrel{-}= 1 \quad x_n \mathrel{+}= 2 \\
\textbf{halt if } x_1, x_2, \ldots, x_{n-1} = 0
\end{array}
$$

High level ideas to prove hardness

Key ingredient: compute a big value in some counter

by iteratively reusing some program

- Previous program "double the value"

take $x_1, \ldots, x_n$

$x_1 \mathrel{+}= 2$
loop
$\quad x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
loop
$\quad x_2 \mathrel{-}= 1 \quad x_3 \mathrel{+}= 2$
$\vdots$
loop
$\quad x_{n-1} \mathrel{-}= 1 \quad x_n \mathrel{+}= 2$
halt if $x_1, x_2, \ldots, x_{n-1} = 0$

High level ideas to prove hardness

Key ingredient: compute a big value in some counter

by iteratively reusing some program

- Previous program "double the value"

take $x_1, \ldots, x_n$

$x_1 \mathrel{+}= 2$
loop
$\quad x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
loop
$\quad x_2 \mathrel{-}= 1 \quad x_3 \mathrel{+}= 2$
$\vdots$
loop
$\quad x_{n-1} \mathrel{-}= 1 \quad x_n \mathrel{+}= 2$
halt if $x_1, x_2, \ldots, x_{n-1} = 0$

High level ideas to prove hardness

Key ingredient: compute a big value in some counter

by iteratively reusing some program

- Previous program "double the value"

take $x_1, \ldots, x_n$

$x_1 \mathrel{+}= 2$
loop
 $x_1 \mathrel{-}= 1 \quad x_2 \mathrel{+}= 2$
loop
 $x_2 \mathrel{-}= 1 \quad x_3 \mathrel{+}= 2$
$\vdots$
loop
 $x_{n-1} \mathrel{-}= 1 \quad x_n \mathrel{+}= 2$
halt if $x_1, x_2, \ldots, x_{n-1} = 0$

$x_{i+1} = 2 \cdot x_i, \quad \text{so } x_n = 2^n$

Showing hardness

What programs to iterate?

- Lipton's program is "squaring": $x_{i+1} = (x_i)^2$, so $x_n = 2^{2^n}$
  (this gives the EXPSPACE lower bound from 1976)

- We managed to do "factorial": $x_{i+1} = (x_i)!$, so $x_n \geq \left. 2^{\cdot^{\cdot^{\cdot^{2}}}} \right\} n$ times.
  (this gives the not elementary lower bound in our paper)

Key idea

For the program $k \rightarrow k!$

Key idea

For the program $k \to k!$

$\mathrm{i} \mathrel{+}= k$

loop

  $\mathrm{x} \mathrel{+}= 1 \quad \mathrm{y} \mathrel{+}= 1 \quad \mathrm{z} \mathrel{+}= 1$

loop

  loop

    $\mathrm{x} \mathrel{-}= \mathrm{i} \quad \mathrm{x}' \mathrel{+}= \mathrm{i} + 1$

  loop

    $\mathrm{x}' \mathrel{-}= 1 \quad \mathrm{x} \mathrel{+}= 1$

  $\mathrm{i} \mathrel{-}= 1$

loop

  $\mathrm{x} \mathrel{-}= (k+1) \quad \mathrm{y} \mathrel{-}= 1$

halt if $\mathrm{y}, \mathrm{i} = 0$

Key idea

For the program $k \to k!$

$\mathrm{i} \mathrel{+}= k$
loop
  $\mathrm{x} \mathrel{+}= 1 \quad \mathrm{y} \mathrel{+}= 1 \quad \boxed{\mathrm{z} \mathrel{+}= 1}$
loop
  loop
    $\mathrm{x} \mathrel{-}= \mathrm{i} \quad \mathrm{x}' \mathrel{+}= \mathrm{i} + 1$
  loop
    $\mathrm{x}' \mathrel{-}= 1 \quad \mathrm{x} \mathrel{+}= 1$
  $\mathrm{i} \mathrel{-}= 1$
loop
  $\mathrm{x} \mathrel{-}= (k+1) \quad \mathrm{y} \mathrel{-}= 1$
halt if $\mathrm{y}, \mathrm{i} = 0$

$\boxed{\{z = k!\}}$

Key idea

For the program $k \to k!$

$\mathrm{i} \mathrel{+}= k$ i goes from k to 0 in the main loop

```
loop
    x += 1    y += 1    z += 1
loop
    loop
        x −= i    x′ += i + 1
    loop
        x′ −= 1    x += 1
    i −= 1
loop
    x −= (k + 1)    y −= 1
halt if y, i = 0
```

$\{z = k!\}$

Key idea

For the program $k \to k!$

$\text{i} \mathrel{+}= k$

loop

$\text{x} \mathrel{+}= 1 \quad \text{y} \mathrel{+}= 1 \quad \text{z} \mathrel{+}= 1$

loop

loop

$\text{x} \mathrel{-}= \text{i} \quad \text{x}' \mathrel{+}= \text{i} + 1$

loop

$\text{x}' \mathrel{-}= 1 \quad \text{x} \mathrel{+}= 1$

$\text{i} \mathrel{-}= 1$

loop

$\text{x} \mathrel{-}= (k+1) \quad \text{y} \mathrel{-}= 1$

halt if $\text{y}, \text{i} = 0$

$\{z = k!\}$

i goes from k to 0 in the main loop

$\text{x}, \text{y}, \text{z}$ initialised to any c

every time x multiplied by at most $\frac{\text{i}+1}{\text{i}}$

Key idea

For the program $k \to k!$

$\mathsf{i} \mathrel{+}= k$

loop

$\mathsf{x} \mathrel{+}= 1 \quad \mathsf{y} \mathrel{+}= 1 \quad \mathsf{z} \mathrel{+}= 1$

loop

loop

$\mathsf{x} \mathrel{-}= \mathsf{i} \quad \mathsf{x}' \mathrel{+}= \mathsf{i} + 1$

loop

$\mathsf{x}' \mathrel{-}= 1 \quad \mathsf{x} \mathrel{+}= 1$

$\mathsf{i} \mathrel{-}= 1$

loop

$\mathsf{x} \mathrel{-}= (k+1) \quad \mathsf{y} \mathrel{-}= 1$

halt if $\mathsf{y}, \mathsf{i} = 0$

$\{z = k!\}$

i goes from k to 0 in the main loop

x, y, z initialised to any c

every time x multiplied by at most $\frac{\mathsf{i}+1}{\mathsf{i}}$

In the end $\mathsf{x} \leq c \cdot \prod_{i=1}^{k} \frac{i+1}{i} = c \cdot (k+1)$

$\frac{k+1}{k} \cdot \frac{k}{k-1} \cdot \frac{k-1}{k-2} \cdots \frac{3}{2} \cdot \frac{2}{1} = k+1$

Key idea

For the program $k \to k!$

$\mathrm{i} \mathrel{+}= k$

loop

$\mathrm{x} \mathrel{+}= 1 \quad \mathrm{y} \mathrel{+}= 1 \quad \mathrm{z} \mathrel{+}= 1$

loop

loop

$\mathrm{x} \mathrel{-}= \mathrm{i} \quad \mathrm{x}' \mathrel{+}= \mathrm{i}+1$

loop

$\mathrm{x}' \mathrel{-}= 1 \quad \mathrm{x} \mathrel{+}= 1$

$\mathrm{i} \mathrel{-}= 1$

loop

$\mathrm{x} \mathrel{-}= (k+1) \quad \mathrm{y} \mathrel{-}= 1$

halt if $\mathrm{y}, \mathrm{i} = 0$

$\{z = k!\}$

i goes from k to 0 in the main loop

$\mathrm{x}, \mathrm{y}, \mathrm{z}$ initialised to any c

every time x multiplied by at most $\frac{\mathrm{i}+1}{\mathrm{i}}$

In the end $\mathrm{x} \leq c \cdot \prod_{i=1}^{k} \frac{i+1}{i} = c \cdot (k+1)$

But we need $\mathrm{x} \geq \mathrm{y} \cdot (k+1) = c \cdot (k+1)$

Key idea

For the program $k \to k!$

$\mathrm{i} \mathrel{+}= k$
loop
  $\mathrm{x} \mathrel{+}= 1$ $\mathrm{y} \mathrel{+}= 1$ $\mathrm{z} \mathrel{+}= 1$
loop
  loop
    $\mathrm{x} \mathrel{-}= \mathrm{i}$ $\mathrm{x}' \mathrel{+}= \mathrm{i}+1$
  loop
    $\mathrm{x}' \mathrel{-}= 1$ $\mathrm{x} \mathrel{+}= 1$
  $\mathrm{i} \mathrel{-}= 1$
loop
  $\mathrm{x} \mathrel{-}= (k+1)$ $\mathrm{y} \mathrel{-}= 1$
halt if $\mathrm{y}, \mathrm{i} = 0$
$\{z = k!\}$

i goes from k to 0 in the main loop

$\mathrm{x}, \mathrm{y}, \mathrm{z}$ initialised to any c

so $k \mid \mathrm{x}, \quad k-1 \mid \mathrm{x}, \ldots$

every time x multiplied by at most $\frac{\mathrm{i}+1}{\mathrm{i}}$

In the end $\mathrm{x} \leq c \cdot \prod_{i=1}^{k} \frac{i+1}{i} = c \cdot (k+1)$

But we need $\mathrm{x} \geq \mathrm{y} \cdot (k+1) = c \cdot (k+1)$

Conclusion

- Several applications and corollaries
  - reachability is harder than we expected
  - many problems are not elementary (FO2 on data words)

Conclusion

- Several applications and corollaries
  - reachability is harder than we expected
  - many problems are not elementary (FO2 on data words)

- The proof is short and self contained

Conclusion

- Several applications and corollaries
  - reachability is harder than we expected
  - many problems are not elementary (FO2 on data words)

- The proof is short and self contained

- The complexity is between Tower ($\mathbf{F}_3$) and Ackermann ($\mathbf{F}_\omega$)

Conclusion

- Several applications and corollaries
  - reachability is harder than we expected
  - many problems are not elementary (FO2 on data words)
- The proof is short and self contained
- The complexity is between Tower ($\mathbf{F}_3$) and Ackermann ($\mathbf{F}_\omega$)
- This originated from studying 1-Pushdown-VASS

Conclusion

- Several applications and corollaries
  - reachability is harder than we expected
  - many problems are not elementary (FO2 on data words)

- The proof is short and self contained

- The complexity is between Tower ($\mathbf{F}_3$) and Ackermann ($\mathbf{F}_\omega$)

- This originated from studying 1-Pushdown-VASS

So maybe it's good to study restrictions of generalizations of etc...

(my current favourite: BOBRVASS)