

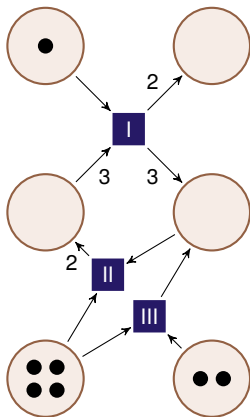
HOW HARD IS THE REACHABILITY PROBLEM FOR PETRI NETS?

Ismaël Jecker

Petri net

A Petri net is a directed graph composed of

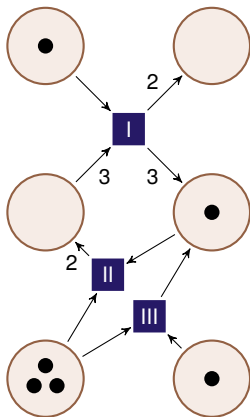
- **Places**, that contain tokens
- **Transitions**, that can transfer tokens



Petri net

A Petri net is a directed graph composed of

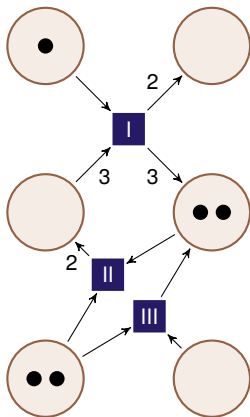
- **Places**, that contain tokens
- **Transitions**, that can transfer tokens



Petri net

A Petri net is a directed graph composed of

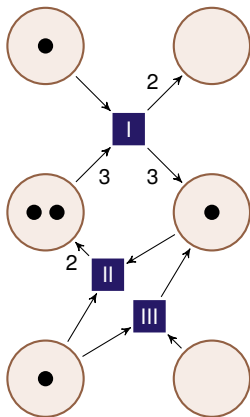
- **Places**, that contain tokens
- **Transitions**, that can transfer tokens



Petri net

A Petri net is a directed graph composed of

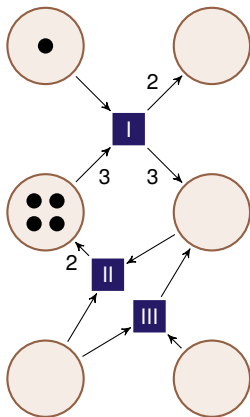
- **Places**, that contain tokens
- **Transitions**, that can transfer tokens



Petri net

A Petri net is a directed graph composed of

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



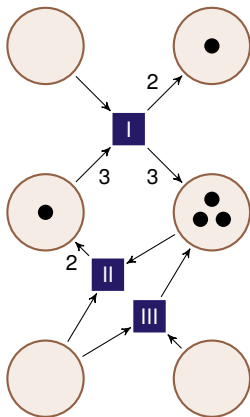
Petri net

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



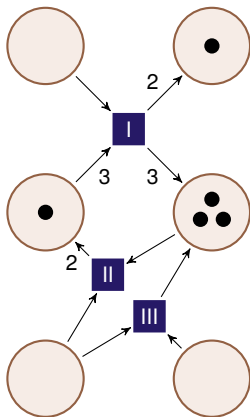
Petri net

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



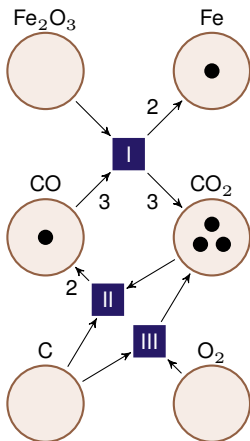
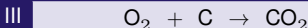
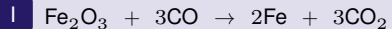
Petri net

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



REACHABILITY

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

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

Question: Starting from s , can P reach t ?

Decidable

1981: Mayr

1982: Kosaraju

1992: Lambert

EXPSpace-hard

1976: Lipton

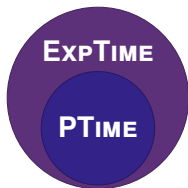
in ACKERMANN

2015: Leroux & Schmitz

2019: Leroux & Schmitz

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

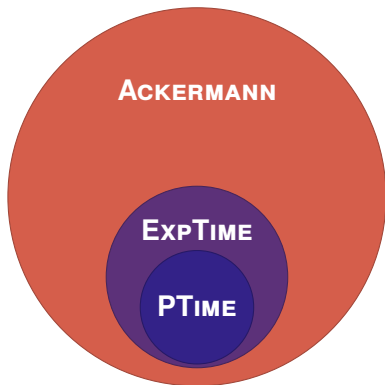
EXPTIME: can be solved in **exponential** time



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

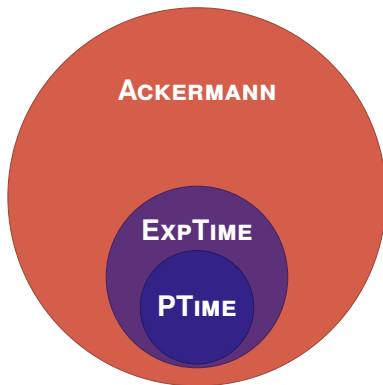
EXPTIME: can be solved in **exponential** time

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

Fast growing hierarchy:

$$f_{i+1}(n) = \left(\underbrace{f_i \circ \dots \circ f_i}_{n \text{ times}} \right)(1)$$

$$f_1(n) = 2n$$



$$f_1(1) = 2$$

$$f_1(2) = 4$$

$$f_1(3) = 6$$

$$f_1(4) = 8$$

$$f_1(5) = 10$$

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

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

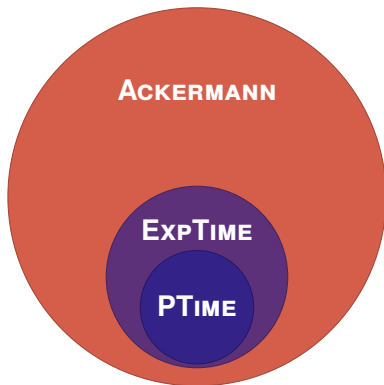
ACKERMANN: can be solved with a **HUGE** amount of time

Fast growing hierarchy:

$$f_{i+1}(n) = \left(\underbrace{f_i \circ \dots \circ f_i}_{n \text{ times}} \right)(1)$$

$$f_1(n) = 2n$$

$$f_2(n) = 2^n$$



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

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

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

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

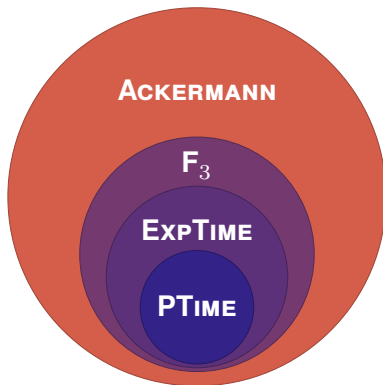
Fast growing hierarchy:

$$f_{i+1}(n) = \left(\underbrace{f_i \circ \dots \circ f_i}_{n \text{ times}} \right)(1)$$

$$f_1(n) = 2n$$

$$f_2(n) = 2^n$$

$$f_3(n) = \underbrace{2^{2^{\dots^2}}}_{n \text{ times}}$$



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

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:

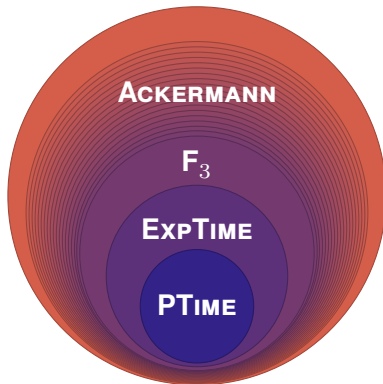
$$f_{i+1}(n) = \left(\underbrace{f_i \circ \dots \circ f_i}_{n \text{ times}} \right)(1)$$

$$f_1(n) = 2n$$

$$f_2(n) = 2^n$$

$$f_3(n) = \underbrace{2^{2^{\dots^2}}}_{n \text{ times}}$$

\vdots



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:

$$f_{i+1}(n) = \left(\underbrace{f_i \circ \dots \circ f_i}_{n \text{ times}} \right)(1)$$

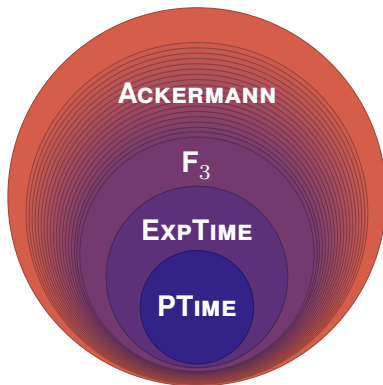
$$f_1(n) = 2n$$

$$f_2(n) = 2^n$$

$$f_3(n) = \underbrace{2^{2^{\dots^2}}}_{n \text{ times}}$$

\vdots

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



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:

$$f_{i+1}(n) = \left(\underbrace{f_i \circ \dots \circ f_i}_{n \text{ times}} \right)(1)$$

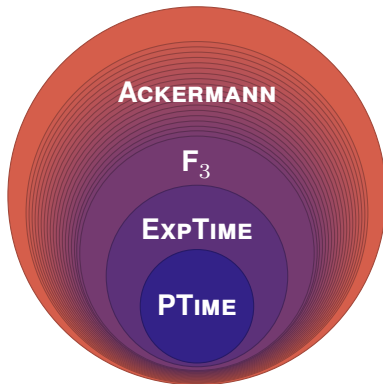
$$f_1(n) = 2n$$

$$f_2(n) = 2^n$$

$$f_3(n) = \underbrace{2^{2^{\dots^2}}}_{n \text{ times}}$$

\vdots

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



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

in ACKERMANN

2015: Leroux & Schmitz

2019: Leroux & Schmitz

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

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

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

F_3 -hard

2019: Czerwiński, Lasota et al.

ACKERMANN-hard

2021: Czerwiński & Orlikowski / Leroux

2022: Lasota

2023: Czerwiński, J., Lasota et al.

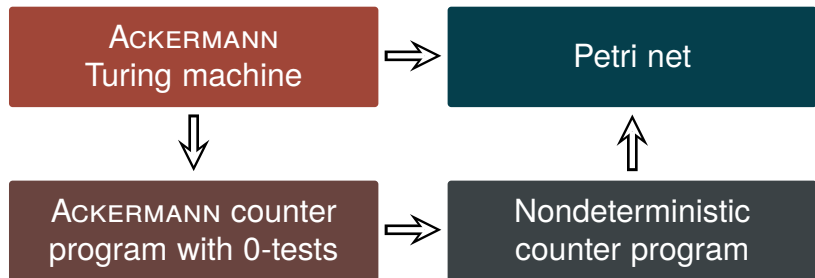


ACKERMANN-complete

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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement Counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Integer division

```
x *= n
```

```
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:

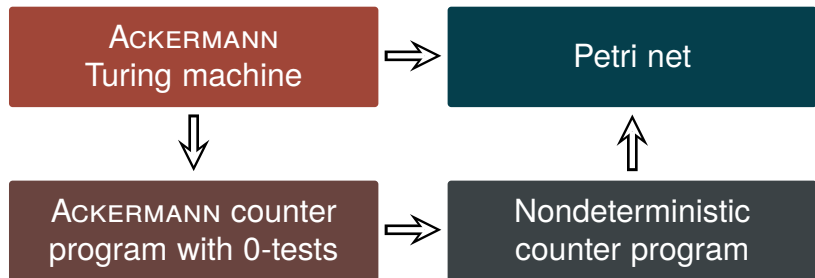
- Multiplication
- Integer division

```
x *= n
```

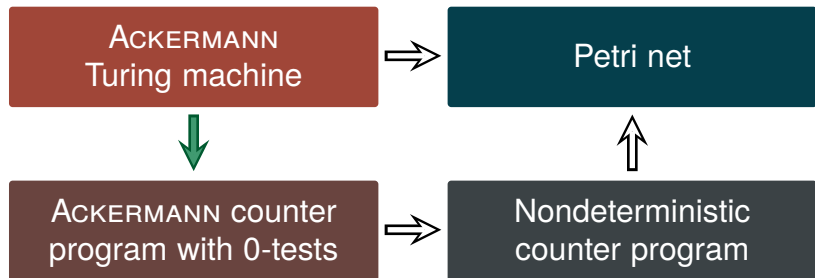
```
x /= n
```

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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
end
```

Advanced instruction:

- Jump nondeterministically

```
goto i or goto j
```

Nondeterministic counter program (NCP)

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

Basic instructions:

- Increment counter
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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                     (1,2)
3  goto 4 or goto 1
4  end
```

Basic instructions:

- Increment counter
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
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
```



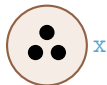
1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



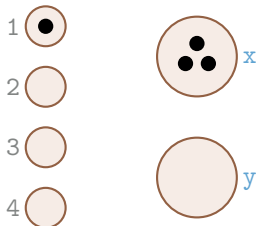
1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



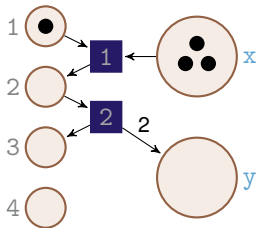
1. One **place** per variable (tokens simulate their contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



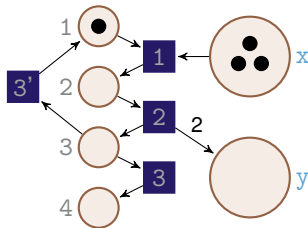
1. One **place** per variable (tokens simulate their contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



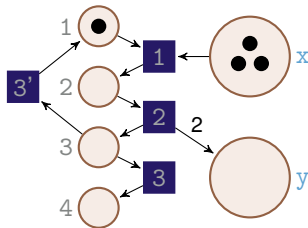
1. One **place** per variable (tokens simulate their contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



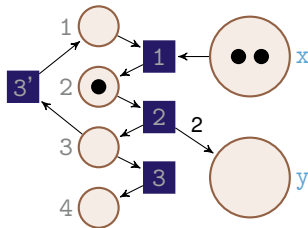
1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



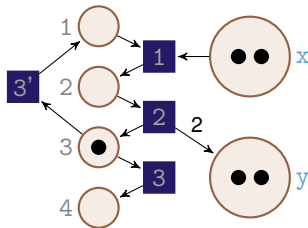
1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



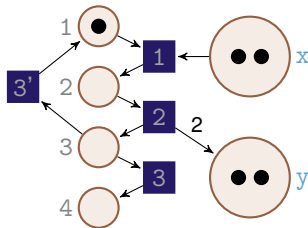
1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



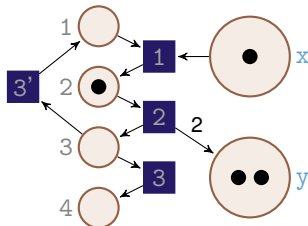
1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



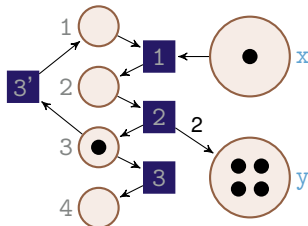
1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

Nondeterministic
counter program



Petri Net

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



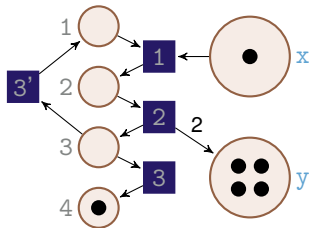
1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

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

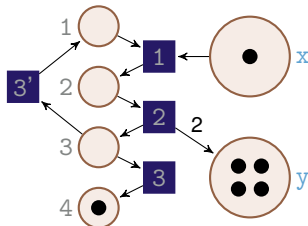
Nondeterministic
counter program



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

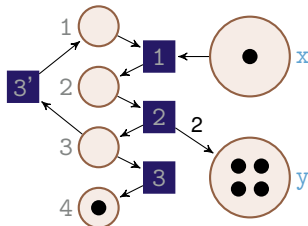
Nondeterministic
counter program



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

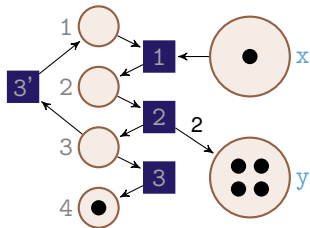
Nondeterministic
counter program



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

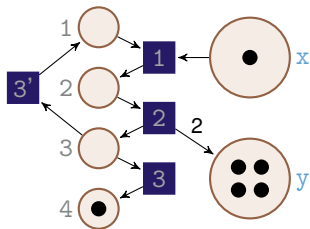
Nondeterministic
counter program



Petri Net

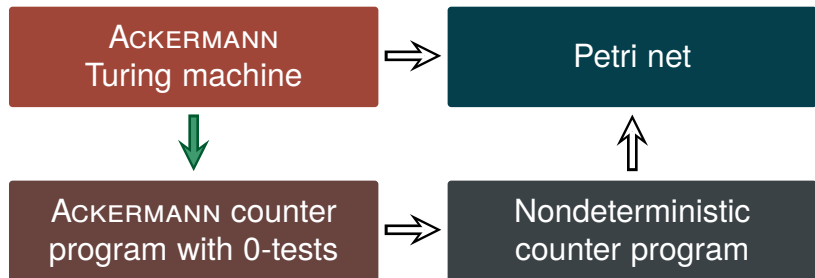
```
erraticDouble(x,y)
```

```
1  x -= 1  
2  y += 2  
3  goto 4 or goto 1  
4  end
```

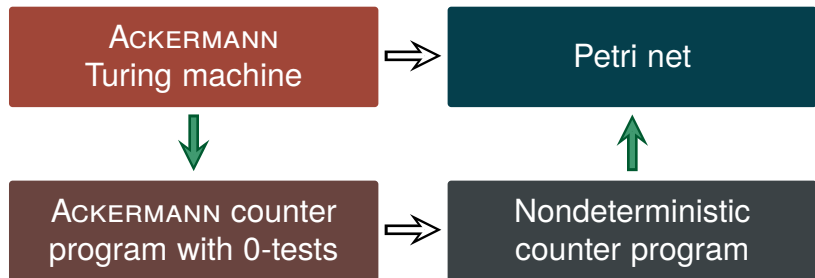


1. One **place** per variable (tokens simulate the contents)
2. One **place** per line (a token simulates the active line)
3. One **transition** per increment/decrement
4. Two **transitions** per nondeterministic jump

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)
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
```

How can we control that the correct path was chosen?

ACKERMANN counter
program with 0-tests



Nondeterministic
counter program

Key technique: Simulating zero tests with invariants

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

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

ACKERMANN counter
program with 0-tests



Nondeterministic
counter program

Key technique: Simulating zero tests with invariants

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

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

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

E && $x == 0 \xrightarrow{\exists \text{ run of } xTest} \textbf{E}$

E && $x \neq 0 \xrightarrow{\forall \text{ run of } xTest} \textbf{B}$

- **E** cannot be repaired:

B $\xrightarrow{\forall \text{ run of } xTest} \textbf{B}$

```
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



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:

$$6n$$

2021

Czerwiński & Orlikowski

$$4n + 5$$

2021

Leroux

$$3n + 2$$

2022

Lasota

$$2n + 3$$

2023

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:

$$6n$$

2021

Czerwiński & Orlikowski

$$4n + 5$$

2021

Leroux

$$3n + 2$$

2022

Lasota

$$2n + 3$$

2023

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:

$$6n$$

2021

Czerwiński & Orlikowski

$$4n + 5$$

2021

Leroux

$$3n + 2$$

2022

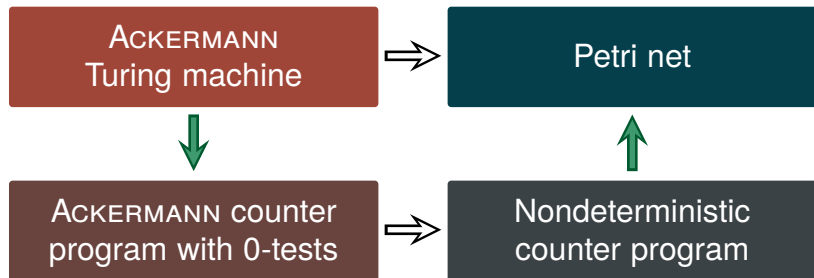
Lasota

$$2n + 3$$

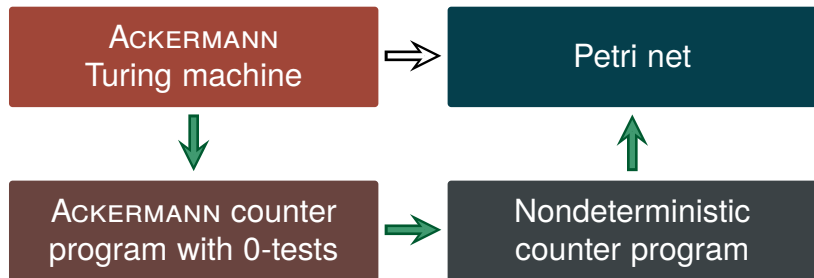
2023

Czerwiński, J., Lasota et al.

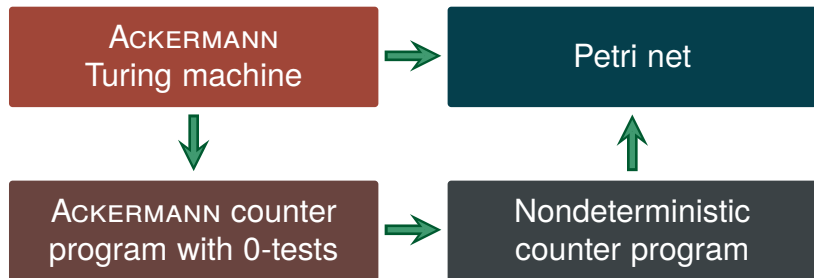
REACHABILITY FOR PETRI NETS IS ACKERMANN-HARD



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 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 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 F_{n+4}

2019

Leroux & Schmitz

$F_{\lfloor \frac{n}{6} \rfloor}$ -hard

2021

Czerwiński & Orlikowski

$F_{\lfloor \frac{n-2}{3} \rfloor}$ -hard

2022

Lasota

$F_{\lfloor \frac{n-3}{2} \rfloor}$ -hard

2023

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}

2019

Leroux & Schmitz

$F_{\lfloor \frac{n}{6} \rfloor}$ -hard

2021

Czerwiński & Orlikowski

$F_{\lfloor \frac{n-2}{3} \rfloor}$ -hard

2022

Lasota

$F_{\lfloor \frac{n-3}{2} \rfloor}$ -hard

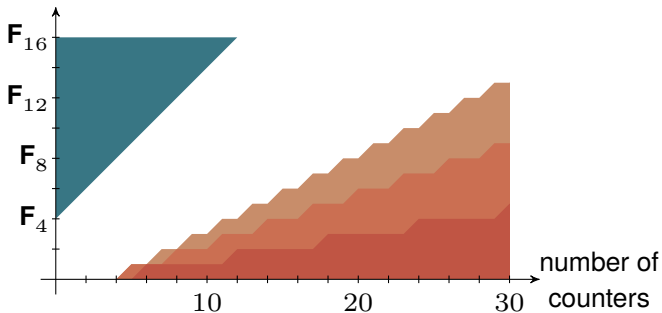
2023

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}

2019
Leroux & Schmitz

$F_{\lfloor \frac{n}{6} \rfloor}$ -hard

2021
Czerwiński & Orlikowski

$F_{\lfloor \frac{n-2}{3} \rfloor}$ -hard

2022
Lasota

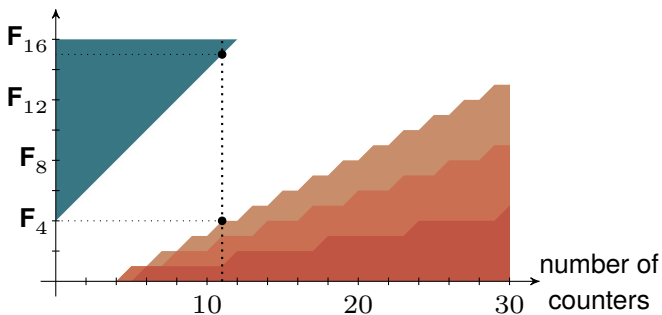
$F_{\lfloor \frac{n-3}{2} \rfloor}$ -hard

2023
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}

2019
Leroux & Schmitz

$F_{\lfloor \frac{n}{6} \rfloor}$ -hard

2021
Czerwiński & Orlikowski

$F_{\lfloor \frac{n-2}{3} \rfloor}$ -hard

2022
Lasota

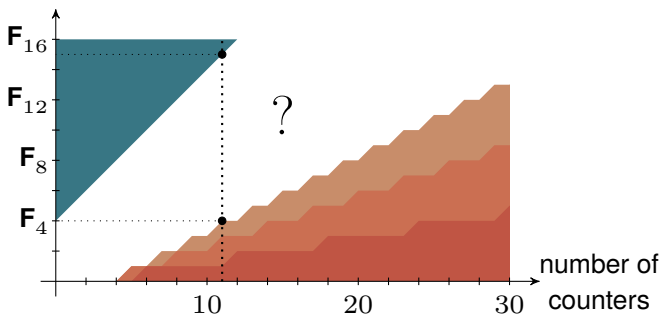
$F_{\lfloor \frac{n-3}{2} \rfloor}$ -hard

2023
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}

2019
Leroux & Schmitz

$F_{\lfloor \frac{n}{6} \rfloor}$ -hard

2021
Czerwinski & Orlikowski

$F_{\lfloor \frac{n-2}{3} \rfloor}$ -hard

2022
Lasota

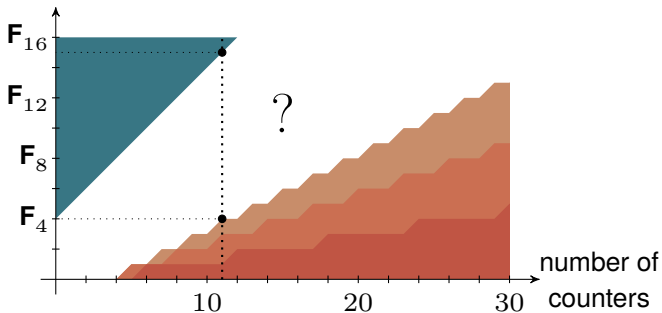
$F_{\lfloor \frac{n-3}{2} \rfloor}$ -hard

2023
Czerwinski, J., Lasota et al.

REACHABILITY

Input: An NCP P and two valuations s, t

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



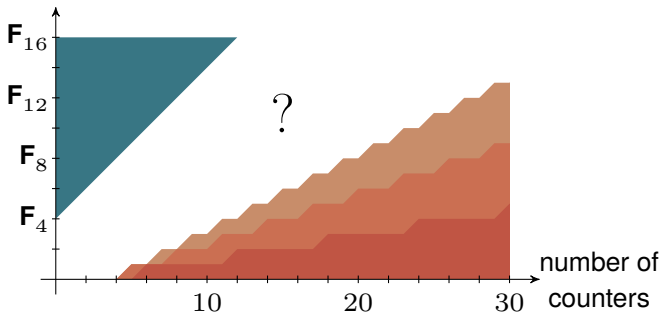
Open question

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 ?



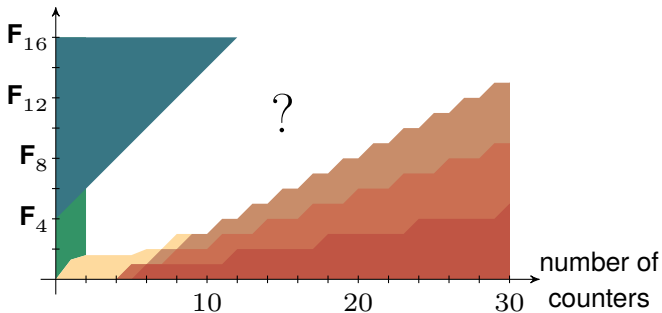
Open question

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 ?



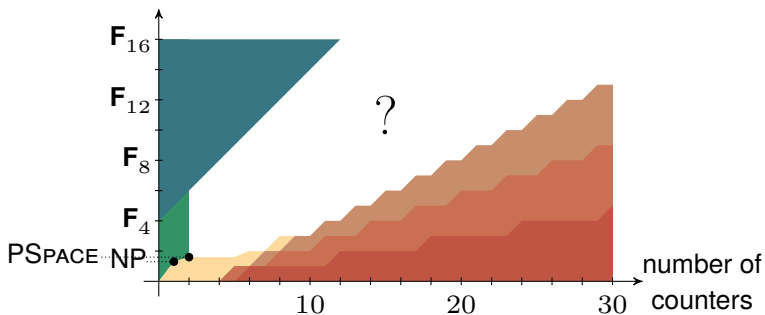
Open question

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

REACHABILITY

Input: An NCP P and two valuations s, t

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



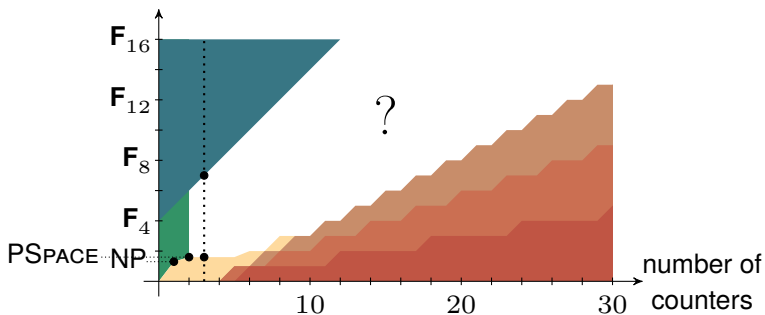
Open question

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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
end
```

Advanced instructions:

- Jump nondet.
- **Push**
- **Pop**
- **Pop-test**

```
goto i or goto j
```

```
push(s,n)
```

```
pop(s)
```

```
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 ?

Basic instructions:

- Increment counter
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
end
```

Advanced instructions:

- Jump nondet.
- Push
- Pop
- Pop-test

```
goto i or goto j
```

```
push(s,n)
```

```
pop(s)
```

```
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
- Decrement counter
- Stop the program

```
x += n
```

```
x -= n
```

```
end
```

Advanced instructions:

- Jump nondet.
- Push
- Pop
- Pop-test

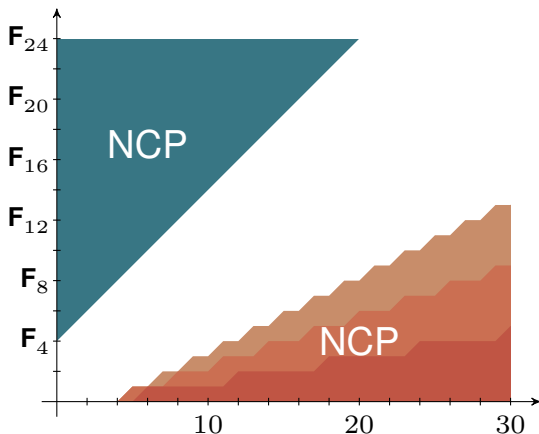
```
goto i or goto j
```

```
push(s,n)
```

```
pop(s)
```

```
if pop(s)==n then goto i else goto j
```

Pushdown nondeterministic counter program



Pushdown nondeterministic counter program

