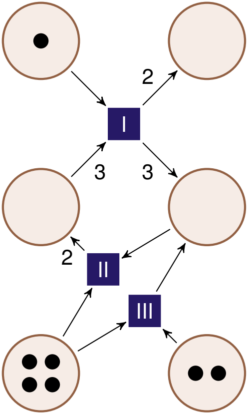
# How hard is the reachability problem for Petri nets?

Ismaël Jecker

# Petri net

A Petri net is a directed graph composed of

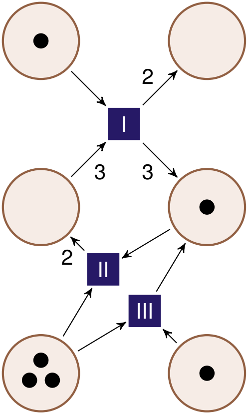
- **Places**, that contain tokens
- **Transitions**, that can transfer tokens



# Petri net

A Petri net is a directed graph composed of

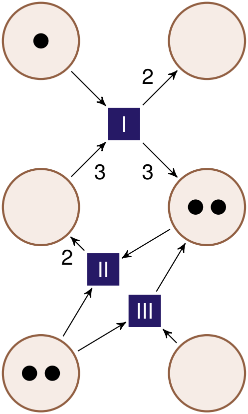
- **Places**, that contain tokens
- **Transitions**, that can transfer tokens



# Petri net

A Petri net is a directed graph composed of

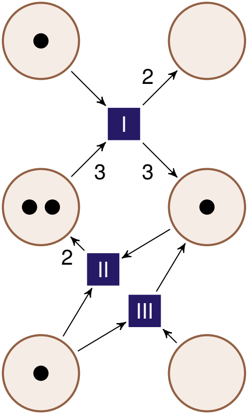
- **Places**, that contain tokens
- **Transitions**, that can transfer tokens



# Petri net

A Petri net is a directed graph composed of

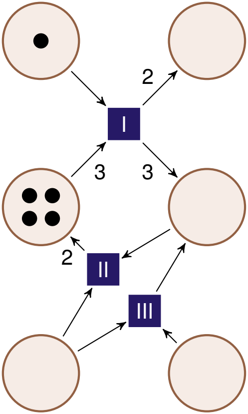
- **Places**, that contain tokens
- **Transitions**, that can transfer tokens



# Petri net

A Petri net is a directed graph composed of

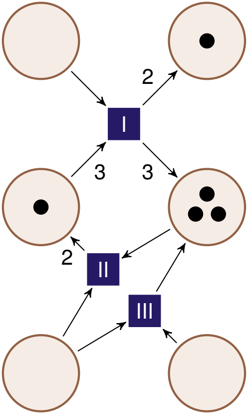
- **Places**, that contain tokens
- **Transitions**, that can transfer tokens



# Petri net

A Petri net is a directed graph composed of

- **Places**, that contain tokens
- **Transitions**, that can transfer tokens




They are used to model:

- distributed systems
- network protocols
- chemical processes

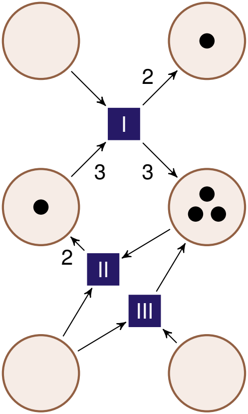
# Petri net

A Petri net is a directed graph composed of

- **Places**, that contain tokens
- **Transitions**, that can transfer tokens

They are used to model:

- distributed systems
- network protocols
- chemical processes



# Petri net

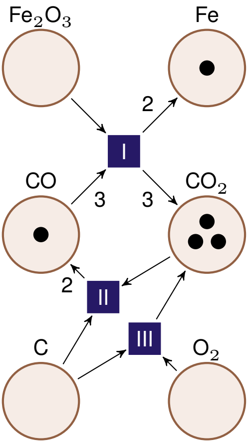
A Petri net is a directed graph composed of

- **Places**, that contain tokens
- **Transitions**, that can transfer tokens

They are used to model:

- distributed systems
- network protocols
- chemical processes

| | |
|---|---|
| I | $Fe_2O_3 + 3CO \rightarrow 2Fe + 3CO_2$ |
| II | $CO_2 + C \rightarrow 2CO$ |
| III | $O_2 + C \rightarrow CO_2$ |



| REACHABILITY | **Input**: Two configurations $s$, $t$ of a Petri Net $P$<br>**Question**: Starting from $s$, can $P$ reach $t$? |
|---|---|

Reachability

**Input**: Two configurations s, t of a Petri Net P
**Question**: Starting from s, can P reach t?

ExpSpace-hard

1976: Lipton

**REACHABILITY**

**Input**: Two configurations s, t of a Petri Net P
**Question**: Starting from s, can P reach t?

**Decidable**

1981: Mayr
1982: Kosaraju
1992: Lambert

**EXPSPACE-hard**

1976: Lipton

| REACHABILITY | **Input**: Two configurations s, t of a Petri Net P<br>**Question**: Starting from s, can P reach t? |
| --- | --- |

## Decidable

1981: Mayr
1982: Kosaraju
1992: Lambert

## in ACKERMANN

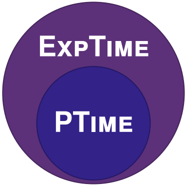
2015: Leroux & Schmitz
2019: Leroux & Schmitz

## EXPSPACE-hard

1976: Lipton

**PTime**: can be solved in **polynomial** time

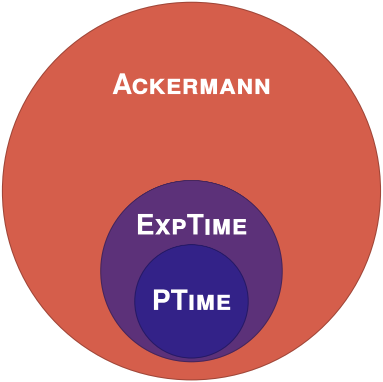
**ExpTime**: can be solved in **exponential** time



**PTime**: can be solved in **polynomial** time

**ExpTime**: can be solved in **exponential** time

**Ackermann**: can be solved with a **HUGE** amount of time



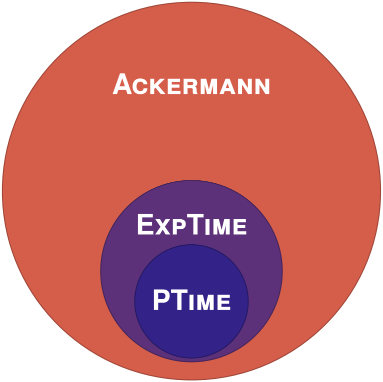
**PTime**: can be solved in **polynomial** time

**ExpTime**: can be solved in **exponential** time

**Ackermann**: can be solved with a **HUGE** amount of time

**Fast growing hierarchy:**

$$
\begin{aligned}
f_{i+1}(n) &= \Big(\underbrace{f_i \circ \ldots \circ f_i}_{n \text{ times}}\Big)(1) \\
f_1(n) &= 2n
\end{aligned}
$$




$f_1(1) = 2$ $f_1(2) = 4$ $f_1(3) = 6$ $f_1(4) = 8$ $f_1(5) = 10$

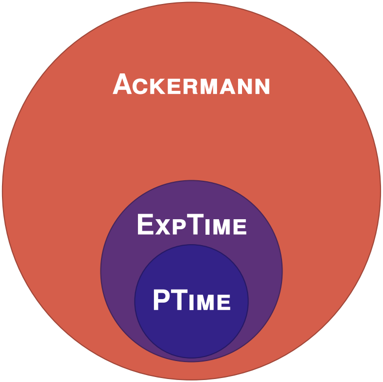
**PTime**: can be solved in **polynomial** time

**ExpTime**: can be solved in **exponential** time

**Ackermann**: can be solved with a **HUGE** amount of time

**Fast growing hierarchy:**

$$
\begin{aligned}
f_{i+1}(n) &= \Big(\underbrace{f_i \circ \ldots \circ f_i}_{n \text{ times}}\Big)(1) \\
f_1(n) &= 2n \\
f_2(n) &= 2^n
\end{aligned}
$$




$f_2(1) = 2$ $\quad f_2(2) = 4$ $\quad f_2(3) = 8$ $\quad f_2(4) = 16$ $\quad f_2(5) = 32$

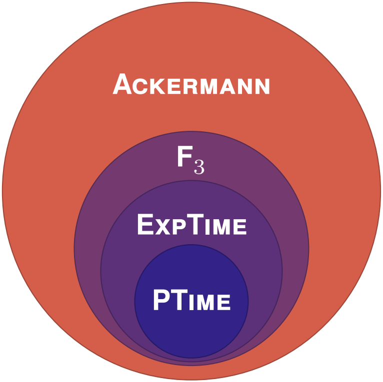
**PTIME**: can be solved in **polynomial** time

**EXPTIME**: can be solved in **exponential** time

**ACKERMANN**: can be solved with a **HUGE** amount of time

**Fast growing hierarchy:**

$$
\begin{aligned}
f_{i+1}(n) &= \Big(\underbrace{f_i \circ \ldots \circ f_i}_{n \text{ times}}\Big)(1) \\
f_1(n) &= 2n \\
f_2(n) &= 2^n \\
f_3(n) &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n \text{ times}}
\end{aligned}
$$




$f_3(1) = 2 \qquad f_3(2) = 4 \qquad f_3(3) = 16 \qquad f_3(4) = 65536 \qquad f_3(5) \simeq 2 \cdot 10^{19728}$

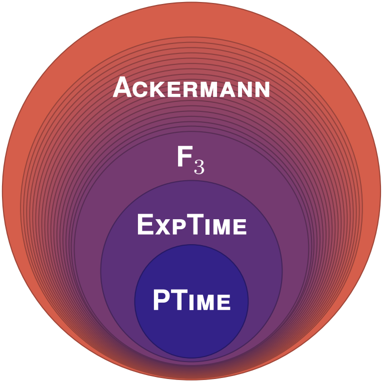
**PTime**: can be solved in **polynomial** time

**ExpTime**: can be solved in **exponential** time

**Ackermann**: can be solved with a **HUGE** amount of time

**Fast growing hierarchy:**

$$\begin{aligned} f_{i+1}(n) &= \Big(\underbrace{f_i \circ \ldots \circ f_i}_{n \text{ times}}\Big)(1) \\ f_1(n) &= 2n \\ f_2(n) &= 2^n \\ f_3(n) &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n \text{ times}} \\ &\vdots \end{aligned}$$



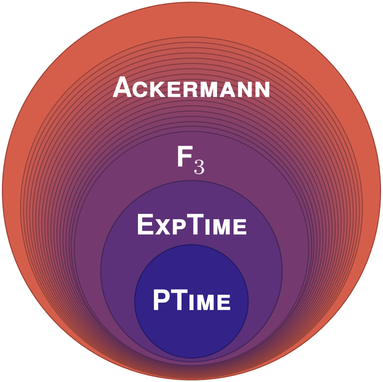
**PTime**: can be solved in **polynomial** time

**ExpTime**: can be solved in **exponential** time

**Ackermann**: can be solved with a **HUGE** amount of time

**Fast growing hierarchy:**

$$
\begin{aligned}
f_{i+1}(n) &= \Big(\underbrace{f_i \circ \ldots \circ f_i}_{n \text{ times}}\Big)(1) \\
f_1(n) &= 2n \\
f_2(n) &= 2^n \\
f_3(n) &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n \text{ times}} \\
&\vdots \\
\mathcal{A}(n) &= f_n(n)
\end{aligned}
$$



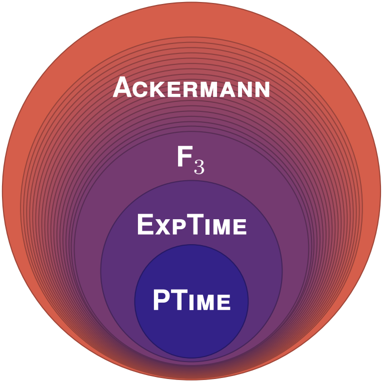
**PTime**: can be solved in **polynomial** time

**ExpTime**: can be solved in **exponential** time

**Ackermann**: can be solved in $\mathcal{A}(n)$ time

**Fast growing hierarchy:**

$$
\begin{aligned}
f_{i+1}(n) &= \Big(\underbrace{f_i \circ \ldots \circ f_i}_{n \text{ times}}\Big)(1) \\
f_1(n) &= 2n \\
f_2(n) &= 2^n \\
f_3(n) &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n \text{ times}} \\
&\vdots \\
\mathcal{A}(n) &= f_n(n)
\end{aligned}
$$



**REACHABILITY**

**Input**: Two configurations s, t of a Petri Net P

**Question**: Starting from s, can P reach t?

**Decidable**

1981: Mayr
1982: Kosaraju
1992: Lambert

**in ACKERMANN**

2015: Leroux & Schmitz
2019: Leroux & Schmitz

**EXPSPACE-hard**

1976: Lipton

**REACHABILITY**

**Input**: Two configurations s, t of a Petri Net P
**Question**: Starting from s, can P reach t?

## Decidable

1981: Mayr
1982: Kosaraju
1992: Lambert

## in ACKERMANN

2015: Leroux & Schmitz
2019: Leroux & Schmitz

## EXPSPACE-hard

1976: Lipton

## $\mathbf{F}_3$-hard

2019: Czerwiński, Lasota et al.

**REACHABILITY**

**Input**: Two configurations s, t of a Petri Net P
**Question**: Starting from s, can P reach t?

**Decidable**

1981: Mayr
1982: Kosaraju
1992: Lambert

**in ACKERMANN**

2015: Leroux & Schmitz
2019: Leroux & Schmitz

**EXPSPACE-hard**

1976: Lipton

**$\mathbf{F}_3$-hard**

2019: Czerwiński, Lasota et al.

**ACKERMANN-hard**

2021: Czerwiński & Orlikowski / Leroux
2022: Lasota
2023: Czerwiński, J., Lasota et al.

**REACHABILITY**

**Input**: Two configurations s, t of a Petri Net P
**Question**: Starting from s, can P reach t?

**Decidable**

1981: Mayr
1982: Kosaraju
1992: Lambert

**in ACKERMANN**

2015: Leroux & Schmitz
2019: Leroux & Schmitz

**EXPSPACE-hard**

1976: Lipton

**$\mathbf{F}_3$-hard**

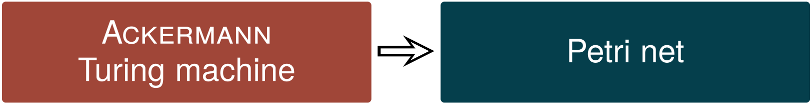
2019: Czerwiński, Lasota et al.

**ACKERMANN-hard**

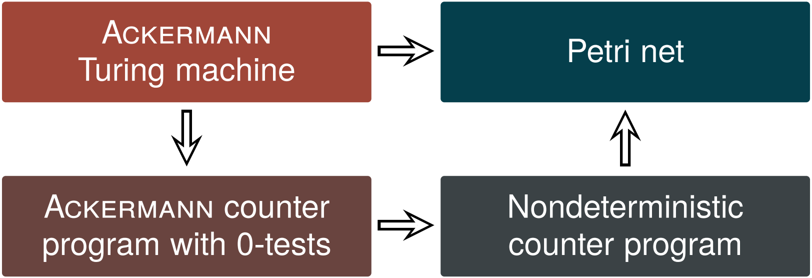
2021: Czerwiński & Orlikowski / Leroux
2022: Lasota
2023: Czerwiński, J., Lasota et al.

**ACKERMANN-complete**

## Reachability for Petri Nets is Ackermann-hard



# Reachability for Petri Nets is Ackermann-hard



# Counter program with 0-tests (CP0)

```
double(x,y)
 1  x -= 1
 2  y += 2
 3  if x == 0 then goto 4 else goto 1
 4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

name list of variables storing **non-negative** integers

list of instructions

```
double(x,y)
1  x -= 1
2  y += 2
3  if x == 0 then goto 4 else goto 1
4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

## Counter program with 0-tests (CP0)

```
double(x,y)
 1  x -= 1
 2  y += 2
 3  if x == 0 then goto 4 else goto 1
 4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                       (3,0)
 1  x -= 1
 2  y += 2
 3  if x == 0 then goto 4 else goto 1
 4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                          (3,0)
1  x -= 1                            (3,0)
2  y += 2
3  if x == 0 then goto 4 else goto 1
4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                  (3,0)
1  x -= 1
2  y += 2                                    (2,0)
3  if x == 0 then goto 4 else goto 1
4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                        (3,0)
1  x -= 1
2  y += 2
3  if x == 0 then goto 4 else goto 1              (2,2)
4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                   (3,0)
1  x -= 1                                     (2,2)
2  y += 2
3  if x == 0 then goto 4 else goto 1
4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                   (3,0)
 1  x -= 1
 2  y += 2                                    (1,2)
 3  if x == 0 then goto 4 else goto 1
 4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                    (3,0)
1  x -= 1
2  y += 2
3  if x == 0 then goto 4 else goto 1           (1,4)
4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                   (3,0)
1  x -= 1                                     (1,4)
2  y += 2
3  if x == 0 then goto 4 else goto 1
4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                   (3,0)
 1  x -= 1
 2  y += 2                                    (0,4)
 3  if x == 0 then goto 4 else goto 1
 4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                  (3,0)
1  x -= 1
2  y += 2
3  if x == 0 then goto 4 else goto 1         (0,6)
4  end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                      (3,0)
1  x -= 1
2  y += 2
3  if x == 0 then goto 4 else goto 1
4  end                                           (0,6)
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                  (3,0)
1  x -= 1
2  y += 2
3  if x == 0 then goto 4 else goto 1
4  end                                       (0,6)
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement Counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Zero-test `if x == 0 then goto i else goto j`

# Counter program with 0-tests (CP0)

```
double(x,y)                                  (3,0)
1  x -= 1
2  y += 2
3  if x == 0 then goto 4 else goto 1
4  end                                       (0,6)
```

Indirectly, these instructions can be used to define:

- Multiplication `x *= n`
- Integer division `x /= n`

# Counter program with 0-tests (CP0)

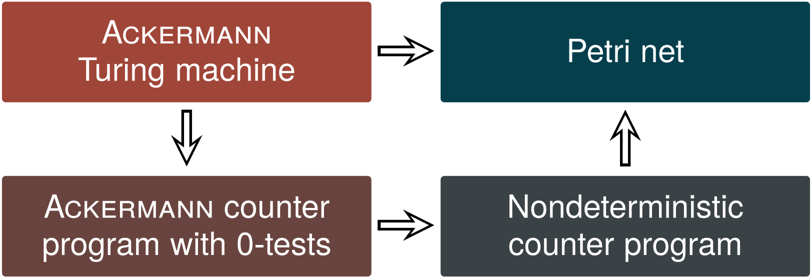
```
double(x,y)                                    (3,0)
 1  x -= 1
 2  y += 2
 3  if x == 0 then goto 4 else goto 1
 4  end                                        (0,6)
```

Indirectly, these instructions can be used to define:

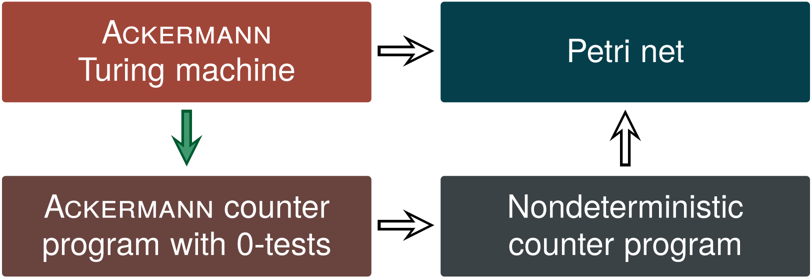
- Multiplication `x *= n`
- Integer division `x /= n`

This is sufficient to simulate runs of Turing Machines! [1967. Minsky]

## Reachability for Petri Nets is Ackermann-hard



# Reachability for Petri Nets is Ackermann-hard



# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)
 1    x -= 1
 2    y += 2
 3    goto 4 or goto 1
 4    end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

name — list of variables

list of instructions

```
erraticDouble(x,y)
1   x -= 1
2   y += 2
3   goto 4 or goto 1
4   end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)
 1    x -= 1
 2    y += 2
 3    goto 4 or goto 1
 4    end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)                    (3,0)
1   x -= 1
2   y += 2
3   goto 4 or goto 1
4   end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)                       (3,0)
1    x -= 1                              (3,0)
2    y += 2
3    goto 4 or goto 1
4    end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)                    (3,0)
1    x -= 1
2    y += 2                           (2,0)
3    goto 4 or goto 1
4    end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)                (3,0)
1   x -= 1
2   y += 2
3   goto 4 or goto 1              (2,2)
4   end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)                  (3,0)
    x -= 1                          (2,2)
    y += 2
    goto 4 or goto 1
    end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)              (3,0)
    x -= 1
    y += 2                      (1,2)
    goto 4 or goto 1
    end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)                    (3,0)
1    x -= 1
2    y += 2
3    goto 4 or goto 1                 (1,4)
4    end
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)                    (3,0)
1   x -= 1
2   y += 2
3   goto 4 or goto 1
4   end                               (1,4)
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

# Nondeterministic counter program (NCP)

```
erraticDouble(x,y)                    (3,0)
1    x -= 1
2    y += 2
3    goto 4 or goto 1
4    end                              (1,4)
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instruction**:

- Jump nondeterministically `goto i or goto j`

Nondeterministic counter program ⇒ Petri Net

```
erraticDouble(x,y)
1    x -= 1                    (3,0)
2    y += 2
3    goto 4 or goto 1
4    end
```

→ ?

Nondeterministic counter program ⇒ Petri Net

```
erraticDouble(x,y)
1   x -= 1                (3,0)
2   y += 2
3   goto 4 or goto 1
4   end
```

1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

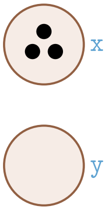
Nondeterministic counter program ⇒ Petri Net

```
erraticDouble(x,y)
1    x -= 1              (3,0)
2    y += 2
3    goto 4 or goto 1
4    end
```

1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

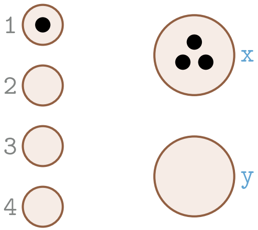
```
erraticDouble(x,y)
1    x -= 1              (3,0)
2    y += 2
3    goto 4 or goto 1
4    end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

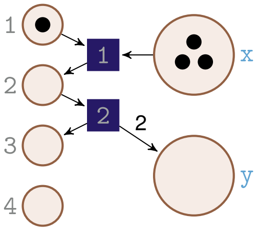
```
erraticDouble(x,y)
x -= 1                    (3,0)
y += 2
goto 4 or goto 1
end
```




1. One **place** per variable (tokens simulate their contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

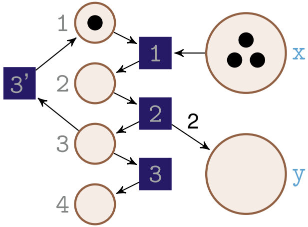
```
erraticDouble(x,y)
1   x -= 1                 (3,0)
2   y += 2
3   goto 4 or goto 1
4   end
```




1. One **place** per variable (tokens simulate their contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

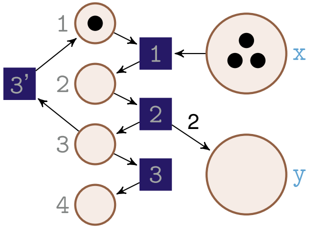
```
erraticDouble(x,y)
    x -= 1                  (3,0)
    y += 2
    goto 4 or goto 1
    end
```




1. One **place** per variable (tokens simulate their contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

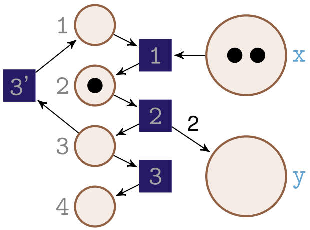
```
erraticDouble(x,y)
1   x -= 1                  (3,0)
2   y += 2
3   goto 4 or goto 1
4   end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

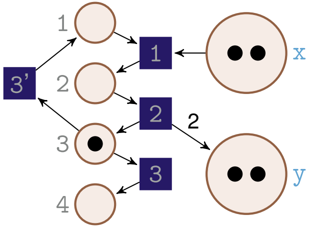
```
erraticDouble(x,y)
1    x -= 1
2    y += 2                (2,0)
3    goto 4 or goto 1
4    end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

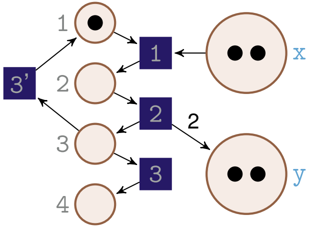
```
erraticDouble(x,y)
    x -= 1
    y += 2
    goto 4 or goto 1   (2,2)
    end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

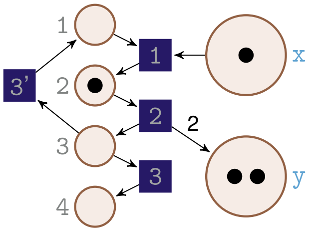
```
erraticDouble(x,y)
    x -= 1                (2,2)
    y += 2
    goto 4 or goto 1
    end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

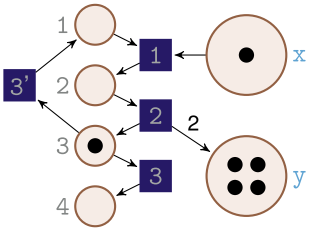
```
erraticDouble(x,y)
    x -= 1
    y += 2                    (1,2)
    goto 4 or goto 1
    end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

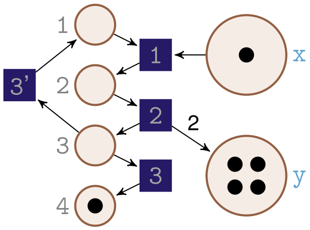
```
erraticDouble(x,y)
1   x -= 1
2   y += 2
3   goto 4 or goto 1  (1,4)
4   end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ **Petri Net**

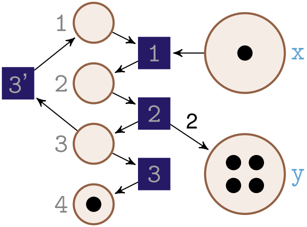
```
erraticDouble(x,y)
 1    x -= 1
 2    y += 2
 3    goto 4 or goto 1
 4    end                  (1,4)
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

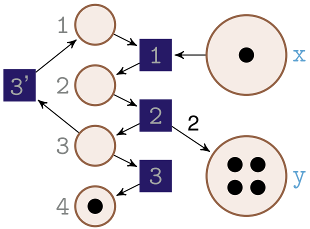
```
erraticDouble(x,y)
 1    x -= 1
 2    y += 2
 3    goto 4 or goto 1
 4    end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

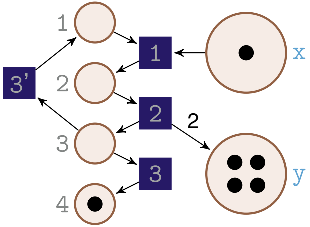
```
erraticDouble(x,y)
 1    x -= 1
 2    y += 2
 3    goto 4 or goto 1
 4    end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic counter program ⇒ Petri Net

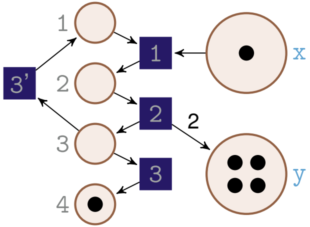
```
erraticDouble(x,y)
 1    x -= 1
 2    y += 2
 3    goto 4 or goto 1
 4    end
```




1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

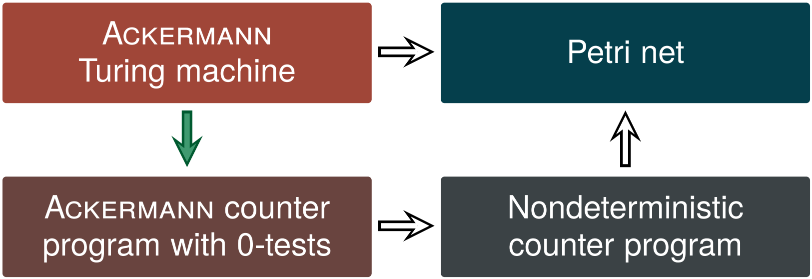
Nondeterministic counter program → Petri Net

```
erraticDouble(x,y)
 1    x -= 1
 2    y += 2
 3    goto 4 or goto 1
 4    end
```

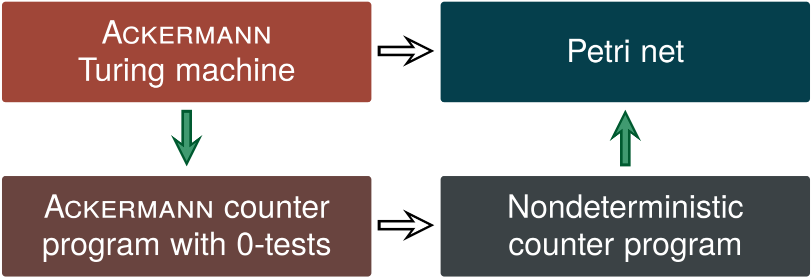



1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

# Reachability for Petri Nets is Ackermann-hard



## Reachability for Petri Nets is Ackermann-hard



Ackermann counter program with 0-tests ⇒ Nondeterministic counter program

```
double(x,y)
1  x -= 1
2  y += 2
3  if x == 0 then goto 4 else goto 1
4  end
```

?

Ackermann counter program with 0-tests ⟹ Nondeterministic counter program

```
double(x,y)
 1  x -= 1
 2  y += 2
 3  if x == 0 then goto 4 else goto 1
 4  end
```

↓

```
erraticDouble(x,y)
 1   x -= 1
 2   y += 2
 3   goto 4 or goto 1
 4   end
```

Ackermann counter program with 0-tests

Nondeterministic counter program

```
double(x,y)
x -= 1
y += 2
if x == 0 then goto 4 else goto 1
end
```

```
erraticDouble(x,y)
x -= 1
y += 2
goto 4 or goto 1
end
```

How can we control that the correct path was chosen?

Ackermann counter program with 0-tests $\Rightarrow$ Nondeterministic counter program

**Key technique:** Simulating zero tests with invariants

E:
```
2 * (x + x1) * x2 == x3
```

B:
```
2 * (x + x1) * x2 <  x3
```

Ackermann counter program with 0-tests $\Longrightarrow$ Nondeterministic counter program

**Key technique:** Simulating zero tests with invariants

**E**: `2 * (x + x1) * x2 == x3`

**B**: `2 * (x + x1) * x2  <  x3`

- **E** can be preserved iff `x` is 0:

  `E && x == 0` $\xrightarrow{\exists \text{ run of } \texttt{xTest}}$ **E**

  `E && x != 0` $\xrightarrow{\forall \text{ run of } \texttt{xTest}}$ **B**

- **E** cannot be repaired:

  **B** $\xrightarrow{\forall \text{ run of } \texttt{xTest}}$ **B**

```
xTest(x,x1,x2,x3)
1    x1 -= 1
2    x += 1
3    x3 -= 1
4    goto 5 or goto 1
5    x -= 1
6    x1 += 1
7    x3 -= 1
8    goto 9 or goto 5
9    x2 -= 1
10   end
```

ACKERMANN counter program with 0-tests ⇒ Nondeterministic counter program

**Key technique:** Simulating zero tests with invariants

**E:**
```
2 * (x + x1) * x2 == x3
```

**B:**
```
2 * (x + x1) * x2  <  x3
```

**Difficulty:** The initialisation requires to compute $\mathcal{A}(n)$

ACKERMANN counter program with 0-tests $\Rightarrow$ Nondeterministic counter program

**Key technique:** Simulating zero tests with invariants

**E**: `2 * (x + x1) * x2 == x3`

**B**: `2 * (x + x1) * x2  <  x3`

**Difficulty:** The initialisation requires to compute $\mathcal{A}(n)$

For all $n \in \mathbb{N}$ we can build a **NCP** with $\mathcal{O}(n)$ lines that **needs** $\mathcal{A}(n)$ steps to go from $\vec{0}$ to $\vec{0}$ and uses the following number of counters:

| $6n$ | $4n + 5$ | $3n + 2$ | $2n + 3$ |
|---|---|---|---|
| 2021 | 2021 | 2022 | 2023 |
| Czerwiński & Orlikowski | Leroux | Lasota | Czerwiński, J., Lasota et al. |

ACKERMANN counter program with 0-tests ⇒ Nondeterministic counter program

**Key technique:** Simulating zero tests with invariants

**E**:
```
2 * (x + x1) * x2 == x3
```

**B**:
```
2 * (x + x1) * x2  <  x3
```

**Difficulty:** The initialisation requires to compute $\mathcal{A}(n)$

For all $n \in \mathbb{N}$ we can build a **NCP** with $\mathcal{O}(n)$ lines that **needs** $\mathcal{A}(n)$ steps to go from $\vec{0}$ to $\vec{0}$ and uses the following number of counters:

| $6n$ | $4n + 5$ | $3n + 2$ | $2n + 3$ |
|---|---|---|---|
| 2021 | 2021 | 2022 | 2023 |
| Czerwiński & Orlikowski | Leroux | Lasota | Czerwiński, J., Lasota et al. |

Ackermann counter program with 0-tests → Nondeterministic counter program

**Key technique:** Simulating zero tests with invariants

**E**: `2 * (x + x1) * x2 == x3`

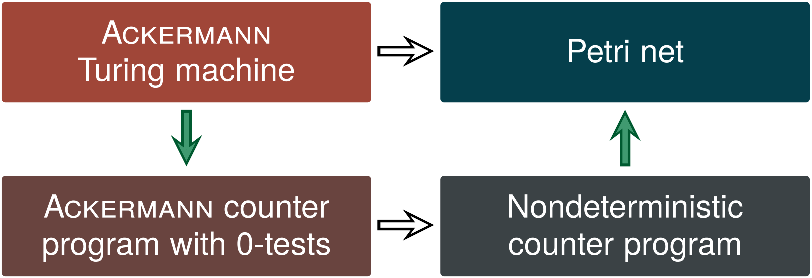
**B**: `2 * (x + x1) * x2  <  x3`

**Difficulty:** The initialisation requires to compute $\mathcal{A}(n)$

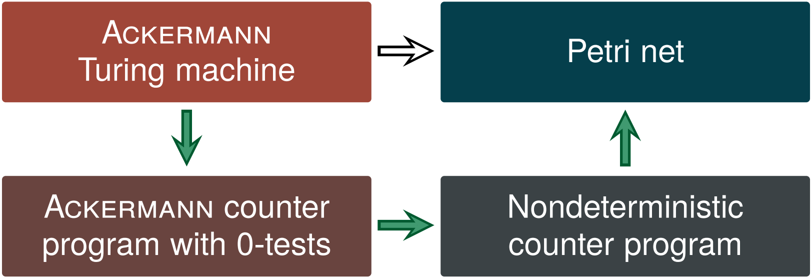
For all $n \in \mathbb{N}$ we can build a **NCP** with $\mathcal{O}(n)$ lines that **needs** $\mathcal{A}(n)$ steps to go from $\vec{0}$ to $\vec{0}$ and uses the following number of counters:

| | | | |
|---|---|---|---|
| $6n$ | $4n + 5$ | $3n + 2$ | $2n + 3$ |
| 2021 | 2021 | 2022 | 2023 |
| Czerwiński & Orlikowski | Leroux | Lasota | Czerwiński, J., Lasota et al. |

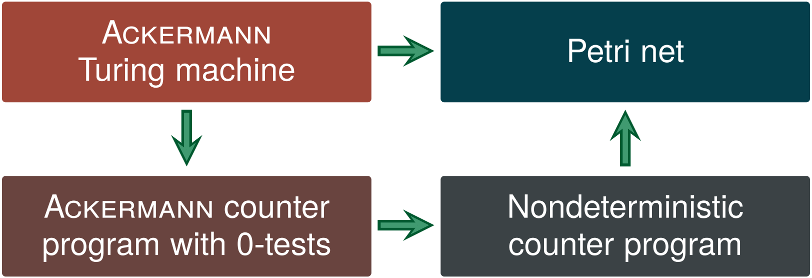
# Reachability for Petri Nets is Ackermann-hard



# Reachability for Petri Nets is Ackermann-hard



## Reachability for Petri Nets is Ackermann-hard



| REACHABILITY | |
|---|---|
| **Input:** | An NCP P and two valuations s, t |
| **Question:** | Can P(s) end with the valuation t? |

**in ACKERMANN**

2015: Leroux & Schmitz
2019: Leroux & Schmitz

**ACKERMANN-hard**

2021: Czerwiński & Orlikowski / Leroux
2022: Lasota
2023: Czerwiński, J., Lasota et al.

→ **ACKERMANN-complete**

| REACHABILITY | |
|---|---|
| **Input:** | An NCP $\mathtt{P}$ and two valuations $s$, $t$ |
| **Question:** | Can $\mathtt{P}(s)$ end with the valuation $t$? |

**in ACKERMANN**

2015: Leroux & Schmitz
2019: Leroux & Schmitz

**ACKERMANN-hard**

2021: Czerwiński & Orlikowski / Leroux
2022: Lasota
2023: Czerwiński, J., Lasota et al.

→ **ACKERMANN-complete**

| REACHABILITY | |
|---|---|
| **Input:** | An NCP P and two valuations s, t |
| **Question:** | Can P(s) end with the valuation t? |

**in ACKERMANN**

2015: Leroux & Schmitz
2019: Leroux & Schmitz

**ACKERMANN-hard**

2021: Czerwiński & Orlikowski / Leroux
2022: Lasota
2023: Czerwiński, J., Lasota et al.

→ **ACKERMANN-complete**

The reachability problem for counter programs **with $n$ counters** is:

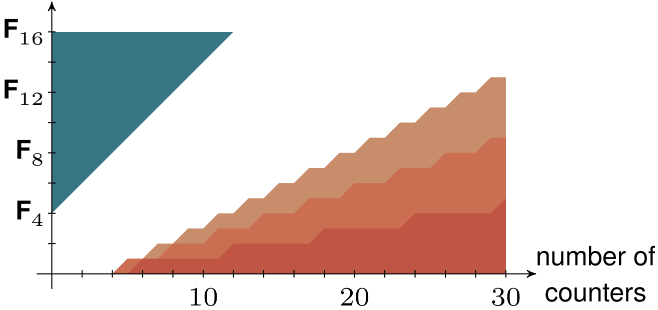
| in $\mathbf{F}_{n+4}$ | $\mathbf{F}_{\lfloor \frac{n}{6} \rfloor}$-hard | $\mathbf{F}_{\lfloor \frac{n-2}{3} \rfloor}$-hard | $\mathbf{F}_{\lfloor \frac{n-3}{2} \rfloor}$-hard |
|---|---|---|---|
| 2019 | 2021 | 2022 | 2023 |
| Leroux & Schmitz | Czerwiński & Orlikowski | Lasota | Czerwiński, J., Lasota et al. |

| | |
|---|---|
| REACHABILITY | **Input:** An NCP $\mathtt{P}$ and two valuations $\mathsf{s}$, $\mathsf{t}$<br>**Question:** Can $\mathtt{P}(\mathsf{s})$ end with the valuation $\mathsf{t}$? |

The reachability problem for counter programs **with $n$ counters** is:

| in $\mathbf{F}_{n+4}$ | $\mathbf{F}_{\lfloor \frac{n}{6} \rfloor}$-hard | $\mathbf{F}_{\lfloor \frac{n-2}{3} \rfloor}$-hard | $\mathbf{F}_{\lfloor \frac{n-3}{2} \rfloor}$-hard |
|---|---|---|---|
| 2019 | 2021 | 2022 | 2023 |
| Leroux & Schmitz | Czerwiński & Orlikowski | Lasota | Czerwiński, J., Lasota et al. |

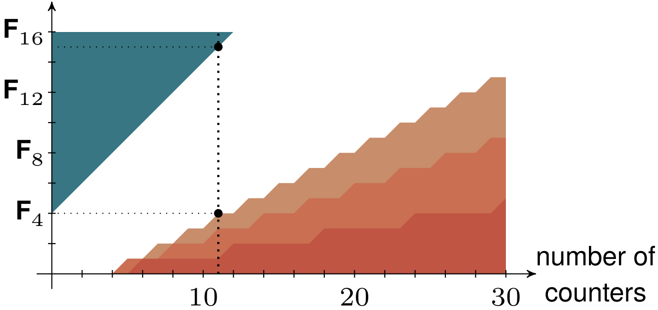
| Reachability | |
|---|---|
| **Input:** | An NCP $\texttt{P}$ and two valuations $\mathsf{s}$, $\mathsf{t}$ |
| **Question:** | Can $\texttt{P}(\mathsf{s})$ end with the valuation $\mathsf{t}$? |




The reachability problem for counter programs **with $n$ counters** is:

| in $F_{n+4}$ | $F_{\lfloor \frac{n}{6} \rfloor}$-hard | $F_{\lfloor \frac{n-2}{3} \rfloor}$-hard | $F_{\lfloor \frac{n-3}{2} \rfloor}$-hard |
|---|---|---|---|
| 2019 | 2021 | 2022 | 2023 |
| Leroux & Schmitz | Czerwiński & Orlikowski | Lasota | Czerwiński, J., Lasota et al. |

| REACHABILITY | |
|---|---|
| | **Input:** An NCP P and two valuations s, t |
| | **Question:** Can P(s) end with the valuation t? |



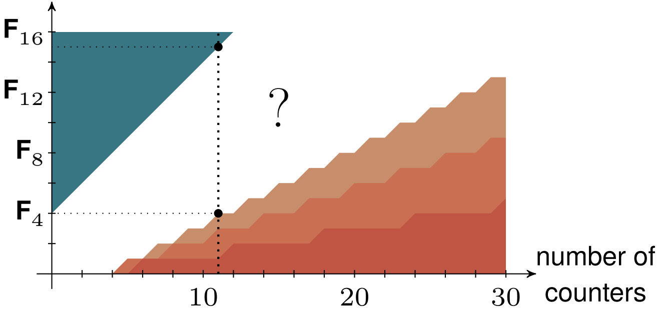

The reachability problem for counter programs **with $n$ counters** is:

| in $F_{n+4}$ | $F_{\lfloor \frac{n}{6} \rfloor}$-hard | $F_{\lfloor \frac{n-2}{3} \rfloor}$-hard | $F_{\lfloor \frac{n-3}{2} \rfloor}$-hard |
|---|---|---|---|
| 2019 | 2021 | 2022 | 2023 |
| Leroux & Schmitz | Czerwiński & Orlikowski | Lasota | Czerwiński, J., Lasota et al. |

**REACHABILITY**

**Input:** An NCP P and two valuations s, t

**Question:** Can P(s) end with the valuation t?

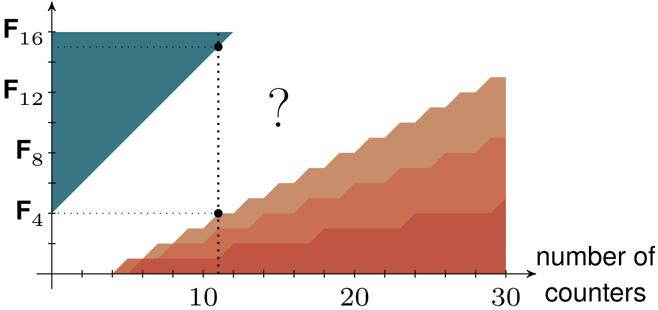



The reachability problem for counter programs **with $n$ counters** is:

| in $F_{n+4}$ | $F_{\lfloor \frac{n}{6} \rfloor}$-hard | $F_{\lfloor \frac{n-2}{3} \rfloor}$-hard | $F_{\lfloor \frac{n-3}{2} \rfloor}$-hard |
|---|---|---|---|
| 2019 | 2021 | 2022 | 2023 |
| Leroux & Schmitz | Czerwiński & Orlikowski | Lasota | Czerwiński, J., Lasota et al. |

**Reachability**

**Input:** An NCP $\mathtt{P}$ and two valuations s, t
**Question:** Can $\mathtt{P}$(s) end with the valuation t?



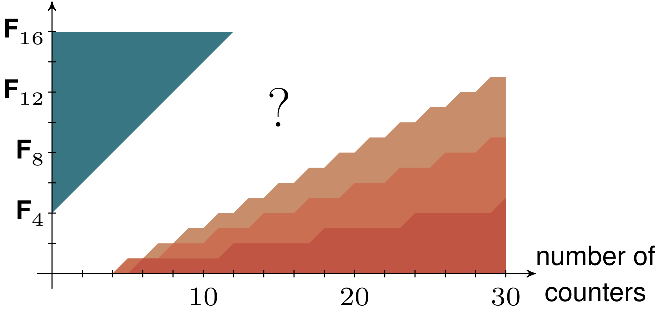

**Open question**

What is the complexity of Reachability for **NCP** parametrised by the number of counters?

**Reachability**

**Input:** An NCP $\mathtt{P}$ and two valuations $\mathsf{s}$, $\mathsf{t}$

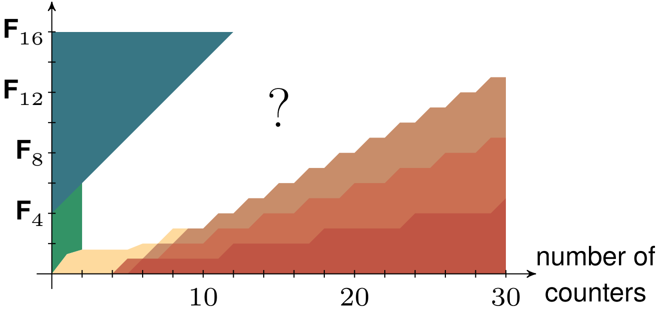
**Question:** Can $\mathtt{P}(\mathsf{s})$ end with the valuation $\mathsf{t}$?




**Open question**

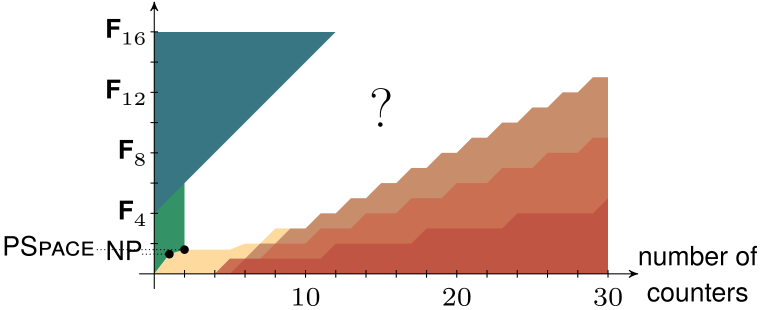
What is the complexity of Reachability for **NCP** parametrised by the number of counters?

| REACHABILITY | |
|---|---|
| | **Input:** An NCP P and two valuations s, t |
| | **Question:** Can P(s) end with the valuation t? |




| Open question | What is the complexity of REACHABILITY for **NCP** parametrised by the number of counters? |
|---|---|

| REACHABILITY | |
|---|---|
| | **Input:** An NCP P and two valuations s, t |
| | **Question:** Can P(s) end with the valuation t? |

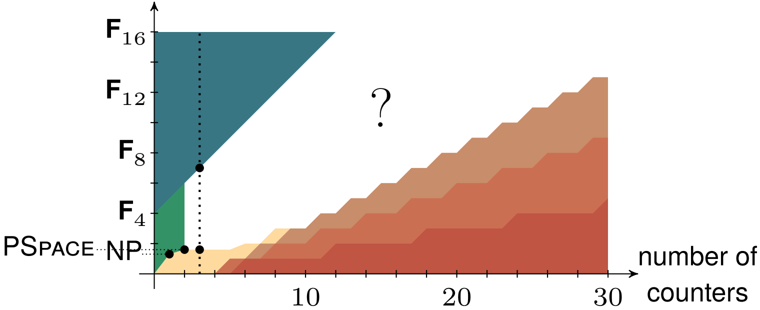



| Open question | What is the complexity of REACHABILITY for **NCP** parametrised by the number of counters? |
|---|---|

**Reachability**

**Input:** An NCP P and two valuations s, t

**Question:** Can P(s) end with the valuation t?




**Open question** What is the complexity of Reachability for **NCP** parametrised by the number of counters?

**Open question** What is the complexity of Reachability for 3-counters **NCP**?

# Pushdown nondeterministic counter program

name stack variables

list of instructions

```
program(s,x,y)
1 …
⋮
```

**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

**Advanced instructions**:

- Jump nondet. `goto i or goto j`
- Push `push(s,n)`
- Pop `pop(s)`
- Pop-test `if pop(s)==n then goto i else goto j`

## Pushdown nondeterministic counter program

| REACHABILITY | |
|---|---|
| **Input:** | A PNCP $\mathtt{P}$ and two valuations $\mathsf{s}$, $\mathsf{t}$ |
| **Question:** | Can $\mathtt{P}(\mathsf{s})$ end with the valuation $\mathsf{t}$? |

**Basic instructions**:

- Increment counter
```
x += n
```
- Decrement counter
```
x -= n
```
- Stop the program
```
end
```

**Advanced instructions**:

- Jump nondet.
```
goto i or goto j
```
- Push
```
push(s,n)
```
- Pop
```
pop(s)
```
- Pop-test
```
if pop(s)==n then goto i else goto j
```

# Pushdown nondeterministic counter program

| REACHABILITY | |
|---|---|
| **Input:** | A PNCP P and two valuations s, t |
| **Question:** | Can P(s) end with the valuation t? |

| Open question | Is REACHABILITY decidable for pushdown NCP? |
|---|---|

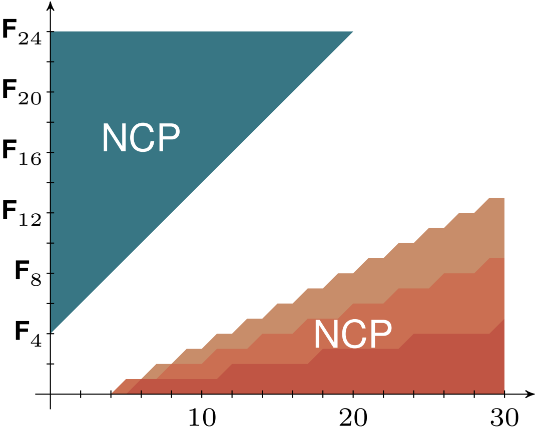
**Basic instructions**:

- Increment counter `x += n`
- Decrement counter `x -= n`
- Stop the program `end`

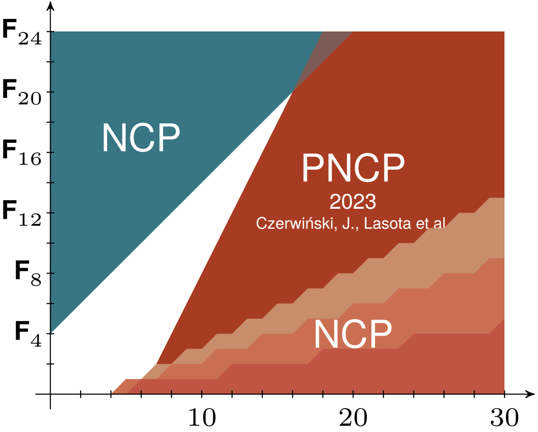
**Advanced instructions**:

- Jump nondet. `goto i or goto j`
- Push `push(s,n)`
- Pop `pop(s)`
- Pop-test `if pop(s)==n then goto i else goto j`

# Pushdown nondeterministic counter program



# Pushdown nondeterministic counter program



For all $n \in \mathbb{N}$ we can build a **NCP** with $\mathcal{O}(n)$ lines that **needs** $\mathcal{A}(n)$ steps to go from $\vec{0}$ to $\vec{0}$ and uses the following number of counters:

| $6n$ | $4n + 5$ | $3n + 2$ | $2n + 3$ |
|---|---|---|---|
| 2021 | 2021 | 2022 | 2023 |
| Czerwiński & Orlikowski | Leroux | Lasota | Czerwiński, J., Lasota et al. |

**Can YOU do better?** $n + c$ OPEN

**NCP instructions**:

- Increment counter

```
x += d
```

- Decrement counter

```
x -= d
```

- Jump nondeterministically

```
goto i or goto j
```

**Ackermann function**:

$$\mathcal{A}(n) = f_n(n)$$

$$f_1(n) = 2n$$

$$f_{i+1}(n) = \Big(\underbrace{f_i \circ \ldots \circ f_i}_{n \text{ times}}\Big)(1)$$