The reachability problem for Petri nets is not elementary

Wojciech Czerwiński
Sławomir Lasota

University of Warsaw

Ranko Lazic

University of Warwick

Jerome Leroux
Filip Mazowiecki

University of Bordeaux

RP'19, Brussels, 2019.09.11

The reachability problem for Petri nets is not elementary

but the proof is so:)

Wojciech Czerwiński
Sławomir Lasota
University of Warsaw

Ranko Lazic
University of Warwick

Jerome Leroux
Filip Mazowiecki
University of Bordeaux

RP'19, Brussels, 2019.09.11

The reachability ... blem
f ...
is ... tary

crash course in counter programming (without zero tests)

... the proof is so:)

Wojciech Czerwiński
Sławomir Lasota

Ranko Lazic

Jerome Leroux
Filip Mazowiecki

University of Warsaw

University of Warwick

University of Bordeaux

RP'19, Brussels, 2019.09.11

Many faces of Petri nets

- Petri nets **[Petri 1962]**
- vector addition systems **VAS** **[Karp, Miller 1969]**
- vector addition systems with states **VASS** **[Hopcroft, Pansiot 1979]**
- automata with counters without zero tests
- **counter programs without zero tests**
- multiset rewriting
- …

Counter programs

a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')
zero? x	(continue if counter x equals 0)

counters are nonnegative

Counter programs

a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')
zero? x	(continue if counter x equals 0)

counters are nonnegative

abort if x=0

Counter programs

a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')
zero? x	(continue if counter x equals 0)

counters are nonnegative

abort if x=0

otherwise abort

Counter programs

a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')
zero? x	(continue if counter x equals 0)

counters are nonnegative

abort if x=0

otherwise abort

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

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)

otherwise abort

Counter programs

a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')
zero? x	(continue if counter x equals 0)

counters are nonnegative

abort if x=0

otherwise abort

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

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)

otherwise abort

Example:

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

Counter programs

a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')
zero? x	(continue if counter x equals 0)

counters are nonnegative

abort if x=0

otherwise abort

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

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)

otherwise abort

Example:

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

initially all counters 0:
x' = x = y = 0

Counter programs

a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')
zero? x	(continue if counter x equals 0)

counters are nonnegative

abort if x=0

otherwise abort

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

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)

otherwise abort

Example:

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

initially all counters 0:
x' = x = y = 0

finally:
x' = 0 x = 100 y = 200

Counter programs

a sequence of commands of the form:

$\mathsf{x} \mathrel{+}= 1$	(increment counter x)
$\mathsf{x} \mathrel{-}= 1$	(decrement counter x)
goto L **or** L'	(jump to either line L or line L')
zero? x	(continue if counter x equals 0)

counters are nonnegative

abort if x=0

otherwise abort

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

halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all the listed counters are zero)

otherwise abort

Example:

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

no zero tests

initially all counters 0:
x' = x = y = 0

finally:
x' = 0 x = 100 y = 200

Minsky machines

the conditional jump of Minsky machines

$$\textbf{if } \mathsf{x} = 0 \textbf{ then goto } L \textbf{ else } \mathsf{x} \mathrel{-}= 1$$

is simulated by counter program **with** zero tests:

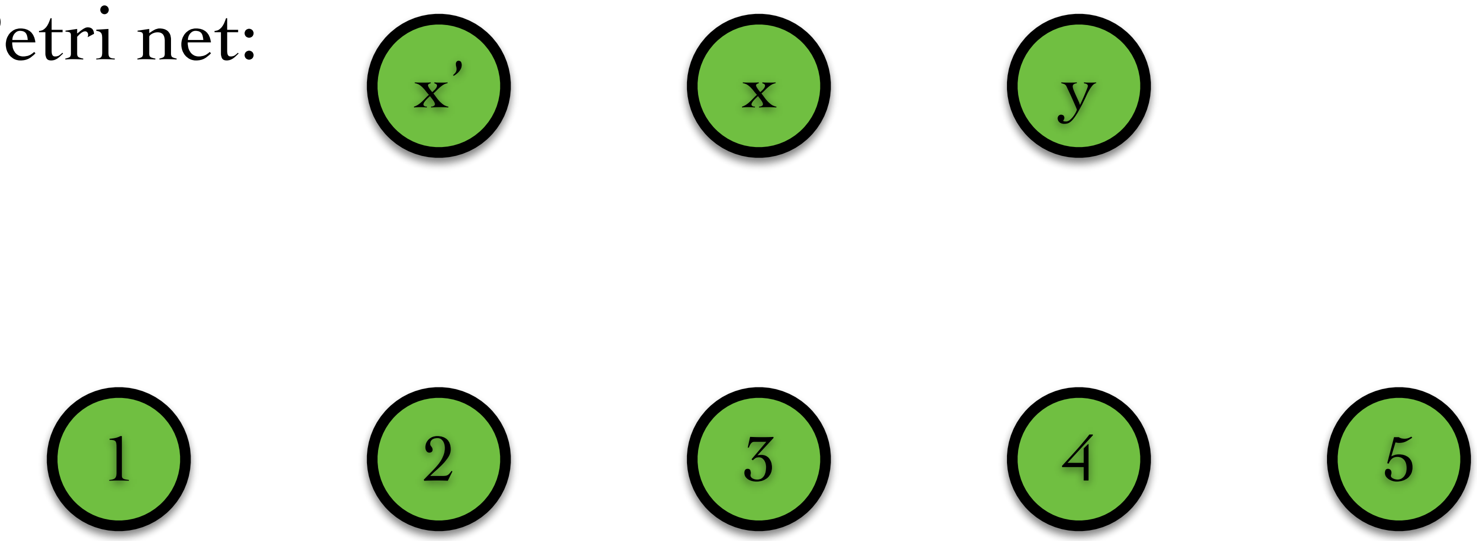
$$\begin{array}{l}
1\text{:}\ \ \textbf{goto } \boxed{2} \textbf{ or } \boxed{4} \\
2\text{:}\ \ \textbf{zero? } \mathsf{x} \\
3\text{:}\ \ \textbf{goto } L \\
4\text{:}\ \ \mathsf{x} \mathrel{-}= 1
\end{array}$$

Many faces of Petri nets

counter program without zero tests:

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

Petri net:

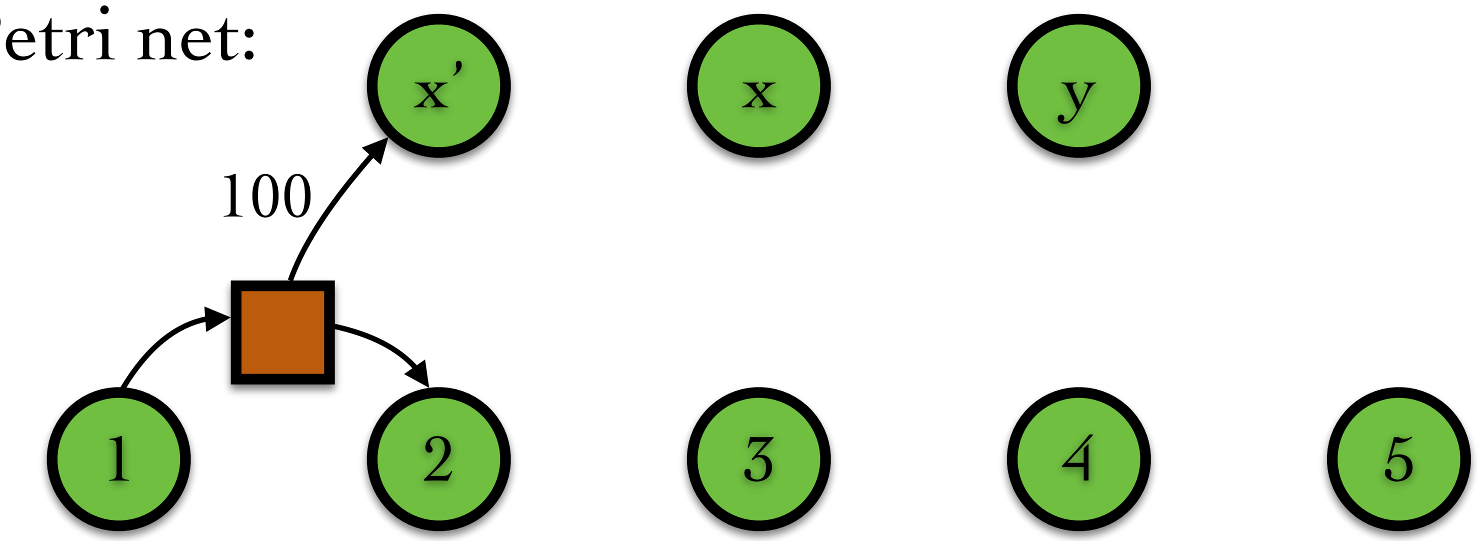


Many faces of Petri nets

counter program without zero tests:

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

Petri net:

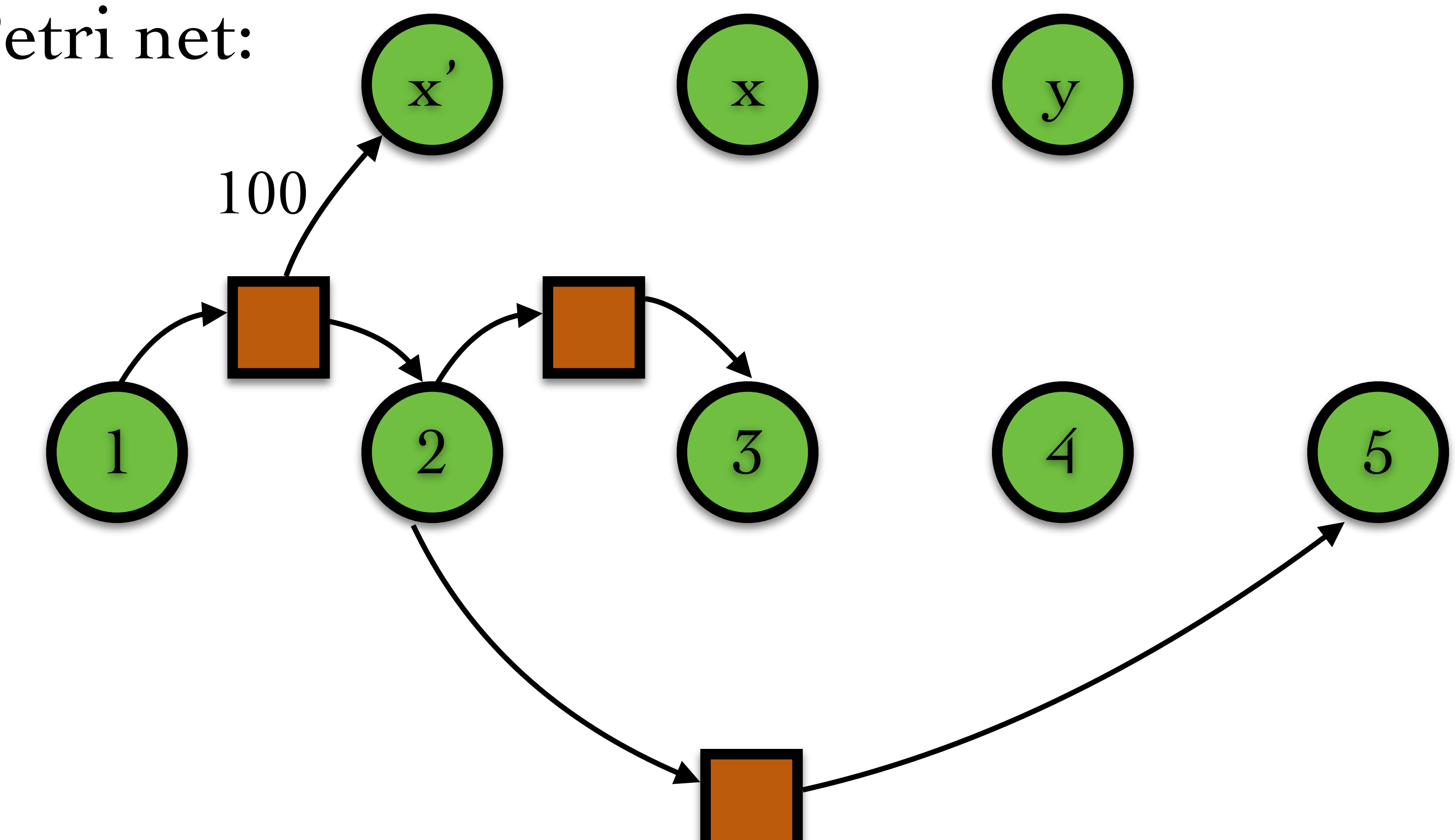


Many faces of Petri nets

counter program without zero tests:

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

Petri net:

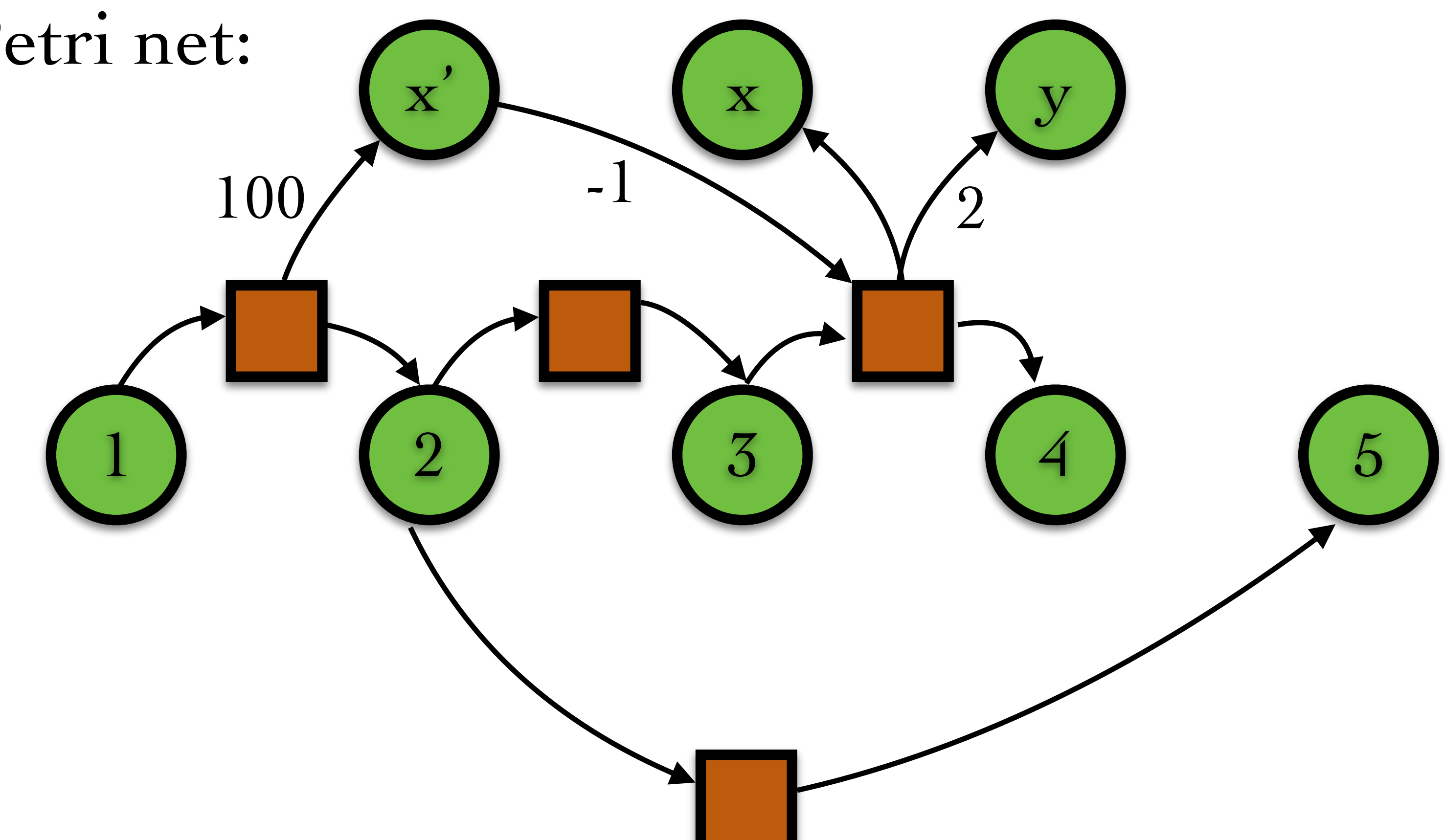


Many faces of Petri nets

counter program without zero tests:

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

Petri net:

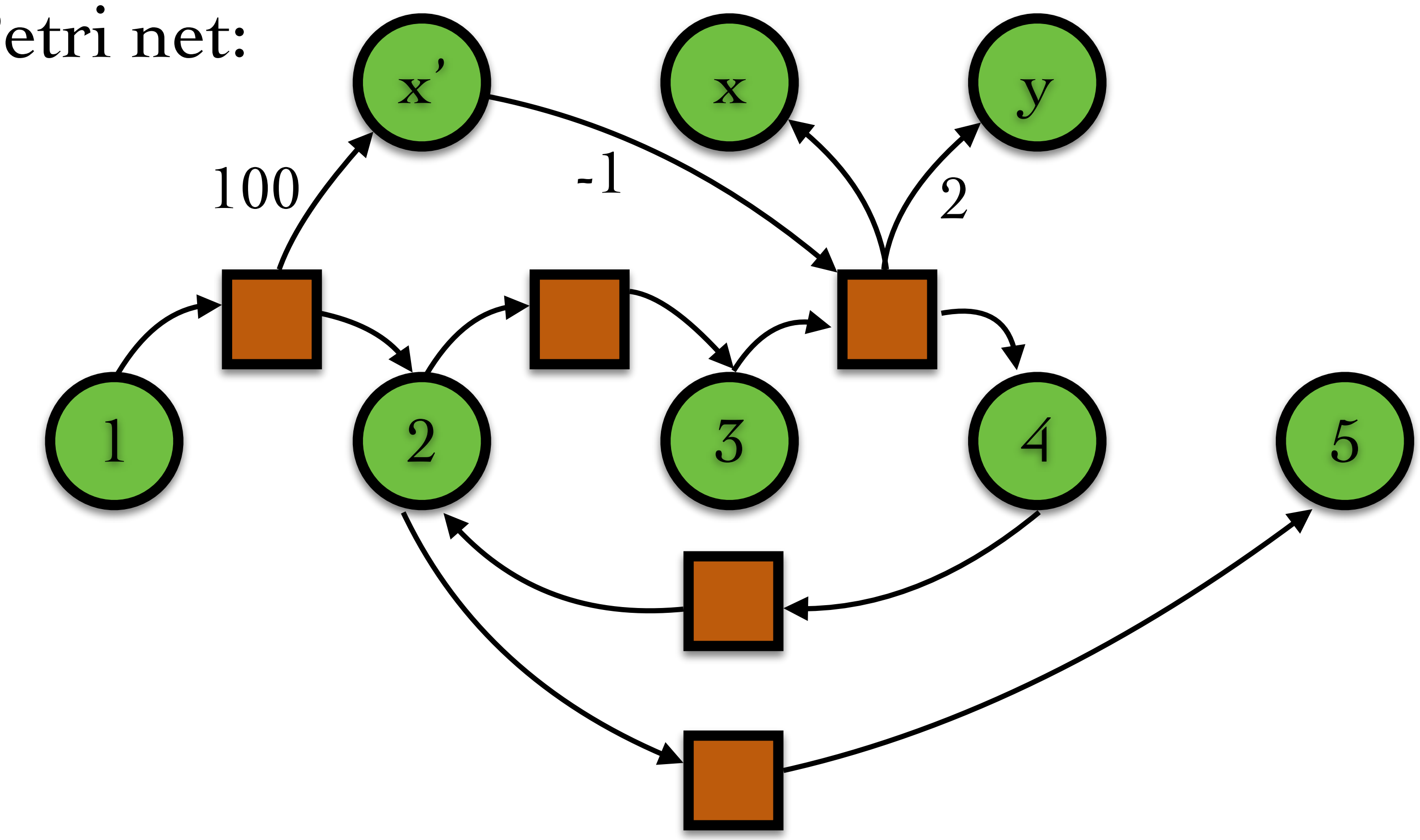


Many faces of Petri nets

counter program without zero tests:

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

Petri net:

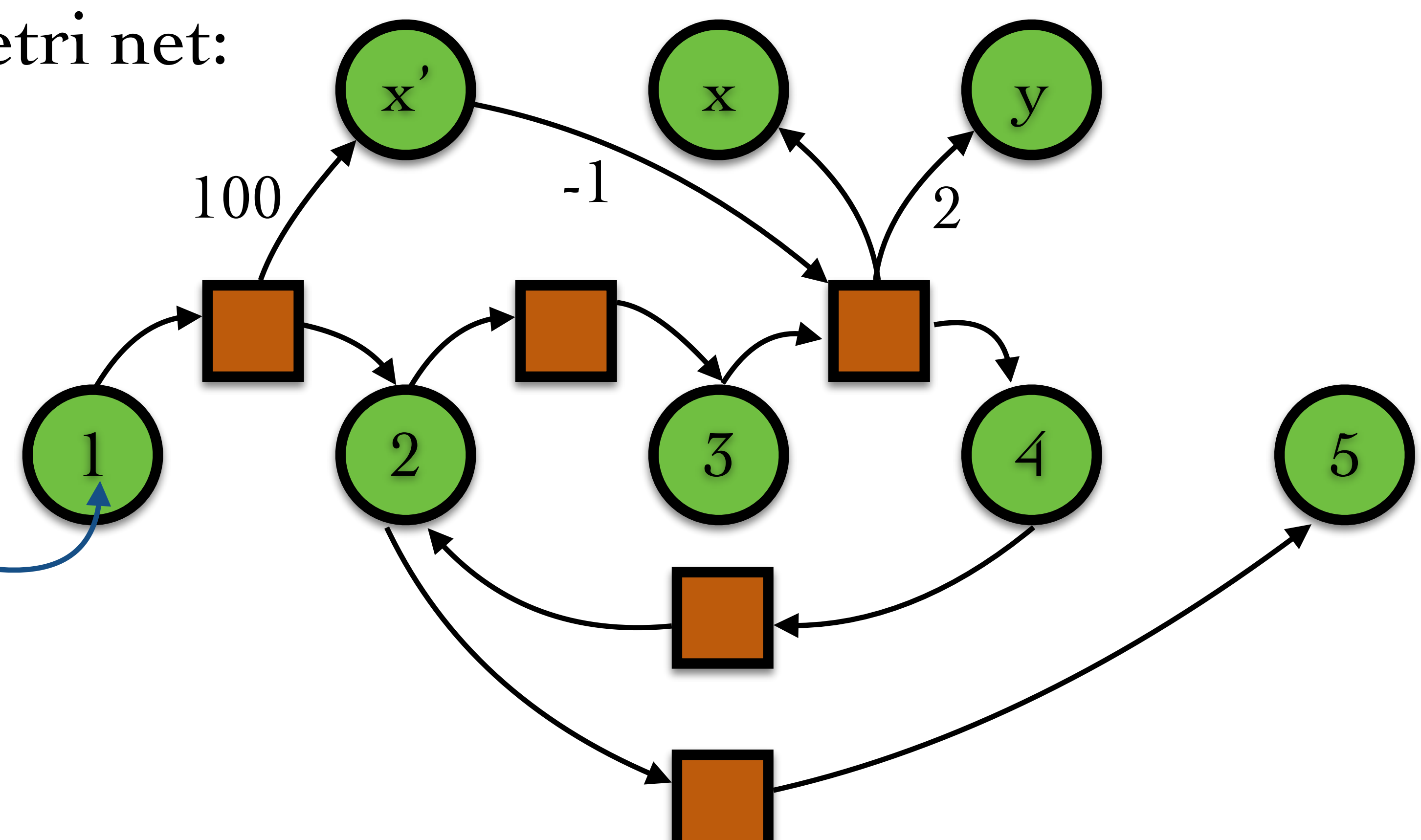


Many faces of Petri nets

counter program without zero tests:

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

Petri net:



initially one token here

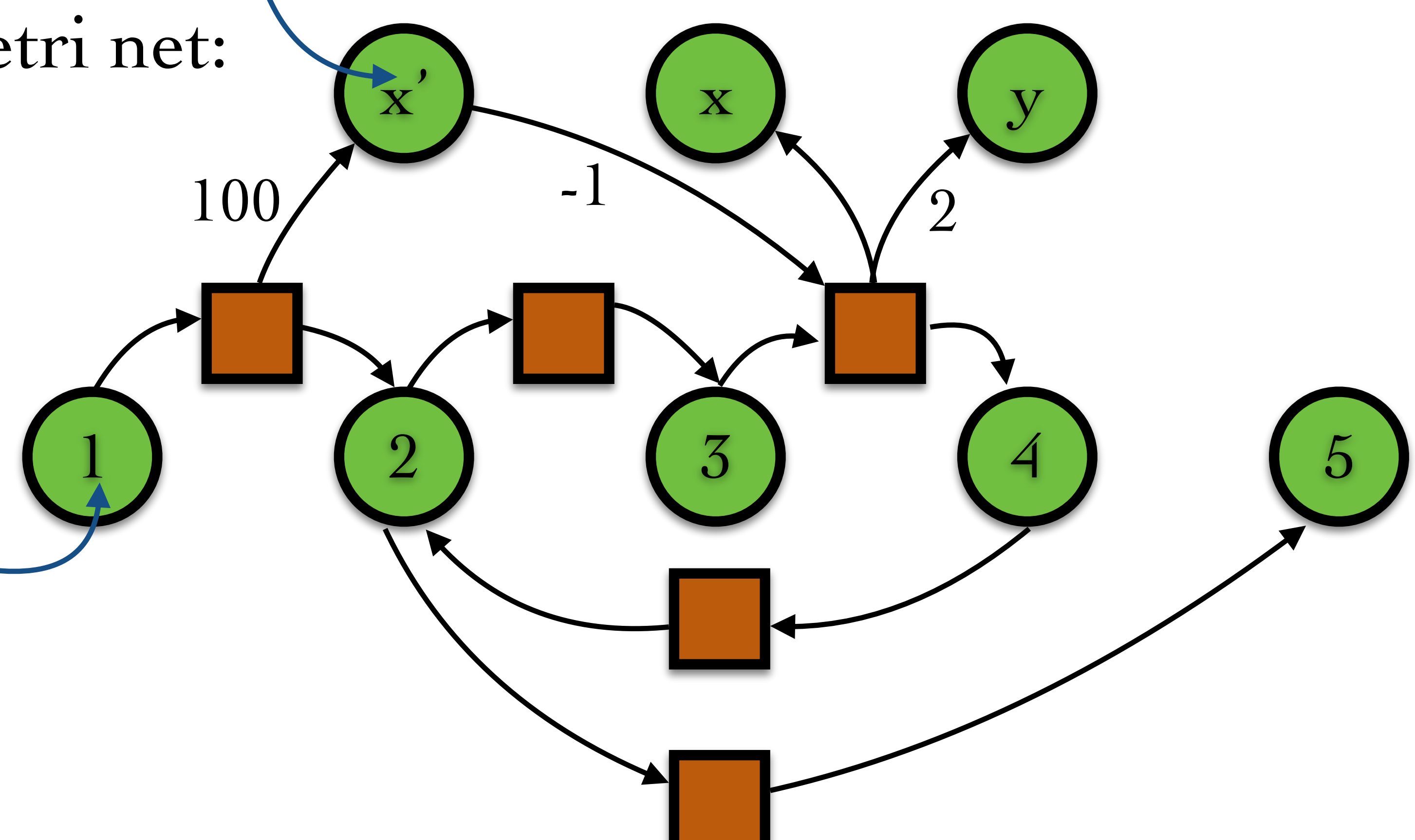
Many faces of Petri nets

counter program without zero tests:

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

halt requires no token here

Petri net:



initially one token here

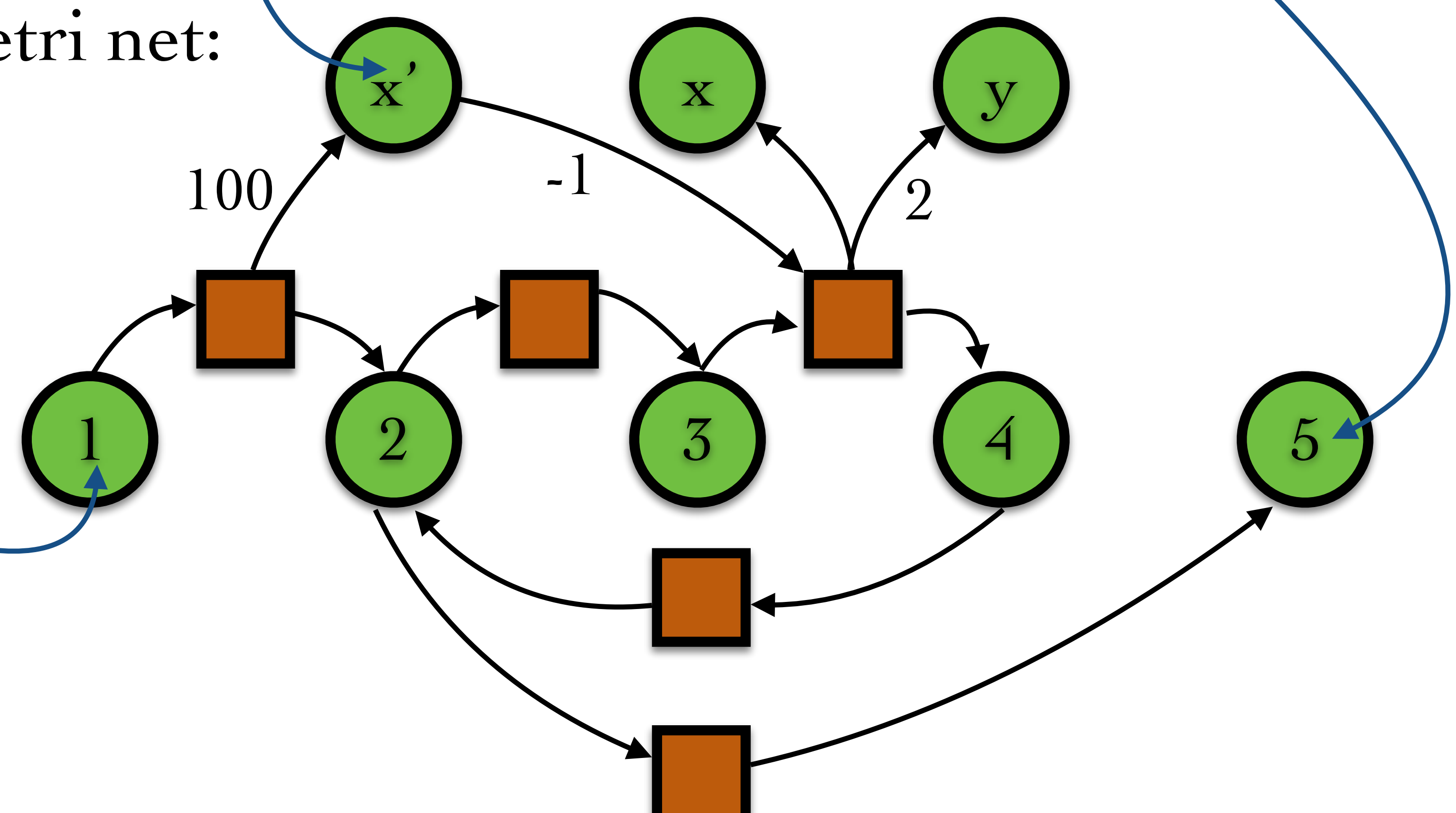
Many faces of Petri nets

counter program without zero tests:

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

halt requires no token here
and one token there

Petri net:



initially one token here

Reachability and coverability

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

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

can it terminate (execute its halt command)?

Reachability and coverability

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

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

can it terminate (execute its halt command)?

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

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

can it terminate (reach its halt command)?

Reachability and coverability

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

$$\begin{array}{ll} 1: & \mathsf{x}' \mathrel{+}= 100 \\ 2: & \textbf{goto } \boxed{5} \textbf{ or } \boxed{3} \\ 3: & \mathsf{x} \mathrel{+}= 1 \quad \mathsf{x}' \mathrel{-}= 1 \quad \mathsf{y} \mathrel{+}= 2 \\ 4: & \textbf{goto } \boxed{2} \\ 5: & \textbf{halt if } \mathsf{x}' = 0. \end{array}$$

configuration reachability

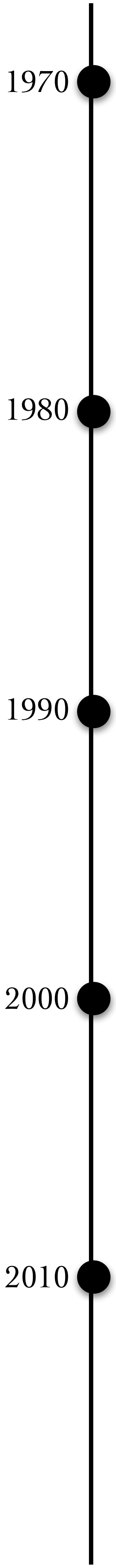
can it terminate (execute its halt command)?

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

$$\begin{array}{ll} 1: & \mathsf{x}' \mathrel{+}= 100 \\ 2: & \textbf{goto } \boxed{5} \textbf{ or } \boxed{3} \\ 3: & \mathsf{x} \mathrel{+}= 1 \quad \mathsf{x}' \mathrel{-}= 1 \quad \mathsf{y} \mathrel{+}= 2 \\ 4: & \textbf{goto } \boxed{2} \\ 5: & \textbf{halt}. \end{array}$$

control-state reachability

can it terminate (reach its halt command)?


1970
1980
1990
2000
2010

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

1970

1980

1990

2000

2010

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

1970

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

1980

1990

2000

2010

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

1970

1976 EXPSPACE lower bound **[Lipton]**

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

1980

1990

2000

2010

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

1970

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

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

1978 — EXPSPACE algorithm for coverability **[Rackoff]**

1980

1990

2000

2010

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

1970

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

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

1978 — EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 — decidability of reachability **[Mayr]**

1990

2000

2010

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

1970

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

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

1978 — EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 — decidability of reachability **[Mayr]**

1982 — decidability of reachability - simplified proof **[Kosaraju]**

1990

2000

2010

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

1970

1976 EXPSPACE lower bound **[Lipton]**

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

1978 EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 decidability of reachability **[Mayr]**

1982 decidability of reachability - simplified proof **[Kosaraju]**

1990

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

2000

2010

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

1970

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

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

1978 — EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 — decidability of reachability **[Mayr]**

1982 — decidability of reachability - simplified proof **[Kosaraju]**

1990

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

$\left.\right\}$ KLMST decomposition (1977–1992)

2000

2010

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

1970

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

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

1978 — EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 — decidability of reachability **[Mayr]**

1982 — decidability of reachability - simplified proof **[Kosaraju]**

1990

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

KLMST decomposition

2000

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

2010

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

1970

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

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

1978 — EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 — decidability of reachability **[Mayr]**

1982 — decidability of reachability - simplified proof **[Kosaraju]**

1990

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

KLMST decomposition

2000

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

2010

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

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

1970

1976 EXPSPACE lower bound **[Lipton]**

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

1978 EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 decidability of reachability **[Mayr]**

1982 decidability of reachability - simplified proof **[Kosaraju]**

KLMST decomposition

1990

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

2000

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

2010

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

2015 first upper bound F_{ω^3} **[Leroux, Schmitz]**

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

1970

1976 EXPSPACE lower bound **[Lipton]**

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

1978 EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 decidability of reachability **[Mayr]**

1982 decidability of reachability - simplified proof **[Kosaraju]**

1990

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

$\left.\begin{array}{l}1977 \ldots 1992\end{array}\right\}$ KLMST decomposition

2000

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

2010

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

2015 first upper bound F_{ω^3} **[Leroux, Schmitz]**

2019 Ackermannian upper bound F_{ω} **[Leroux, Schmitz]**

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

1970

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

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

1978 — EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 — decidability of reachability **[Mayr]**

1982 — decidability of reachability - simplified proof **[Kosaraju]**

1990

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

KLMST decomposition

2000

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

2010

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

2015 — first upper bound F_{ω^3} **[Leroux, Schmitz]**

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

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

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

1970

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

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

1978 — EXPSPACE algorithm for coverability **[Rackoff]**

1980

1981 — decidability of reachability **[Mayr]**

1982 — decidability of reachability - simplified proof **[Kosaraju]**

1990

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

} KLMST decomposition

2000

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

2010

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

2015 — first upper bound F_{ω^3} **[Leroux, Schmitz]**

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

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

} $F_3 \ldots F_\omega$ gap

TOWER lower bound

$$\text{TOWER}(n) \quad = \quad \underbrace{2^{2^{2^{\cdots^{2}}}}}_{n \text{ times}}$$

Theorem: The reachability problem for Petri nets is TOWER-hard

TOWER lower bound

$$\mathrm{TOWER}(n) \quad = \quad \underbrace{2^{2^{2^{\cdots^{2}}}}}_{n\,\text{times}}$$

Theorem: The reachability problem for Petri nets is TOWER-hard

Theorem: The reachability problem is h-EXPSPACE-hard for

TOWER lower bound

$$\text{TOWER}(n) \quad = \quad \underbrace{2^{2^{2^{\cdots^{2}}}}}_{n\,\text{times}}$$

Theorem: The reachability problem for Petri nets is TOWER-hard

Theorem: The reachability problem is h-EXPSPACE-hard for

- counter programs **without zero tests** with h+13 counters

TOWER lower bound

$$\mathrm{TOWER}(n) \quad = \quad \underbrace{2^{2^{2^{\cdots^{2}}}}}_{n \text{ times}}$$

Theorem: The reachability problem for Petri nets is TOWER-hard

Theorem: The reachability problem is h-EXPSPACE-hard for

- counter programs **without zero tests** with h+13 counters
- VASS of dimension h+13

TOWER lower bound

$$\mathrm{TOWER}(n) \quad = \quad \underbrace{2^{2^{2^{\cdots^{2}}}}}_{n \text{ times}}$$

Theorem: The reachability problem for Petri nets is TOWER-hard

Theorem: The reachability problem is h-EXPSPACE-hard for

- counter programs **without zero tests** with h+13 counters
- VASS of dimension h+13
- VAS of dimension h+16

TOWER lower bound

$$\mathrm{TOWER}(n) \quad = \quad \underbrace{2^{2^{2^{\cdots^{2}}}}}_{n\,\text{times}}$$

Theorem: The reachability problem for Petri nets is TOWER-hard

Theorem: The reachability problem is h-EXPSPACE-hard for

- counter programs **without zero tests** with h+13 counters
- VASS of dimension h+13
- VAS of dimension h+16
- Petri nets with h+16 places

Computing large numbers

[Mayr, Meyer 1981]: Petri net of size $O(n)$ can **weakly** compute

$$\mathit{Ackermann}(\mathrm{n}) = F_{\omega}(\mathrm{n}) = \mathrm{F}_{\mathrm{n}}(\mathrm{n})$$

Computing large numbers

[Mayr, Meyer 1981]: Petri net of size $O(n)$ can **weakly** compute

$$Ackermann(n) = F_\omega(n) = F_n(n)$$

[Lipton 1976]: Petri net of size $O(n^2)$ can **exactly** compute 2^{2^n}

Computing large numbers

[Mayr, Meyer 1981]: Petri net of size O(n) can **weakly** compute

$$Ackermann(\mathrm{n}) = F_\omega(\mathrm{n}) = \mathrm{F_n}(\mathrm{n})$$

[Lipton 1976]: Petri net of size O(n^2) can **exactly** compute 2^{2^n}

We prove that Petri net of size O(n) can **exactly** compute $\mathrm{TOWER}(n)$

Computing large numbers ... or long shortest runs

[Mayr, Meyer 1981]: Petri net of size O(n) can **weakly** compute

$$\mathit{Ackermann}(\mathrm{n}) = F_{\omega}(\mathrm{n}) = \mathrm{F}_{\mathrm{n}}(\mathrm{n})$$

[Lipton 1976]: Petri net of size O(n²) can **exactly** compute 2^{2^n}

has the **shortest** run of length 2^{2^n}

We prove that Petri net of size O(n) can **exactly** compute $\mathrm{TOWER}(n)$

Computing large numbers … or long shortest runs

[Mayr, Meyer 1981]: Petri net of size $O(\mathrm{n})$ can **weakly** compute

$$\mathit{Ackermann}(\mathrm{n}) = F_\omega(\mathrm{n}) = \mathrm{F}_{\mathrm{n}}(\mathrm{n})$$

[Lipton 1976]: Petri net of size $O(\mathrm{n}^2)$ can **exactly** compute 2^{2^n}

has the **shortest** run of length 2^{2^n}

We prove that Petri net of size $O(\mathrm{n})$ can **exactly** compute $\mathrm{TOWER}(n)$

has the **shortest** run of length $\mathrm{TOWER}(n)$

Computing large numbers **... or long shortest runs**

[Mayr, Meyer 1981]: Petri net of size O(n) can **weakly** compute

$$\mathit{Ackermann}(\mathrm{n}) = F_\omega(\mathrm{n}) = \mathrm{F_n}(\mathrm{n})$$

has the **longest** run of length $\mathit{Ackermann}$(n)

[Lipton 1976]: Petri net of size $O(\mathrm{n}^2)$ can **exactly** compute 2^{2^n}

has the **shortest** run of length 2^{2^n}

We prove that Petri net of size O(n) can **exactly** compute $\mathrm{TOWER}(n)$

has the **shortest** run of length $\mathrm{TOWER}(n)$

Is the lower bound relevant?

Is the lower bound relevant?

- proves reachability harder than coverability and henceforth refutes long-standing EXPSPACE-completness conjecture

Is the lower bound relevant?

- proves reachability harder than coverability and henceforth refutes long-standing EXPSPACE-completness conjecture

- plethora of problems admit reduction to/from reachability, e.g.:
    - non-emptiness of data automata
    - logics over data words
    - fragments of linear logic
    - process calculi
    - solvability of linear equations with ordered data

Is the lower bound relevant?

- proves reachability harder than coverability and henceforth refutes long-standing EXPSPACE-completness conjecture
- plethora of problems admit reduction to/from reachability, e.g.:
  - non-emptiness of data automata
  - logics over data words
  - fragments of linear logic
  - process calculi
  - solvability of linear equations with ordered data
- makes obsolete previously known TOWER lower bounds for:
  - branching VASS
  - pushdown VASS

let's embark on the proof…

Loop programs

1: $x' \mathrel{+}= 100$

2: **goto** $\boxed{5}$ **or** $\boxed{3}$

3: $x \mathrel{+}= 1 \quad x' \mathrel{-}= 1 \quad y \mathrel{+}= 2$

4: **goto** $\boxed{2}$

5: **halt if** $x' = 0$.

Loop programs

```
1: x′ += 100
2: goto 5 or 3
3: x += 1    x′ −= 1    y += 2
4: goto 2
5: halt if x′ = 0.
```

```
1: x′ += 100
2: loop
3:     x += 1    x′ −= 1    y += 2
4: halt if x′ = 0.
```

EXPSPACE lower bound for coverability

EXPSPACE lower bound for coverability

- simulation of 2^{2^n}-bounded counter machine **with zero tests**

EXPSPACE lower bound for coverability

- simulation of 2^{2^n}-bounded counter machine **with zero tests**
- subroutine $\mathbf{Dec}_n$ that $\boxed{\text{decrements a counter exactly } 2^{2^n} \text{ times}}$

EXPSPACE lower bound for coverability

- simulation of 2^{2^n}-bounded counter machine **with zero tests**
- subroutine $\mathbf{Dec}_n$ that $\boxed{\text{decrements a counter exactly } 2^{2^n} \text{ times}}$

or aborts

EXPSPACE lower bound for coverability

- simulation of 2^{2^n}-bounded counter machine **with zero tests**
- subroutine $\mathbf{Dec}_n$ that $\boxed{\text{decrements a counter exactly } 2^{2^n} \text{ times}}$
- for every simulated counter introduce a shadow counter, initiate to

$$\mathsf{x} = 0 \qquad \hat{\mathsf{x}} = 2^{2^n}$$

or aborts

EXPSPACE lower bound for coverability

- simulation of 2^{2^n}-bounded counter machine **with zero tests**
- subroutine $\mathbf{Dec}_n$ that $\boxed{\text{decrements a counter exactly } 2^{2^n} \text{ times}}$
- for every simulated counter introduce a shadow counter, initiate to

$$\mathsf{x} = 0 \qquad \hat{\mathsf{x}} = 2^{2^n}$$

or aborts

- maintain invariant

$$\mathsf{x} + \hat{\mathsf{x}} = 2^{2^n}$$

EXPSPACE lower bound for coverability

- simulation of 2^{2^n}-bounded counter machine **with zero tests**
- subroutine $\mathbf{Dec}_n$ that $\boxed{\text{decrements a counter exactly } 2^{2^n} \text{ times}}$
- for every simulated counter introduce a shadow counter, initiate to

$$\mathsf{x} = 0 \qquad \hat{\mathsf{x}} = 2^{2^n}$$

or aborts

- maintain invariant

$$\mathsf{x} + \hat{\mathsf{x}} = 2^{2^n}$$

- zero test: $\mathbf{Dec}_n\ \hat{\mathsf{x}} \quad \mathbf{Dec}_n\ \mathsf{x}$

EXPSPACE lower bound for coverability

- simulation of 2^{2^n}-bounded counter machine **with zero tests**
- subroutine $\mathbf{Dec}_n$ that $\boxed{\text{decrements a counter exactly } 2^{2^n} \text{ times}}$
- for every simulated counter introduce a shadow counter, initiate to

$$\mathsf{x} = 0 \qquad \hat{\mathsf{x}} = 2^{2^n}$$

or aborts

- maintain invariant

$$\mathsf{x} + \hat{\mathsf{x}} = 2^{2^n}$$

- zero test: $\mathbf{Dec}_n\ \hat{\mathsf{x}} \quad \mathbf{Dec}_n\ \mathsf{x}$

- how to implement $\mathbf{Dec}_n$?

Implementation of $\mathbf{Dec}_n$:

Implementation of $\mathbf{Dec}_n$:

- iterated squaring

$$\overbrace{((2^2)^2\ldots)^2}^{n\text{ times}} \quad = \quad 2^{\overbrace{2\cdot 2\cdot \ldots \cdot 2}^{n\text{ times}}} \quad = \quad 2^{2^n}$$

Implementation of $\mathbf{Dec}_n$:

- iterated squaring

$$\overbrace{((2^2)^2\ldots)^2}^{n\text{ times}} \quad = \quad 2^{\overbrace{2\cdot 2\cdot\ldots\cdot 2}^{n\text{ times}}} \quad = \quad 2^{2^n}$$

- subroutine $\mathbf{Dec}_i\ \mathsf{x}_i$ that decrements x_i exactly 2^{2^i} times, $i = 1\ldots n$

or aborts

Implementation of $\mathbf{Dec}_n$:

- iterated squaring

$$\overbrace{((2^2)^2\dots)^2}^{n\text{ times}} \quad = \quad 2^{\overbrace{2\cdot 2\cdot\dots\cdot 2}^{n\text{ times}}} \quad = \quad 2^{2^n}$$

- subroutine $\mathbf{Dec}_i\ \mathsf{x}_i$ that decrements x_i exactly 2^{2^i} times, $i = 1 \dots n$

or aborts

- the code of $\mathbf{Dec}_{i+1}\ \hat{\mathsf{x}}_{i+1}$:

```
loop
    x_i += 1    x̂_i −= 1
    loop
        y_i += 1    ŷ_i −= 1
        x̂_{i+1} −= 1    x_{i+1} += 1
    Dec_i y_i
Dec_i x_i.
```

iterated exactly $2^{2^{i+1}}$ times

EXPSPACE lower bound for coverability

EXPSPACE lower bound for coverability

- $\boxed{\text{key idea: compute exactly } 2^{2^n} \text{ due to } \textbf{iterated squaring}\text{:}}$

$$\overbrace{((2^2)^2 \ldots)^2}^{n \text{ times}} \quad = \quad 2^{\overbrace{2 \cdot 2 \cdot \ldots \cdot 2}^{n \text{ times}}} \quad = \quad 2^{2^n}$$

EXPSPACE lower bound for coverability

- $\boxed{\text{key idea: compute exactly } 2^{2^n} \text{ due to } \textbf{iterated squaring}\text{:}}$

$$\overbrace{((2^2)^2\ldots)^2}^{n\,\text{times}} \quad = \quad 2^{\overbrace{2\cdot 2\cdot\ldots\cdot 2}^{n\,\text{times}}} \quad = \quad 2^{2^n}$$

- simulation of 2^{2^n}-bounded counter program **with zero tests**

EXPSPACE lower bound for coverability

- $\boxed{\text{key idea: compute exactly } 2^{2^n} \text{ due to } \textbf{iterated squaring}\text{:}}$

$$\overbrace{((2^2)^2 \ldots)^2}^{n \text{ times}} \quad = \quad 2^{\overbrace{2 \cdot 2 \cdot \ldots \cdot 2}^{n \text{ times}}} \quad = \quad 2^{2^n}$$

- simulation of 2^{2^n}-bounded counter program **with zero tests**

TOWER lower bound for reachability

EXPSPACE lower bound for coverability

- $\boxed{\text{key idea: compute exactly } 2^{2^n} \text{ due to } \textbf{iterated squaring}\text{:}}$

$$\overbrace{((2^2)^2\ldots)^2}^{n\text{ times}} \quad = \quad 2^{\overbrace{2\cdot 2\cdot\ldots\cdot 2}^{n\text{ times}}} \quad = \quad 2^{2^n}$$

- simulation of 2^{2^n}-bounded counter program **with zero tests**

TOWER lower bound for reachability

- $\boxed{\text{key idea: compute a pair of numbers with ratio } 3!^n \text{ due to } \textbf{iterated factorial}\text{:}}$

$$3!^n \quad = \quad \overbrace{((3!)!\ldots)!}^{n\text{ times}}$$

EXPSPACE lower bound for coverability

- $\boxed{\text{key idea: compute exactly } 2^{2^n} \text{ due to } \textbf{iterated squaring}\text{:}}$

$$\overbrace{((2^2)^2\ldots)^2}^{n\text{ times}} \quad = \quad 2^{\overbrace{2\cdot 2\cdot\ldots\cdot 2}^{n\text{ times}}} \quad = \quad 2^{2^n}$$

- simulation of 2^{2^n}-bounded counter program **with zero tests**

TOWER lower bound for reachability

- $\boxed{\text{key idea: compute a pair of numbers with ratio } 3!^n \text{ due to } \textbf{iterated factorial}\text{:}}$

$$3!^n \quad = \quad \overbrace{((3!)!\ldots)!}^{n\text{ times}}$$

- simulation of $3!^n$-bounded counter program **with zero tests**

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R$$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$
with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$$ ratio R

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \text{ ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

Let R - fixed positive integer.

Suppose some *3* counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \quad \text{ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

The idea:

loop

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

$\quad \mathsf{d} \mathrel{-}= 1$

$\mathsf{c} \mathrel{-}= 1.$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \text{ ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

The idea:

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$

loop

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

$\quad \mathsf{d} \mathrel{-}= 1$

$\mathsf{c} \mathrel{-}= 1.$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

Let R - fixed positive integer.

Suppose some *3* counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \text{ ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

The idea:

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$

loop
  $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
  $\mathsf{d} \mathrel{-}= 1$
$\mathsf{c} \mathrel{-}= 1.$

at most R iterations

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \text{ ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

The idea:

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$

$$\left.\begin{array}{l}
\textbf{loop} \\
\quad \mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1 \\
\quad \mathsf{d} \mathrel{-}= 1 \\
\mathsf{c} \mathrel{-}= 1.
\end{array}\right\} \text{at most } R \text{ iterations}$$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \text{ ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

The idea:

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$ forward invariant

loop
$\quad \mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
$\quad \mathsf{d} \mathrel{-}= 1$
$\mathsf{c} \mathrel{-}= 1.$

at most R iterations

Using ratio R to simulate R-bounded counter program $\mathcal{P}$
with zero tests

Let R - fixed positive integer.

Suppose some *3* counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \quad \text{ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

The idea:

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$ forward invariant

loop

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

$\quad \mathsf{d} \mathrel{-}= 1$

$\mathsf{c} \mathrel{-}= 1.$

at most R iterations

$\mathsf{d} = \mathsf{c} \cdot R$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$
with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$$ ratio R

How to simulate R-bounded counter program **with zero tests**?

The idea:

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$ forward invariant

loop
$\quad \mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
$\quad \mathsf{d} \mathrel{-}= 1$
$\mathsf{c} \mathrel{-}= 1.$

at most R iterations

$\mathsf{d} = \mathsf{c} \cdot R$

implied by **halt if** $\ldots, \mathsf{d} = 0.$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$
with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \quad \text{ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

The idea:

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$ forward invariant

loop
$\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
$\mathsf{d} \mathrel{-}= 1$
$\mathsf{c} \mathrel{-}= 1.$

exactly ~~at most~~ R iterations

$\mathsf{d} = \mathsf{c} \cdot R$

implied by **halt if** $\ldots, \mathsf{d} = 0.$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \quad \text{ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

The idea:

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$ forward invariant

$$\left.\begin{array}{l} \textbf{loop} \\ \quad \mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1 \\ \quad \boxed{\begin{array}{l} \mathsf{d} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1. \end{array}} \end{array}\right\} \text{exactly } \cancel{\text{at most}}\ R \text{ iterations}$$

$\mathsf{d} = \mathsf{c} \cdot R$ backward invariant

implied by **halt if** $\ldots, \mathsf{d} = 0.$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$
with zero tests

Let R - fixed positive integer.

Suppose some 3 counters b, c, d are initially set **nondeterministically** to:

$$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R} \quad \text{ratio } R$$

How to simulate R-bounded counter program **with zero tests**?

The idea:

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} > \mathsf{c} \cdot R$ forward invariant

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$ forward invariant

loop

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

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

$\mathsf{c} \mathrel{-}= 1.$

exactly ~~at most~~ R iterations

$\mathsf{d} = \mathsf{c} \cdot R$ backward invariant

implied by **halt if** $\ldots, \mathsf{d} = 0.$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$

with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$

with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them <u>to at most R</u>:

loop

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

$\quad \mathsf{d} \mathrel{-}= 1 \quad \boxed{\mathsf{b} \mathrel{-}= 1}$

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

(only for zero-tested counters)

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

loop
    $\hat{\mathsf{x}} \mathrel{+}= 1 \quad \hat{\mathsf{y}} \mathrel{+}= 1 \quad \ldots$
    $\mathsf{d} \mathrel{-}= 1 \quad \boxed{\mathsf{b} \mathrel{-}= 1}$
$\mathsf{c} \mathrel{-}= 1$

(only for zero-tested counters)

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

$$\begin{array}{l} \textbf{loop} \\ \quad \hat{\mathsf{x}} \mathrel{+}= 1 \quad \hat{\mathsf{y}} \mathrel{+}= 1 \quad \ldots \\ \quad \mathsf{d} \mathrel{-}= 1 \quad \boxed{\mathsf{b} \mathrel{-}= 1} \\ \mathsf{c} \mathrel{-}= 1 \end{array}$$

(only for zero-tested counters)

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

$$\begin{array}{l} \textbf{loop} \\ \quad \hat{\mathsf{x}} \mathrel{+}= 1 \quad \hat{\mathsf{y}} \mathrel{+}= 1 \quad \ldots \\ \quad \mathsf{d} \mathrel{-}= 1 \quad \boxed{\mathsf{b} \mathrel{-}= 1} \\ \mathsf{c} \mathrel{-}= 1 \end{array}$$

(only for zero-tested counters)

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
- **zero?** x replace by

$$\begin{array}{l} \textbf{loop} \\ \quad \mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1 \\ \quad \mathsf{d} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \\ \textbf{loop} \\ \quad \mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1 \\ \quad \mathsf{d} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \end{array}$$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$

with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

```
loop
    x̂ += 1    ŷ += 1    . . .
    d −= 1    b −= 1
c −= 1
```

(only for zero-tested counters)

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
- **zero?** x replace by

```
loop
    x += 1    x̂ −= 1
    d −= 1
c −= 1
loop
    x −= 1    x̂ += 1
    d −= 1
c −= 1
```

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

```
loop
    x̂ += 1    ŷ += 1    ...
    d −= 1    b −= 1
c −= 1
```

(only for zero-tested counters)

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
- **zero?** x replace by

```
loop
    x += 1    x̂ −= 1
    d −= 1
c −= 1
loop
    x −= 1    x̂ += 1
    d −= 1
c −= 1
```

forward invariant

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

$$\begin{array}{l} \textbf{loop} \\ \quad \hat{\mathsf{x}} \mathrel{+}= 1 \quad \hat{\mathsf{y}} \mathrel{+}= 1 \quad \ldots \\ \quad \mathsf{d} \mathrel{-}= 1 \quad \mathsf{b} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \end{array}$$

(only for zero-tested counters)

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
- **zero?** x replace by

$$\begin{array}{l} \textbf{loop} \\ \quad \mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1 \\ \quad \mathsf{d} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \\ \textbf{loop} \\ \quad \mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1 \\ \quad \mathsf{d} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \end{array}$$

forward invariant

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$

- extend halt: **halt if** $\ldots, \mathsf{d} = 0.$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$ with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

$$\begin{array}{l} \textbf{loop} \\ \quad \hat{\mathsf{x}} \mathrel{+}= 1 \quad \hat{\mathsf{y}} \mathrel{+}= 1 \quad \ldots \\ \quad \mathsf{d} \mathrel{-}= 1 \quad \mathsf{b} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \end{array}$$

(only for zero-tested counters)

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
- **zero?** x replace by

$$\begin{array}{l} \textbf{loop} \\ \quad \mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1 \\ \quad \mathsf{d} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \\ \textbf{loop} \\ \quad \mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1 \\ \quad \mathsf{d} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \end{array}$$

forward invariant

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$

- extend halt: **halt if** $\ldots, \mathsf{d} = 0.$

backward invariant

$\mathsf{d} = \mathsf{c} \cdot R$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$

with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

$$\left.\begin{array}{l} \textbf{loop} \\ \quad \hat{\mathsf{x}} \mathrel{+}= 1 \quad \hat{\mathsf{y}} \mathrel{+}= 1 \quad \cdots \\ \quad \boxed{\mathsf{d} \mathrel{-}= 1 \quad \mathsf{c} \mathrel{-}= 1} \quad \boxed{\mathsf{b} \mathrel{-}= 1} \end{array}\right\} \text{exactly } R \text{ iterations}$$

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
- **zero?** x replace by

$$\left.\begin{array}{l} \textbf{loop} \\ \quad \mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1 \\ \quad \boxed{\mathsf{d} \mathrel{-}= 1 \quad \mathsf{c} \mathrel{-}= 1} \end{array}\right\} \text{exactly } R \text{ iterations}$$

$$\left.\begin{array}{l} \textbf{loop} \\ \quad \mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1 \\ \quad \boxed{\mathsf{d} \mathrel{-}= 1 \quad \mathsf{c} \mathrel{-}= 1} \end{array}\right\} \text{exactly } R \text{ iterations}$$

- extend halt: **halt if** $\ldots, \mathsf{d} = 0.$

forward invariant

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$

backward invariant

$\mathsf{d} = \mathsf{c} \cdot R$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$

with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

loop
$\hat{\mathsf{x}} \mathrel{+}= 1 \quad \hat{\mathsf{y}} \mathrel{+}= 1 \quad \cdots$
$\boxed{\mathsf{d} \mathrel{-}= 1 \quad \mathsf{c} \mathrel{-}= 1} \quad \boxed{\mathsf{b} \mathrel{-}= 1}$

exactly R iterations

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
- **zero?** x replace by

loop
$\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
$\boxed{\mathsf{d} \mathrel{-}= 1 \quad \mathsf{c} \mathrel{-}= 1}$

exactly R iterations

loop
$\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
$\boxed{\mathsf{d} \mathrel{-}= 1 \quad \mathsf{c} \mathrel{-}= 1}$

exactly R iterations

forward invariant
$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$

violation punished at the end

backward invariant
$\mathsf{d} = \mathsf{c} \cdot R$

- extend halt: **halt if** $\ldots, \mathsf{d} = 0.$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$

with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

  loop
  $\hat{\mathsf{x}} \mathrel{+}= 1 \quad \hat{\mathsf{y}} \mathrel{+}= 1 \quad \cdots$
  $\boxed{\mathsf{d} \mathrel{-}= 1 \quad \mathsf{c} \mathrel{-}= 1} \quad \boxed{\mathsf{b} \mathrel{-}= 1}$

  exactly R iterations

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
- **zero?** x replace by

  loop
  $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
  $\boxed{\mathsf{d} \mathrel{-}= 1 \quad \mathsf{c} \mathrel{-}= 1}$

  exactly R iterations

  loop
  $\mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1$
  $\boxed{\mathsf{d} \mathrel{-}= 1 \quad \mathsf{c} \mathrel{-}= 1}$

  exactly R iterations

- extend halt: **halt if** $\ldots, \mathsf{d} = 0.$

forward invariant
$\mathsf{x} + \hat{\mathsf{x}} \le R$ and $\mathsf{d} \ge \mathsf{c} \cdot R$

violation punished at the end

backward invariant
$\mathsf{d} = \mathsf{c} \cdot R$

Using ratio R to simulate R-bounded counter program $\mathcal{P}$

with zero tests

$\boxed{\mathsf{b} = R \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R}$

- introduce shadow counters and initiate them to at most R:

$$\left.\begin{array}{l} \textbf{loop} \\ \quad \hat{\mathsf{x}} \mathrel{+}= 1 \quad \hat{\mathsf{y}} \mathrel{+}= 1 \quad \cdots \\ \quad \mathsf{d} \mathrel{-}= 1 \quad \mathsf{b} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \end{array}\right\} \text{exactly } R \text{ iterations}$$

- $\mathsf{x} \mathrel{+}= 1$ replace by $\mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1$
- $\mathsf{x} \mathrel{-}= 1$ replace by $\mathsf{x} \mathrel{-}= 1$
- **zero?** x replace by

the construction doesn't depend on R

forward invariant

$\mathsf{x} + \hat{\mathsf{x}} \leq R$ and $\mathsf{d} \geq \mathsf{c} \cdot R$

$$\left.\begin{array}{l} \textbf{loop} \\ \quad \mathsf{x} \mathrel{+}= 1 \quad \hat{\mathsf{x}} \mathrel{-}= 1 \\ \quad \mathsf{d} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \end{array}\right\} \text{exactly } R \text{ iterations}$$

$$\left.\begin{array}{l} \textbf{loop} \\ \quad \mathsf{x} \mathrel{-}= 1 \quad \hat{\mathsf{x}} \mathrel{+}= 1 \\ \quad \mathsf{d} \mathrel{-}= 1 \\ \mathsf{c} \mathrel{-}= 1 \end{array}\right\} \text{exactly } R \text{ iterations}$$

violation punished at the end

backward invariant

$\mathsf{d} = \mathsf{c} \cdot R$

- extend halt: **halt if** $\ldots, \mathsf{d} = 0$.

Computing and using ratio

$$\mathcal{A} \triangleright \mathcal{P}$$

Computing and using ratio

$$\mathcal{A} \triangleright \mathcal{P}$$

R-bounded counter program **with zero tests**
which is simulated using ratio R

Computing and using ratio

$$\mathcal{A} \triangleright \mathcal{P}$$

counter program **without zero tests** that computes ratio R

R-bounded counter program **with zero tests** which is simulated using ratio R

Computing and using ratio

$$\overbrace{\mathcal{A} \triangleright \mathcal{P}}^{\text{counter program } \textbf{without zero tests}}$$

counter program **without zero tests** that computes ratio R

R-bounded counter program **with zero tests** which is simulated using ratio R

Computing and using ratio

$$\overbrace{\mathcal{A} \triangleright \mathcal{P}}^{\substack{\text{counter program}\\ \textbf{without zero tests}}}$$

counter program **without zero tests**
that computes ratio R

R-bounded counter program **with zero tests**
which is simulated using ratio R

merged halts of $\mathcal{A}$ and $\mathcal{P}$

- extend halt: **halt if** $\ldots, \mathsf{d} = 0.$

How to compute ratio?

$$3!^n = \overbrace{((3!)!\ldots)!}^{n\ \text{times}}$$

How to compute ratio?

$$3!^n = \overbrace{((3!)!\ldots)!}^{n\text{ times}}$$

- ratio 3:

```
b += 3
c += 1    d += 3
loop
      c += 1    d += 3
halt.
```

How to compute ratio?

$$3!^n \quad = \quad \overbrace{((3!)!\ldots)!}^{n\text{ times}}$$

- ratio 3:

```
b += 3
c += 1    d += 3
loop
    c += 1    d += 3
halt.
```

- we define a counter program that, using ratio R, computes ratio $R!$

factorial amplifier

How to compute ratio?

$$3!^n = \overbrace{((3!)!\ldots)!}^{n\text{ times}}$$

- ratio 3:

```
b += 3
c += 1    d += 3
loop
     c += 1    d += 3
halt.
```

hece the amplifier can use *R*-bounded zero-tested counters

- we define a counter program that, using ratio R, computes ratio $R!$

factorial amplifier

How to compute ratio?

$$3!^n = \overbrace{((3!)!\ldots)!}^{n \text{ times}}$$

- ratio 3:

```
1: b += 3
2: c += 1    d += 3
3: loop
4:     c += 1    d += 3
5: halt.
```

hece the amplifier can use
R-bounded zero-tested
counters

- we define a counter program that, using ratio R, computes ratio $R!$

factorial amplifier

- and self-compose it sufficiently many times: $\overbrace{((\mathcal{A}_3 \triangleright \mathcal{F}) \triangleright \mathcal{F}) \triangleright \cdots \mathcal{F}}^{n \text{ compositions}}$

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio $R!$

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio R!

$$\frac{2}{1} \quad \cdot \quad \frac{3}{2} \quad \cdot \quad \ldots \quad \cdot \quad \frac{R}{R-1} \quad = \quad R$$

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio $R!$

$$\frac{2}{1} \quad \cdot \quad \frac{3}{2} \quad \cdot \quad \ldots \quad \cdot \quad \frac{R}{R-1} \quad = \quad R$$

```
i += 1    x += 1    y += 1
loop
    x += 1    y += 1
loop
    loop
        x −= i    x′ += i + 1
    loop
        x′ −= 1    x += 1
    i += 1
zero? î
loop
    x −= i    y −= 1
halt if y = 0
```

a zero test

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio $R!$

$$\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1} = R$$

```
i += 1    x += 1    y += 1
loop
    x += 1    y += 1
loop
    loop
        x -= i    x' += i + 1
    loop
        x' -= 1    x += 1
    i += 1
zero? î
loop
    x -= i    y -= 1
halt if y = 0
```

initially equal R

a zero test

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio $R!$

$$\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1} = R$$

```
i += 1    x += 1    y += 1
loop
    x += 1    y += 1
loop
    loop
        x -= i    x' += i + 1
    loop
        x' -= 1    x += 1
    i += 1
zero? i
loop
    x -= i    y -= 1
halt if y = 0
```

initially equal R

a zero test

```
loop
    i -= 1    i' += 1    x -= 1
zero? i
loop
    i' -= 1    i += 1
zero? i'
```

further zero tests

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio $R!$

$$\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1} = R$$

```
i += 1    x += 1    y += 1
loop
    x += 1    y += 1
loop
    loop
        x −= i    x′ += i + 1
    loop
        x′ −= 1    x += 1
    i += 1
zero? î
loop
    x −= i    y −= 1
halt if y = 0
```

initially equal R

a zero test

```
loop
    i −= 1    i′ += 1    x −= 1
zero? i
loop
    i′ −= 1    i += 1
zero? i′
```

further zero tests

```
x′ += 1
loop
    i −= 1    i′ += 1    x′ += 1
zero? i
loop
    i′ −= 1    i += 1
zero? i′
```

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio $R!$

$$\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1} = R$$

```
i += 1    x += 1    y += 1
loop                    nondeterministic init
    x += 1    y += 1
loop
    loop
        x −= i    x′ += i + 1
    loop
        x′ −= 1    x += 1
    i += 1
zero? î
loop
    x −= i    y −= 1
halt if y = 0
```

initially equal R

a zero test

```
loop
    i −= 1    i′ += 1    x −= 1
zero? i
loop
    i′ −= 1    i += 1
zero? i′
```

further zero tests

```
x′ += 1
loop
    i −= 1    i′ += 1    x′ += 1
zero? i
loop
    i′ −= 1    i += 1
zero? i′
```

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio $R!$

$$\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1} = R$$

```
i += 1    x += 1    y += 1
loop            nondeterministic init
    x += 1    y += 1
loop
    loop
        x -= i    x' += i + 1     weak multiplication by (i+1)/i
    loop
        x' -= 1    x += 1
    i += 1
zero? i
loop
    x -= i    y -= 1
halt if y = 0
```

initially equal R

a zero test

```
loop
    i -= 1    i' += 1    x -= 1
zero? i
loop
    i' -= 1    i += 1
zero? i'
```

further zero tests

```
x' += 1
loop
    i -= 1    i' += 1    x' += 1
zero? i
loop
    i' -= 1    i += 1
zero? i'
```

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio $R!$

$$\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1} = R$$

```
i += 1    x += 1    y += 1
loop
    x += 1    y += 1
loop
    loop
        x -= i    x' += i + 1
    loop
        x' -= 1    x += 1
    i += 1
zero? i
loop
    x -= i    y -= 1
halt if y = 0
```

nondeterministic init

weak multiplication by $\frac{i+1}{i}$

initially equal R

a zero test

tests if $x \geq y \cdot R$

```
loop
    i -= 1    i' += 1    x -= 1
zero? i
loop
    i' -= 1    i += 1
zero? i'
```

further zero tests

```
x' += 1
loop
    i -= 1    i' += 1    x' += 1
zero? i
loop
    i' -= 1    i += 1
zero? i'
```

Factorial amplifier - the idea

counter program that, using ratio R, computes ratio $R!$

$$\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1} = R$$

```
i += 1    x += 1    y += 1
loop
    x += 1    y += 1
loop
    loop
        x -= i    x' += i + 1
    loop
        x' -= 1    x += 1
    i += 1
zero? î
loop
    x -= i    y -= 1
halt if y = 0
```

nondeterministic init

exact ~~weak~~ multiplication by $\frac{i+1}{i}$

initially equal R

a zero test

tests if $x \geq y \cdot R$

```
loop
    i -= 1    i' += 1    x -= 1
zero? i
loop
    i' -= 1    i += 1
zero? i'
```

further zero tests

```
x' += 1
loop
    i -= 1    i' += 1    x' += 1
zero? i
loop
    i' -= 1    i += 1
zero? i'
```

Factorial amplifier - the idea

counter program th
ratio R

fine, but where is the factorial?

$$\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1} = R$$

```
i += 1    x += 1    y += 1
loop
    x += 1    y += 1
loop
    loop
        x −= i    x′ += i + 1
    loop
        x′ −= 1    x += 1
    i += 1
zero? i
loop
    x −= i    y −= 1
halt if y = 0
```

nondeterministic init

initially equal R

a zero test

exact ~~weak~~ multiplication by $\frac{i+1}{i}$

tests if $x \geq y \cdot R$

```
loop
    i −= 1    i′ += 1    x −= 1
zero? i
loop
    i′ −= 1    i += 1
zero? i′
```

further zero tests

```
x′ += 1
loop
    i −= 1    i′ += 1    x′ += 1
zero? i
loop
    i′ −= 1    i += 1
zero? i′
```

Factorial amplifier - the idea

counter program th
ratio R

fine, but where is the factorial?

$$\boxed{\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1}} = R$$

```
i += 1   x += 1   y += 1
loop
    x += 1   y += 1
loop
    loop
        x −= i   x′ += i + 1
    loop
        x′ −= 1   x += 1
    i += 1
zero? î
loop
    x −= i   y −= 1
halt if y = 0
```

nondeterministic init

exact ~~weak~~ multiplication by $\frac{i+1}{i}$

initially equal R

a zero test

tests if $x \geq y \cdot R$

```
loop
    i −= 1   i′ += 1   x −= 1
zero? i
loop
    i′ −= 1   i += 1
zero? i′
```

further zero tests

```
x′ += 1
loop
    i −= 1   i′ += 1   x′ += 1
zero? i
loop
    i′ −= 1   i += 1
zero? i′
```

Factorial amplifier $\mathsf{b} = R! \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R!$

counter program that, using ratio R, computes ratio $R!$

Factorial amplifier $b = R! \quad c > 0 \quad d = c \cdot R!$

counter program that, using ratio R, computes ratio $R!$

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

```
loop
    b -= 1   b' += 1
loop
    b' -= 1   b += 1
    <body>
```

Factorial amplifier $\mathsf{b} = R!$ $\quad \mathsf{c} > 0$ $\quad \mathsf{d} = \mathsf{c} \cdot R!$

counter program that, using ratio R, computes ratio $R!$

invariant
$\boxed{\mathsf{d} = \mathsf{c} \cdot \mathsf{b}}$

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

```
loop
    b −= 1   b′ += 1
loop
    b′ −= 1   b += 1
    <body>
```

Factorial amplifier $\mathsf{b} = R! \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R!$

counter program that, using ratio R, computes ratio $R!$

invariant
$\boxed{\mathsf{d} = \mathsf{c} \cdot \mathsf{b}}$

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

```
loop
    b −= 1   b′ += 1
loop
    b′ −= 1   b += 1
    <body>
```

Factorial amplifier $\mathsf{b} = R!$ $\quad \mathsf{c} > 0$ $\quad \mathsf{d} = \mathsf{c} \cdot R!$

counter program that, using ratio R, computes ratio $R!$

invariant
$\boxed{\mathsf{d} = \mathsf{c} \cdot \mathsf{b}}$

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

```
loop
    b −= 1    b′ += 1
loop
    b′ −= 1    b += 1
    <body>
```

weak multiplication by $\frac{\mathsf{i}+1}{\mathsf{i}}$

Factorial amplifier $\mathsf{b} = R!$ $\quad \mathsf{c} > 0$ $\quad \mathsf{d} = \mathsf{c} \cdot R!$

counter program that, using ratio R, computes ratio $R!$

invariant
$\mathsf{d} = \mathsf{c} \cdot \mathsf{b}$

```
i += 1    x += 1    y += 1    b += 1    c += 1    d += 1
loop                                        nondeterministic init
    x += 1    y += 1    c += 1    d += 1
loop
    loop
        c −= i    c′ += 1
        loop at most b times
            x −= i    d −= i    x′ += i + 1      weak multiplication by (i+1)/i
    loop
        b −= 1    b′ += i + 1
    loop
        b′ −= 1    b += 1
    loop
        c′ −= 1    c += 1
        loop at most b times
            x′ −= 1    x += 1    d += 1
    i += 1
zero? î
loop
    x −= i    y −= 1
halt if y = 0
```

tests if $\mathsf{x} \geq \mathsf{y} \cdot R$

```
loop
    b −= 1    b′ += 1
loop
    b′ −= 1    b += 1
    <body>
```

Factorial amplifier $\mathsf{b} = R!$ $\quad \mathsf{c} > 0$ $\quad \mathsf{d} = \mathsf{c} \cdot R!$

counter program that, using ratio R, computes ratio $R!$

$$\boxed{\text{invariant} \quad \mathsf{d} = \mathsf{c} \cdot \mathsf{b}}$$

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

exact ~~weak~~ multiplication by $\frac{\mathsf{i}+1}{\mathsf{i}}$

```
loop
    b −= 1    b′ += 1
loop
    b′ −= 1    b += 1
    <body>
```

tests if $\mathsf{x} \geq \mathsf{y} \cdot R$

Factorial amplifier $\mathsf{b} = R!$ $\quad \mathsf{c} > 0$ $\quad \mathsf{d} = \mathsf{c} \cdot R!$

counter program that, using ratio R, computes ratio $R!$

$\boxed{\text{invariant } \mathsf{d} = \mathsf{c} \cdot \mathsf{b}}$

```
i += 1    x += 1    y += 1    b += 1    c += 1    d += 1
loop                                          nondeterministic init
    x += 1    y += 1    c += 1    d += 1
loop
    loop      weak division by i
        c −= i    c′ += 1
        loop at most b times
            x −= i    d −= i    x′ += i + 1      exact ~~weak~~ multiplication by (i+1)/i
    loop
        b −= 1    b′ += i + 1
    loop
        b′ −= 1    b += 1
    loop
        c′ −= 1    c += 1
        loop at most b times
            x′ −= 1    x += 1    d += 1
    i += 1
zero? î
loop
    x −= i    y −= 1
halt if y = 0
```

tests if $\mathsf{x} \geq \mathsf{y} \cdot R$

```
loop
    b −= 1    b′ += 1
loop
    b′ −= 1    b += 1
    <body>
```

Factorial amplifier $b = R! \quad c > 0 \quad d = c \cdot R!$

counter program that, using ratio R, computes ratio $R!$

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

exact ~~weak~~ division by i

exact ~~weak~~ multiplication by $\frac{i+1}{i}$

tests if $x \geq y \cdot R$

invariant
$d = c \cdot b$

```
loop
    b −= 1   b′ += 1
loop
    b′ −= 1   b += 1
    <body>
```

Factorial amplifier $\mathsf{b} = R!\quad \mathsf{c} > 0\quad \mathsf{d} = \mathsf{c}\cdot R!$

counter program that, using ratio R, computes ratio $R!$

invariant
$\mathsf{d} = \mathsf{c}\cdot\mathsf{b}$

```
i += 1    x += 1    y += 1    b += 1    c += 1    d += 1
loop                                        nondeterministic init
    x += 1    y += 1    c += 1    d += 1
loop
    loop                 exact ~~weak~~ division by i
        c −= i    c′ += 1
        loop at most b times
            x −= i    d −= i    x′ += i + 1      exact ~~weak~~ multiplication by (i+1)/i
    loop
        b −= 1    b′ += i + 1                    weak multiplication by i+1
    loop
        b′ −= 1    b += 1
    loop
        c′ −= 1    c += 1
        loop at most b times
            x′ −= 1    x += 1    d += 1
    i += 1
zero? î
loop
    x −= i    y −= 1
halt if y = 0
```

tests if $\mathsf{x} \geq \mathsf{y}\cdot R$

```
loop
    b −= 1    b′ += 1
loop
    b′ −= 1    b += 1
    <body>
```

Factorial amplifier $\mathsf{b} = R! \quad \mathsf{c} > 0 \quad \mathsf{d} = \mathsf{c} \cdot R!$

counter program that, using ratio R, computes ratio $R!$

invariant $\boxed{\mathsf{d} = \mathsf{c} \cdot \mathsf{b}}$

```
i += 1    x += 1    y += 1    b += 1    c += 1    d += 1
loop                                        nondeterministic init
    x += 1    y += 1    c += 1    d += 1
loop
    loop                  exact ~~weak~~ division by i
        c −= i    c′ += 1
        loop at most b times
            x −= i    d −= i    x′ += i + 1      exact ~~weak~~ multiplication by (i+1)/i
    loop                                          exact ~~weak~~ multiplication by i+1
        b −= 1    b′ += i + 1
    loop
        b′ −= 1    b += 1
    loop
        c′ −= 1    c += 1
        loop at most b times
            x′ −= 1    x += 1    d += 1
    i += 1
zero? î
loop
    x −= i    y −= 1
halt if y = 0
```

tests if $\mathsf{x} \geq \mathsf{y} \cdot R$

```
loop
    b −= 1    b′ += 1
loop
    b′ −= 1    b += 1
    <body>
```

Factorial amplifier $b = R!$ $\quad c > 0 \quad d = c \cdot R!$

counter program that, using ratio R, computes ratio $R!$

```
i += 1   x += 1   y += 1   b += 1   c += 1   d += 1
loop                                   nondeterministic init
    x += 1   y += 1   c += 1   d += 1
loop
    loop            exact ~~weak~~ division by i
        c -= i   c' += 1
        loop at most b times
            x -= i   d -= i   x' += i + 1      exact ~~weak~~ multiplication by (i+1)/i
    loop                                       exact ~~weak~~ multiplication by i+1
        b -= 1   b' += i + 1
    loop
        b' -= 1   b += 1
    loop
        c' -= 1   c += 1
        loop at most b times
            x' -= 1   x += 1   d += 1
    i += 1
zero? î
loop
    x -= i   y -= 1
halt if y = 0
```

tests if $x \geq y \cdot R$

```
loop
    b -= 1   b' += 1
loop
    b' -= 1   b += 1
    <body>
```

invariant
$\boxed{d = c \cdot b}$

$\boxed{b = R!}$

Future work

Future work

- TOWER…ACKERMANN gap $\qquad$ F3…Fω gap

Future work

- TOWER…ACKERMANN gap $F_3 \ldots F_\omega$ gap
- improving the lower bound? TOWER amplifier?

Future work

- TOWER…ACKERMANN gap

F3…Fω gap

- improving the lower bound? TOWER amplifier?
- better lower bounds for
  - branching VASS
  - pushdown VASS
  - VASS with 1 zero test
  - VASS with hierarchical zero tests

Future work

- TOWER…ACKERMANN gap

F3…Fω gap

- improving the lower bound? TOWER amplifier?
- better lower bounds for
  - branching VASS
  - pushdown VASS
  - VASS with 1 zero test
  - VASS with hierarchical zero tests
- refined analysis for fixed dimension

Future work

- TOWER…ACKERMANN gap $\qquad$ $F_3 \ldots F_\omega$ gap
- improving the lower bound? TOWER amplifier?
- better lower bounds for
  - branching VASS
  - pushdown VASS
  - VASS with 1 zero test
  - VASS with hierarchical zero tests
- refined analysis for fixed dimension
- decidability status of reachability is open for
  - branching VASS
  - pushdown VASS
  - equality data VASS

Future work

- TOWER…ACKERMANN gap

$F_3 \ldots F_\omega$ gap

- improving the lower bound? TOWER amplifier?
- better lower bounds for
  - branching VASS
  - pushdown VASS
  - VASS with 1 zero test
  - VASS with hierarchical zero tests
- refined analysis for fixed dimension
- decidability status of reachability is open f
  - branching VASS
  - pushdown VASS
  - equality data VASS

thank you!