The reachability problem for Petri nets is not elementary

Wojciech Czerwiński
Sławomir Lasota

University of Warsaw

Jerome Leroux
Filip Mazowiecki

Ranko Lazic

University of Warwick

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

Jerome Leroux
Filip Mazowiecki

Ranko Lazic
University of Warwick

University of Bordeaux

RP’19, Brussels, 2019.09.11
The reachability problem for Petri nets is not elementary:
crash course in counter programming (without zero tests)

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
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:

\[
\begin{array}{ll}
\text{x } & \text{+= 1} \quad \text{(increment counter x)} \\
\text{x } & \text{--= 1} \quad \text{(decrement counter x)} \\
\text{goto } L \text{ or } L' \quad \text{(jump to either line } L \text{ or line } L' \text{)} \\
\text{zero? x} \quad \text{(continue if counter } x \text{ equals 0)} \\
\end{array}
\]

counters are nonnegative
Counter programs

a sequence of commands of the form:

- $x += 1$ (increment counter $x$)
- $x -= 1$ (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:

\[
\begin{align*}
x & +:= 1 & \text{(increment counter } x) \\
x & -:= 1 & \text{(decrement counter } x) \\
goto & L \text{ or } L' & \text{(jump to either line } L \text{ or line } L') \\
zero? & x & \text{(continue if counter } x \text{ equals 0)}
\end{align*}
\]

- counters are nonnegative
- abort if \(x=0\)
- otherwise abort
Counter programs

a sequence of commands of the form:

\[
\begin{align*}
  x &\;+=\; 1 \\
  x &\;-=\; 1 \\
  \text{go to } &\;L \;\text{or}\; L' \\
  \text{zero? } &\;x
\end{align*}
\]

- (increment counter \(x\))
- (decrement counter \(x\))
- (jump to either line \(L\) or line \(L'\))
- (continue if counter \(x\) equals 0)

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

\[
\text{halt if } \;x_1, \ldots, x_i = 0
\]

(terminate provided all the listed counters are zero)

- counters are nonnegative
- abort if \(x=0\)
- otherwise abort
- otherwise abort
- otherwise abort
Counter programs

a sequence of commands of the form:

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

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

\texttt{halt if $x_1, \ldots, x_i = 0$} (terminate provided all the listed counters are zero)

Example:

1: $x' \mathinner{+=} 100$
2: \texttt{goto 5 or 3}
3: $x \mathinner{+=} 1$  \hspace{1em} $x' \mathinner{-=} 1$ \hspace{1em} $y \mathinner{+=} 2$
4: \texttt{goto 2}
5: \texttt{halt if $x' = 0$.}
Counter programs

a sequence of commands of the form:

\[
\begin{align*}
&x \;+\; 1 \quad \text{(increment counter } x) \\
&x \;\ -\; 1 \quad \text{(decrement counter } x) \\
&\text{goto } L \text{ or } L' \quad \text{(jump to either line } L \text{ or line } L') \\
&\text{zero? } x \quad \text{(continue if counter } x \text{ equals 0)}
\end{align*}
\]

counters are nonnegative

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

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

Example:

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

initially all counters 0:
\[
\begin{align*}
x' &= x = y = 0
\end{align*}
\]
## Counter programs

A sequence of commands of the form:

- $x += 1$ (increment counter $x$)
- $x -= 1$ (decrement counter $x$)
- `goto L or L'` (jump to either line $L$ or line $L'$)
- `zero? x` (continue if counter $x$ equals 0)

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

- `halt if x$_1$, ..., x$_i$ = 0` (terminate provided all the listed counters are zero)

### Example:

1: $x' += 100$
2: `goto 5 or 3`
3: $x += 1$  $x' -= 1$  $y += 2$
4: `goto 2`
5: `halt if 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:

\[
\begin{align*}
x & \;+\;= \;1 \quad \text{(increment counter } x) \\
x & \;=\; -1 \quad \text{(decrement counter } x) \\
goto \;L \;or \;L' & \quad \text{(jump to either line } L \text{ or line } L') \\
\text{zero? } x & \quad \text{(continue if counter } x \text{ equals 0)}
\end{align*}
\]

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

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

**Example:**

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

Initially all counters 0:  
\[
\begin{align*}
x' &= x = y = 0
\end{align*}
\]

Finally:  
\[
\begin{align*}
x' &= 0 \quad x = 100 \quad y = 200
\end{align*}
\]
Minsky machines

the conditional jump of Minsky machines

\[
\text{if } x = 0 \text{ then goto } L \text{ else } x -= 1
\]

is simulated by counter program with zero tests:

1: \texttt{goto 2 or 4}
2: \texttt{zero? x}
3: \texttt{goto L}
4: \texttt{x -= 1}
Many faces of Petri nets

counter program without zero tests:

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

Petri net:
counter program without zero tests:

1: $x' \text{ += } 100$
2: $\text{goto 5 or 3}$
3: $x \text{ += } 1 \quad x' \text{ -= } 1 \quad y \text{ += } 2$
4: $\text{goto 2}$
5: $\text{halt if } x' = 0.$
Many faces of Petri nets

counter program without zero tests:

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

Petri net:
Many faces of Petri nets

counter program without zero tests:

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

Petri net:
Many faces of Petri nets

counter program without zero tests:

1: $x' += 100$
2: goto 5 or 3
3: $x += 1$  $x' -= 1$  $y += 2$
4: goto 2
5: halt if $x' = 0$.  

Petri net:
counter program without zero tests:

1: \( x' += 100 \)
2: \( \text{goto 5 or 3} \)
3: \( x += 1 \quad x' -= 1 \quad y += 2 \)
4: \( \text{goto 2} \)
5: \( \text{halt if } x' = 0. \)
Many faces of Petri nets

counter program without zero tests:

1: \[ x' \leftarrow 100 \]
2: \[ \text{goto 5 or 3} \]
3: \[ x \leftarrow 1 \quad x' \leftarrow -1 \quad y \leftarrow 2 \]
4: \[ \text{goto 2} \]
5: \[ \text{halt if } x' = 0. \]

Petri net:

- Initially one token here
- Halt requires no token here
counter program without zero tests:

1: \( x' \) += 100
2: goto 5 or 3
3: \( x \) += 1 \( x' \) -= 1 \( y \) += 2
4: goto 2
5: halt if \( x' = 0 \).
Reachability and coverability

Reachability problem: given a counter program without zero tests

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

Can it terminate (execute its halt command)?
Reachability and coverability

Reachability problem: given a counter program without zero tests

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

Can it terminate (execute its halt command)?

Coverability problem: given a counter program without zero tests

with trivial halt command

1: \( x' += 100 \)
2: goto 5 or 3
3: \( x += 1 \quad x' -= 1 \quad y += 2 \)
4: goto 2
5: halt.

Can it terminate (reach its halt command)?
Reachability and coverability

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

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

Can it terminate (execute its halt command)?

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

with trivial halt command

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

Can it terminate (reach its halt command)?
decidability of coverability [Karp, Miller]
1969 — decidability of coverability [Karp, Miller]

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

EXPSPACE lower bound [Lipton]

(incomplete) decidability of reachability [Sacerdote, Tenney]
1969 — decidability of coverability [Karp, Miller]

1976 — EXPSPACE lower bound [Lipton]

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

1978 — EXPSPACE algorithm for coverability [Rackoff]
1969  decidability of coverability [Karp, Miller]
1976  EXPSPACE lower bound [Lipton]
1977  (incomplete) decidability of reachability [Sacerdote, Tenney]
1978  EXPSPACE algorithm for coverability [Rackoff]
1981  decidability of reachability [Mayr]
Decidability of coverability [Karp, Miller] (1969)

EXPSPACE lower bound [Lipton] (1976)

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

EXPSPACE algorithm for coverability [Rackoff] (1978)


1969 — decidability of coverability [Karp, Miller]

1976 — EXPSPACE lower bound [Lipton]

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

1978 — EXPSPACE algorithm for coverability [Rackoff]

1980 —

1981 — decidability of reachability [Mayr]

1982 — decidability of reachability - simplified proof [Kosaraju]

1990 —

1992 — decidability of reachability - refined data structure [Lambert]
1969  
1970  
decidability of coverability [Karp, Miller]

1976  
EXPSPACE lower bound [Lipton]

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

1978  
EXPSPACE algorithm for coverability [Rackoff]

1980  
1981  
decidability of reachability [Mayr]

1982  
decidability of reachability - simplified proof [Kosaraju]

1990  
1992  
decidability of reachability - refined data structure [Lambert]

KLMST decomposition
decidability of coverability [Karp, Miller]

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

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

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

2009 — decidability of reachability by Presburger invariants [Leroux]

KLMST decomposition
1969 — decidability of coverability [Karp, Miller]

1976 — EXPSPACE lower bound [Lipton]

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

1978 — EXPSPACE algorithm for coverability [Rackoff]

1980 —

1981 — decidability of reachability [Mayr]

1982 — decidability of reachability - simplified proof [Kosaraju]

1990 —

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

2000 —

2009 — decidability of reachability by Presburger invariants [Leroux]

2010 —

2011 — decidability of reachability without KLMST decomposition [Leroux]

KLMST decomposition
1969 — decidability of coverability [Karp, Miller]

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

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

1990 —

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

KLMST decomposition

1992 —

2000 —

2009 — decidability of reachability by Presburger invariants [Leroux]
2010 —

2011 — decidability of reachability without KLMST decomposition [Leroux]
2015 — first upper bound $F_{\omega^3}$ [Leroux, Schmitz]
1969—decidability of coverability [Karp, Miller]

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

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

1990—

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

1999—

2009—decidability of reachability by Presburger invariants [Leroux]
2010—

2011—decidability of reachability without KLMST decomposition [Leroux]

2015—first upper bound $F_{\omega^3}$ [Leroux, Schmitz]

2019—Ackermannian upper bound $F_\omega$ [Leroux, Schmitz]
1969—decidability of coverability [Karp, Miller]

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

1980—decidability of reachability [Mayr]
1981—decidability of reachability [Mayr]
1982—decidability of reachability - simplified proof [Kosaraju]

KLMST decomposition

1990—decidability of reachability - refined data structure [Lambert]

2000—

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

- decidability of coverability [Karp, Miller]
- EXPSPACE lower bound [Lipton]
- (incomplete) decidability of reachability [Sacerdote, Tenney]
- EXPSPACE algorithm for coverability [Rackoff]
- decidability of reachability [Mayr]
- decidability of reachability - simplified proof [Kosaraju]
- decidability of reachability - refined data structure [Lambert]

KLMST decomposition

- decidability of reachability by Presburger invariants [Leroux]
- decidability of reachability without KLMST decomposition [Leroux]
- first upper bound $F_{\omega^3}$ [Leroux, Schmitz]
- Ackermannian upper bound $F_{\omega}$ [Leroux, Schmitz]
- TOWER lower bound $F_3$ [Czerwiński, Lazic, Leroux, Mazowiecki]

$F_3 \ldots F_{\omega}$ gap
TOWER lower bound

\[
\text{TOWER}(n) = \underbrace{2^2 \cdots 2}_{n \text{ times}}
\]

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

\[ \text{TOWER}(n) = \underbrace{2^2 \cdots 2}_{n \text{ times}} \]

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

**Theorem:** The reachability problem is **h-EXPSPACE-hard** for
TOWER lower bound

\[ \text{TOWER}(n) = \underbrace{2^2^2 \cdots 2}_{n \text{ times}} \]

**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

\[ \text{TOWER}(n) = \underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{n \text{ times}} \]

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

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

- counter programs \textbf{without zero tests} with \( h+13 \) counters
- VASS of dimension \( h+13 \)
TOWER lower bound

\[ \text{TOWER}(n) = \underbrace{2^2 \cdots 2}_{\text{n times}} \]

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

**Theorem:** The reachability problem is \textit{h-EXPSPACE-hard} for

- counter programs \textbf{without zero tests} with \( h+13 \) counters
- VASS of dimension \( h+13 \)
- VAS of dimension \( h+16 \)
Theorem: The reachability problem for Petri nets is TOWER-hard

\[ \text{TOWER}(n) = \underbrace{2^2 \cdots 2}_{n \text{ times}} \]

Theorem: The reachability problem is h-EXPSPACE-hard for
- counter programs \textit{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

$$\text{Ackermann}(n) = F_\omega(n) = F_n(n)$$
Computing large numbers

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

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

[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 \textbf{weakly} compute

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

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

We prove that Petri net of size $O(n)$ can \textbf{exactly} compute $\text{TO} \text{W} \text{E} \text{R}(n)$
We prove that Petri net of size $O(n)$ can exactly compute $\text{Ackermann}(n) = F_\omega(n) = F_n(n)$

[Lipton 1976]: Petri net of size $O(n^2)$ 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 $\text{TOWER}(n)$
We prove that Petri net of size $O(n)$ can \textbf{exactly} compute

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

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

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

\textbf{[Lipton 1976]}: Petri net of size $O(n^2)$ can \textbf{exactly} compute $2^{2^n}$
has the \textbf{shortest} run of length $2^{2^n}$

We prove that Petri net of size $O(n)$ can \textbf{exactly} compute $\text{TOWER}(n)$
has the \textbf{shortest} run of length $\text{TOWER}(n)$
We prove that Petri net of size $O(n)$ can \textbf{exactly} compute $\text{ACK}(n) = F_\omega(n) = F_n(n)$

has the \textbf{longest} run of length $\text{ACK}(n)$

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

$$\text{ACK}(n) = F_\omega(n) = F_n(n)$$

has the \textbf{longest} run of length $\text{ACK}(n)$

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

has the \textbf{shortest} run of length $2^{2n}$

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

has the \textbf{shortest} run of length $\text{TOWER}(n)$
Is the lower bound relevant?
Is the lower bound relevant?

• proves reachability harder than coverability and henceforth refutes long-standing EXPSPACE-completeness conjecture
Is the lower bound relevant?

• proves reachability harder than coverability and henceforth refutes long-standing EXPSPACE-completeness 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 \textit{EXPSPACE}-completeness 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 \textit{TOWER} lower bounds for:
  • branching VASS
  • pushdown VASS
let’s embark on the proof...
Loop programs

1: \( x' += 100 \)
2: \textbf{goto} 5 \textbf{or} 3
3: \( x += 1 \quad x' -= 1 \quad y += 2 \)
4: \textbf{goto} 2
5: \textbf{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 $\text{Dec}_n$ that decrements a counter exactly $2^{2^n}$ times
EXPSPACE lower bound for coverability

- simulation of $2^{2^n}$-bounded counter machine with zero tests
- subroutine $\text{Dec}_n$ that decrements a counter exactly $2^{2^n}$ times or aborts
EXPSPACE lower bound for coverability

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

• subroutine $\text{Dec}_n$ that decrements a counter exactly $2^{2^n}$ times

• for every simulated counter introduce a shadow counter, initiate to

\[
\begin{align*}
x &= 0 \\
\hat{x} &= 2^{2^n}
\end{align*}
\]

or aborts
EXPSPACE lower bound for coverability

• simulation of $2^{2^n}$-bounded counter machine with zero tests
• subroutine $\text{Dec}_n$ that decrements a counter exactly $2^{2^n}$ times
• for every simulated counter introduce a shadow counter, initiate to
  $$x = 0 \quad \hat{x} = 2^{2^n}$$
• maintain invariant
  $$x + \hat{x} = 2^{2^n}$$

or aborts
**EXPSPACE lower bound for coverability**

- simulation of $2^{2^n}$-bounded counter machine with zero tests
- subroutine $\text{Dec}_n$ that decrements a counter exactly $2^{2^n}$ times
- for every simulated counter introduce a shadow counter, initiate to $x = 0$, $\hat{x} = 2^{2^n}$
- maintain invariant $x + \hat{x} = 2^{2^n}$
- zero test: $\text{Dec}_n \hat{x} \text{ Dec}_n x$

or aborts
EXPSPACE lower bound for coverability

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

• subroutine $\text{Dec}_n$ that decrements a counter exactly $2^{2^n}$ times

• for every simulated counter introduce a shadow counter, initiate to $x = 0$ $\hat{x} = 2^{2^n}$

• maintain invariant

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

• zero test: $\text{Dec}_n \hat{x}$ $\text{Dec}_n x$

• how to implement $\text{Dec}_n$?
Implementation of $\text{Dec}_n$: 
Implementation of $\text{Dec}_n$:

- iterated squaring

\[
\underbrace{\left( (2^2)^2 \ldots \right)^2}^{n \text{ times}} = \underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_n = 2^{2^n}
\]
Implementation of $\text{Dec}_n$:

- iterated squaring

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

- subroutine $\text{Dec}_i x_i$ that decrements $x_i$ exactly $2^{2^i}$ times, $i = 1 \ldots n$ or aborts
Implementation of $\text{Dec}_n$:

- iterated squaring

\[
\underbrace{(2^2)^2 \ldots}_n \cdot 2^{n-1} = 2^{n+1}
\]

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

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

```
loop
x_i += 1 \quad \hat{x}_i -= 1
```

```
loop
y_i += 1 \quad \hat{y}_i -= 1
```

```
\hat{x}_{i+1} -= 1 \quad x_{i+1} += 1
```

```
\text{Dec}_i y_i
```

```
\text{Dec}_i x_i.
```

or aborts
EXPSPACE lower bound for coverability
EXPSPACE lower bound for coverability

- key idea: compute exactly $2^{2^n}$ due to **iterated squaring**:

$$\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{n \text{ times}} = 2^{2^n}$$
EXPSPACE lower bound for coverability

- key idea: compute exactly $2^{2^n}$ due to **iterated squaring**:

\[
\underbrace{(2^2 \cdot 2 \cdot \ldots \cdot 2)}_{\text{n times}} = 2^{2^n}
\]

- simulation of $2^{2^n}$-bounded counter program with zero tests
EXPSPACE lower bound for coverability

- key idea: compute exactly $2^{2^n}$ due to **iterated squaring**:

\[
\underbrace{(\underbrace{(2^2)^2 \ldots)^2}}_{n \text{ times}} = \underbrace{2^2 \cdot 2 \cdot \ldots \cdot 2}_{n \text{ times}} = 2^{2^n}
\]

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

TOWER lower bound for reachability
EXPSPACE lower bound for coverability

- key idea: compute exactly $2^{2n}$ due to **iterated squaring**:

  $$\underbrace{(\underbrace{(2^2)^2 \ldots}^n \cdot 2}^n \cdot 2 \ldots \cdot 2 = 2^{2n}$$

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

TOWER lower bound for reachability

- key idea: compute a pair of numbers with ratio $3!^n$ due to **iterated factorial**:

  $$3!^n = \underbrace{((3!)! \ldots)!}_n$$
EXPSPACE lower bound for coverability

- key idea: compute exactly $2^{2^n}$ due to **iterated squaring**:

$$
\underbrace{((2^2)^2 \ldots)^2}_{n \text{ times}} = \underbrace{2^2 \cdot 2 \cdot \ldots \cdot 2}_{n \text{ times}} = 2^{2^n}
$$

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

TOWER lower bound for reachability

- key idea: compute a pair of numbers with ratio $3!^n$ due to **iterated factorial**:

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

- simulation of $3!^n$-bounded counter program with zero tests
Using ratio $R$ to simulate $R$-bounded counter program $P$ 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 \textbf{nondeterministically} to:

\[
    b = R \quad c > 0 \quad d = c \cdot R
\]
Using ratio $R$ to simulate $R$-bounded counter program $P$ with zero tests

Let $R$ - fixed positive integer.
Suppose some 3 counters $b$, $c$, $d$ are initially set **nondeterministically** to:

$b = R \quad c > 0 \quad d = c \cdot R$  \hspace{1cm} \text{ratio } R
Let \( R \) - fixed positive integer.
Suppose some 3 counters \( b, c, d \) are initially set \textbf{nondeterministically} to:

\[
\begin{align*}
\text{b} &= R \\
\text{c} &> 0 \\
\text{d} &= c \cdot R
\end{align*}
\]

How to simulate \( R \)-bounded counter program \textbf{with} zero tests?
Let $R$ - fixed positive integer.
Suppose some 3 counters $b$, $c$, $d$ are initially set nondeterministically to:

\[ b = R \quad c > 0 \quad d = c \cdot R \]

ratio $R$

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

The idea:

```
loop
  x += 1    \hat{x} -= 1
  d -= 1
  c -= 1.
```
Using ratio $R$ to simulate $R$-bounded counter program $P$ with zero tests

Let $R$ - fixed positive integer.
Suppose some 3 counters $b$, $c$, $d$ are initially set nondeterministically to:

\[ b = R \quad c > 0 \quad d = c \cdot R \]

ratio $R$

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

The idea:

\[ x + \hat{x} \leq R \text{ and } d \geq c \cdot R \]

loop
\[
\begin{align*}
&x \text{ ++ } 1 \\
&\hat{x} \text{ -- } 1 \\
&d \text{ -- } 1 \\
&c \text{ -- } 1.
\end{align*}
\]
Let $R$ - fixed positive integer.
Suppose some 3 counters $b$, $c$, $d$ are initially set nondeterministically to:

\[
\begin{align*}
    b &= R \\
    c &> 0 \\
    d &= c \cdot R
\end{align*}
\]  

ratio $R$

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

The idea:

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

\[
\begin{align*}
    \text{loop} & \quad \begin{align*}
        x &=+ 1 \\
        \hat{x} &= -1 \\
        d &= -1 \\
        c &= -1
    \end{align*}
\end{align*}
\]

} at most $R$ iterations
Let $R$ - fixed positive integer.
Suppose some 3 counters $b, c, d$ are initially set \textbf{nondeterministically} to:
\[
\begin{align*}
    b &= R \\
    c &> 0 \\
    d &= c \cdot R
\end{align*}
\]

How to simulate $R$-bounded counter program \textbf{with zero tests}?

The idea:

\[
\begin{align*}
    x + \hat{x} &\leq R \text{ and } d \geq c \cdot R \\
\end{align*}
\]

\[
\begin{array}{c}
\text{loop} \hspace{1cm} \begin{align*}
    x &+= 1 \\
    \hat{x} &-= 1 \\
    d &-= 1 \\
    c &-= 1
\end{align*}
\end{array}
\]

\{ \text{at most } R \text{ iterations} \}
Let $R$ - fixed positive integer.

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

\[
\begin{align*}
  b &= R \\
  c &= > 0 \\
  d &= c \cdot R
\end{align*}
\]

ratio $R$

How to simulate $R$-bounded counter program \textbf{with zero tests}?

The idea:

\[
\begin{align*}
  x + \hat{x} &\leq R \text{ and } d \geq c \cdot R \\
\end{align*}
\]

\textbf{forward invariant}

\begin{align*}
\text{loop} &
\begin{align*}
  x &\text{ += 1} \\
  \hat{x} &\text{ -= 1} \\
  d &\text{ -= 1} \\
  c &\text{ -= 1.}
\end{align*}
\end{align*}

\textbf{at most $R$ iterations}
Let $R$ - fixed positive integer.
Suppose some 3 counters $b, c, d$ are initially set \textit{nondeterministically} to:
\[
\begin{align*}
    b &= R \\
    c &= > 0 \\
    d &= c \cdot R
\end{align*}
\]

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

\textbf{The idea:}

\[
\begin{align*}
    x + \hat{x} &\leq R \quad \text{and} \quad d \geq c \cdot R \\
    \text{loop} \\
    x &\mathbin{+}= 1 \\
    \hat{x} &\mathbin{-}= 1 \\
    d &\mathbin{-}= 1 \\
    c &\mathbin{-}= 1.
\end{align*}
\]

\textit{forward invariant}

\textbf{at most $R$ iterations}
Let $R$ - fixed positive integer.
Suppose some 3 counters $b$, $c$, $d$ are initially set nondeterministically to:

\[
\begin{align*}
    b &= R, \\
    c &> 0, \\
    d &= c \cdot R
\end{align*}
\]

How to simulate $R$-bounded counter program \textbf{with zero tests}?

The idea:

\[
\begin{align*}
    x + \hat{x} &\leq R \quad \text{and} \quad d \geq c \cdot R \\
\text{loop} &\quad x += 1, \quad \hat{x} -= 1, \\
    &\quad d -= 1, \quad c -= 1. \\
    d &= c \cdot R
\end{align*}
\]

\textbf{forward invariant}

\{ \text{at most } R \text{ iterations} \}

implied by \textbf{halt if} \ldots, d = 0.
Let $R$ - fixed positive integer.

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

\[ b = R \quad c > 0 \quad d = c \cdot R \]

ratio $R$

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

The idea:

\[ x + \hat{x} \leq R \text{ and } d \geq c \cdot R \text{ forward invariant} \]

\[ \begin{align*}
\text{loop} \\
x &\leftarrow 1 \\
\hat{x} &\leftarrow 1 \\
d &\leftarrow 1 \\
c &\leftarrow 1.
\end{align*} \]

\[ \{ \text{exactly at most } R \text{ iterations} \}

implied by \textbf{halt if ...}, $d = 0$. 

Let $R$ - fixed positive integer.
Suppose some 3 counters $b$, $c$, $d$ are initially set \textbf{nondeterministically} to:

$$b = R \quad c > 0 \quad d = c \cdot R$$

ratio $R$

How to simulate $R$-bounded counter program \textbf{with zero tests}?

The idea:

\begin{align*}
\begin{array}{c}
\text{loop} \\
\text{x += 1} \quad \hat{x} -= 1 \\
\text{d -= 1} \\
\text{c -= 1.}
\end{array}
\end{align*}

forward invariant

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

exactly at most $R$ iterations

backward invariant

$$d = c \cdot R$$

implied by halt if ..., $d = 0$. 

16
Let $R$ - fixed positive integer.
Suppose some 3 counters $b, c, d$ are initially set nondeterministically to:

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

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

The idea:

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

\[\begin{align*}
\text{loop} & \quad x \mathrel{+}= 1 \quad \hat{x} \mathrel{-}= 1 \\
& \quad d \mathrel{-}= 1 \\
& \quad c \mathrel{-}= 1.
\end{align*}\]

\[\begin{align*}
\leq \{ \text{exactly at most } R \text{ iterations} \}
\end{align*}\]

$$d = c \cdot R \quad \text{backward invariant}$$

implied by $\textbf{halt if } \ldots, d = 0.$
Using ratio $R$ to simulate $R$-bounded counter program $\mathcal{P}$ with zero tests

\[ b = R \quad c > 0 \quad d = c \cdot R \]
Using ratio $R$ to simulate $R$-bounded counter program $P$ with zero tests

$b = R \quad c > 0 \quad d = c \cdot R$

• introduce shadow counters and initiate them to at most $R$:

```
loop
  \hat{x} += 1 \quad \hat{y} += 1 \quad \ldots
  d -= 1 \quad b -= 1
  c -= 1
```

(only for zero-tested counters)
Using ratio $R$ to simulate $R$-bounded counter program $\mathcal{P}$ with zero tests

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

  $b = R \quad c > 0 \quad d = c \cdot R$

  - loop
    
    - $\hat{x} \leftarrow 1$
    - $\hat{y} \leftarrow 1$
    - $d \leftarrow 1$
    - $b \leftarrow 1$
    - $c \leftarrow 1$

    (only for zero-tested counters)

- $x \leftarrow 1$ replace by $x \leftarrow 1 \hat{x} \leftarrow 1$
Using ratio $R$ to simulate $R$-bounded counter program $P$ with zero tests

$$b = R \quad c > 0 \quad d = c \cdot R$$

- Introduce shadow counters and initiate them to at most $R$:

  ```
  loop
  \hat{x} += 1 \quad \hat{y} += 1 \quad \ldots
  d -= 1 \quad b -= 1
  c -= 1
  ```

  *(only for zero-tested counters)*

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

- $x -= 1$ replace by $x -= 1 \quad \hat{x} += 1$
Using ratio $R$ to simulate $R$-bounded counter program $\mathcal{P}$ with zero tests

\[ b = R \quad c > 0 \quad d = c \cdot R \]

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

```
loop
  \hat{x} ++ 1 \quad \hat{y} ++ 1 \quad \ldots
  d -- 1 \quad b -- 1
  c = - 1
```

(only for zero-tested counters)

- $x += 1$ replace by $x += 1 \quad \hat{x} -= 1$
- $x -= 1$ replace by $x -= 1 \quad \hat{x} += 1$
- $\text{zero?} \times$ replace by

```
loop
  x ++ 1 \quad \hat{x} -= 1
  d -- 1
  c -= 1

loop
  x -= 1 \quad \hat{x} += 1
  d -- 1
  c -= 1
```
Using ratio $R$ to simulate $R$-bounded counter program $P$ with zero tests

$b = R \quad c > 0 \quad d = c \cdot R$

- Introduce shadow counters and initiate them to at most $R$:

```
loop
  \hat{x} +\!\!\!\!\equiv 1 \quad \hat{y} +\!\!\!\!\equiv 1 \quad \ldots
  d \equiv 1
  b \equiv 1
  c \equiv 1
```

(only for zero-tested counters)

- $x +\!\!\!\!\equiv 1$ replace by $x +\!\!\!\!\equiv 1 \quad \hat{x} \equiv 1$

- $x \equiv 1$ replace by $x \equiv 1 \quad \hat{x} +\!\!\!\!\equiv 1$

- **zero?** $x$ replace by

```
loop
  x +\!\!\!\!\equiv 1 \quad \hat{x} \equiv 1
  d \equiv 1
  c \equiv 1

loop
  x \equiv 1 \quad \hat{x} +\!\!\!\!\equiv 1
  d \equiv 1
  c \equiv 1
```
Using ratio $R$ to simulate $R$-bounded counter program $P$

with zero tests

\[ b = R \quad c > 0 \quad d = c \cdot R \]

- Introduce shadow counters and initiate them to at most $R$:
  
  \[
  \begin{align*}
  \hat{x} & \equiv 1 \\
  \hat{y} & \equiv 1 \\
  d & \equiv 1 \\
  b & \equiv 1 \\
  c & \equiv 1
  \end{align*}
  \]

  (Only for zero-tested counters)

- $x \equiv 1$ replace by $x \equiv 1 \quad \hat{x} \equiv 1$

- $x \equiv 1$ replace by $x \equiv 1 \quad \hat{x} \equiv 1$

- zero? $x$ replace by

  \[
  \begin{align*}
  \hat{x} & \equiv 1 \\
  d & \equiv 1 \\
  c & \equiv 1
  \end{align*}
  \]

  Forward invariant

  \[ x + \hat{x} \leq R \text{ and } d \geq c \cdot R \]
Using ratio $R$ to simulate $R$-bounded counter program $P$ with zero tests

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

```
loop
\hat{x} += 1 \quad \hat{y} += 1 \quad \ldots
\hline
\text{d} -= 1 \quad \text{b} -= 1
\hline
\text{c} -= 1
```

(only for zero-tested counters)

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

```
loop
\hline
x += 1 \quad \hat{x} -= 1
\hline
d -= 1
c -= 1
```

forward invariant

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

- extend halt: \textbf{halt if } \ldots, d = 0.
Using ratio $R$ to simulate $R$-bounded counter program $P$ with zero tests

\[ b = R \quad c > 0 \quad d = c \cdot R \]

- Introduce shadow counters and initiate them to at most $R$:
  
  \[
  \begin{align*}
  \text{loop} \quad \hat{x} &\leftarrow 1 \quad \hat{y} \leftarrow 1 \quad \ldots \\
  d &\leftarrow 1 \quad b \leftarrow 1 \\
  c &\leftarrow 1
  \end{align*}
  \]

  (only for zero-tested counters)

- $x \leftarrow 1$ replace by $x \leftarrow 1 \quad \hat{x} \leftarrow 1$

- $x \leftarrow 1$ replace by $x \leftarrow 1 \quad \hat{x} \leftarrow 1$

- zero? $x$ replace by

  \[
  \begin{align*}
  \text{loop} \\
  x &\leftarrow 1 \quad \hat{x} \leftarrow 1 \\
  d &\leftarrow 1 \\
  c &\leftarrow 1 \\
  \text{loop} \\
  x &\leftarrow 1 \quad \hat{x} \leftarrow 1 \\
  d &\leftarrow 1 \\
  c &\leftarrow 1
  \end{align*}
  \]

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

- Extend halt: **halt if** $\ldots, d = 0$.

  backward invariant $d = c \cdot R$
Using ratio $R$ to simulate $R$-bounded counter program $P$

with zero tests

$b = R \quad c > 0 \quad d = c \cdot R$

• introduce shadow counters and initiate them to at most $R$:

\[
\begin{align*}
\text{loop} & \quad \hat{x} += 1 & \quad \hat{y} += 1 & \quad \ldots \\
& \quad d -- 1 & \quad b -- 1 \\
& \quad c -- 1
\end{align*}
\]  

\{ exactly $R$ iterations \}

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

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

• zero? $x$ replace by

\[
\begin{align*}
\text{loop} & \quad x += 1 & \quad \hat{x} -- 1 \\
& \quad d -- 1 \\
& \quad c -- 1
\end{align*}
\]

\{ exactly $R$ iterations \}

\[
\begin{align*}
\text{loop} & \quad x -= 1 & \quad \hat{x} += 1 \\
& \quad d -- 1 \\
& \quad c -- 1
\end{align*}
\]

\{ exactly $R$ iterations \}

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

• extend halt: \textbf{halt if} \ldots, $d = 0$.

backward invariant \quad $d = c \cdot R$
Using ratio $R$ to simulate $R$-bounded counter program $\mathcal{P}$ with zero tests

$b = R \quad c > 0 \quad d = c \cdot R$

- Introduce shadow counters and initiate them to at most $R$:

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

  \begin{align*}
  \text{exactly } R \text{ iterations}
  \end{align*}

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

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

- $\text{zero? } x$ replace by

  - $\text{loop}$
    - $x += 1 \quad \hat{x} -- 1$
    - $d -- 1$
    - $c -- 1$

  - $\text{loop}$
    - $x -= 1 \quad \hat{x} += 1$
    - $d -- 1$
    - $c -- 1$

  \begin{align*}
  \text{exactly } R \text{ iterations}
  \end{align*}

  \begin{align*}
  \text{exactly } R \text{ iterations}
  \end{align*}

- Violation punished at the end

  - Forward invariant
    - $x + \hat{x} \leq R$ and $d \geq c \cdot R$

  - Backward invariant
    - $d = c \cdot R$

- Extend halt: $\text{halt if } \ldots, d = 0.$
Using ratio $R$ to simulate $R$-bounded counter program $P$
with zero tests

\[ b = R \quad c > 0 \quad d = c \cdot R \]

- introduce shadow counters and initiate them to at most $R$:
  
  \[
  \begin{align*}
  &\text{loop} \\
  &\hat{x} ++ 1 \quad \hat{y} ++ 1 \quad \cdots \\
  &d -- 1 \quad b -- 1 \\
  &c -- 1
  \end{align*}
  \]

  \{ \text{exactly } R \text{ iterations} \}

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

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

- \textbf{zero? } $x$ replace by
  
  \[
  \begin{align*}
  &\text{loop} \\
  &x ++ 1 \quad \hat{x} -- 1 \\
  &d -- 1 \\
  &c -- 1 \\
  &\text{loop} \\
  &x -- 1 \quad \hat{x} += 1 \\
  &d -- 1 \\
  &c -- 1
  \end{align*}
  \]

  \{ \text{exactly } R \text{ iterations} \}

  \{ \text{exactly } R \text{ iterations} \}

- \textbf{extend halt: } \textbf{halt if } \ldots, d = 0.

  \{ \text{violation punished at the end} \}

  \{ \text{backward invariant } \} \quad d = c \cdot R
Using ratio $R$ to simulate $R$-bounded counter program $\mathcal{P}$

with zero tests

$b = R$  $c > 0$  $d = c \cdot R$

• introduce shadow counters and initiate them to at most $R$:

\[
\begin{align*}
\text{loop} \\
\hat{x} &+= 1 & \hat{y} &+= 1 & \ldots \\
d &-= 1 & b &-= 1 \\
c &-= 1
\end{align*}
\]

\{ exactly $R$ iterations \}

• $x += 1$ replace by $x += 1$  \(\hat{x} -= 1\)

• $x -= 1$ replace by $x -= 1$

• \text{zero?} $x$ replace by

\[
\begin{align*}
\text{loop} \\
x &+= 1 & \hat{x} &-= 1 \\
d &-= 1 \\
c &-= 1 \\
\text{loop} \\
x &-= 1 & \hat{x} &+= 1 \\
d &-= 1 \\
c &-= 1
\end{align*}
\]

\{ exactly $R$ iterations \}

\{ exactly $R$ iterations \}

the construction doesn’t depend on $R$

\begin{align*}
\text{forward invariant} & \\
x + \hat{x} \leq R & \text{ and } d \geq c \cdot R
\end{align*}

violation punished at the end

\begin{align*}
\text{backward invariant} & \\
d & = c \cdot R
\end{align*}

• extend halt: \textbf{halt if} \ldots, $d = 0$. 

17
Computing and using ratio

\[ A > P \]
Computing and using ratio

$A \triangleright P$

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

counter program \textbf{without zero tests} that computes ratio $R$

$A \implies P$

$R$-bounded counter program \textbf{with zero tests} which is simulated using ratio $R$
Computing and using ratio

counter program without zero tests

A $\naboovearrow$ 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

counter program without zero tests

counter program without zero tests
that computes ratio \( R \)

\( \mathcal{A} \uparrow \mathcal{P} \)

R-bounded counter program with zero tests
which is simulated using ratio \( R \)

merged halts of \( \mathcal{A} \) and \( \mathcal{P} \)

• extend halt: \texttt{halt if \ldots, d = 0.}
How to compute ratio?

$$3!^n = \underbrace{(3!)! \ldots !}_{n \text{ times}}$$
How to compute ratio?

• ratio 3:

1: \( b \leftarrow 3 \)
2: \( c \leftarrow 1 \quad d \leftarrow 3 \)
3: \textbf{loop}
4: \( c \leftarrow 1 \quad d \leftarrow 3 \)
5: \textbf{halt}.

\[
3!^n = \underbrace{(3!)(3!) \ldots}_{\text{n times}}
\]
How to compute ratio?

- ratio 3:

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

- we define a counter program that, using ratio $R$, computes ratio $R!$
How to compute ratio?

• ratio 3:

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

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

$$3!^n = ((3!)! \ldots)!$$

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

factorial amplifier
How to compute ratio?

• ratio 3:

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

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

• and self-compose it sufficiently many times: $((A_3 \triangleright F) \triangleright F) \triangleright \cdots \triangleright F$

$3^n = ((3!)! \ldots)!$

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

factorial amplifier

$n$ compositions
Factorial amplifier - the idea
counter program that, using ratio $R$, computes ratio $R!$
Factorial amplifier - the idea

\[
\frac{2}{1} \cdot \frac{3}{2} \cdot \ldots \cdot \frac{R}{R-1} = R
\]
Factorial amplifier - the idea

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

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

1: $i \ += \ 1$  $x \ += \ 1$  $y \ += \ 1$
2: loop
3: $x \ += \ 1$  $y \ += \ 1$
4: loop
5: loop
6: $x \ -= \ i$  $x' \ += \ i + 1$
7: loop
8: $x' \ -= \ 1$  $x \ += \ 1$
9: $i \ += \ 1$
10: zero? $i$
11: loop
12: $x \ -= \ i$  $y \ -= \ 1$
13: halt if $y = 0$

a zero test
Factorial amplifier - the idea

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

counter program that, using ratio \( R \), computes ratio \( R! \)

```
1: i += 1  x += 1  y += 1
2: loop
3:   x += 1  y += 1
4:   loop
5:     loop
6:       x -= i  x' += i + 1
7:     loop
8:       x' -= 1  x += 1
9:         i += 1
10: zero? i
11: loop
12:   x -= i  y -= 1
13: halt if y = 0
```

initially equal \( R \)
a zero test
Factorial amplifier - the idea

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

counter program that, using ratio \( R \), computes ratio \( R! \)

Initially equal \( R \)

A zero test

Loop

\[
\begin{align*}
1: & \quad i := 1 \quad x := 1 \quad y := 1 \\
2: & \quad \text{loop} \\
3: & \quad x := 1 \quad y := 1 \\
4: & \quad \text{loop} \\
5: & \quad \text{loop} \\
6: & \quad x := i \quad x' := i + 1 \\
7: & \quad \text{loop} \\
8: & \quad x' := 1 \quad x := 1 \\
9: & \quad i := 1 \\
10: & \quad \text{zero? } i \\
11: & \quad \text{loop} \\
12: & \quad x := i \quad y := 1 \\
13: & \quad \text{halt if } y = 0
\end{align*}
\]

Further zero tests
Factorial amplifier - the idea

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

Counter program that, using ratio \( R \), computes ratio \( R! \)

1: i += 1  x += 1  y += 1
2: loop
3: x += 1  y += 1
4: loop
5: loop
6: x -= i  x' += i + 1
7: loop
8: x' -= 1  x += 1
9: i += 1
10: zero? \( i \)
11: loop
12: x -= i  y -= 1
13: 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

\[
\begin{array}{cccc}
\frac{2}{1} & \cdot & \frac{3}{2} & \cdot \ldots \cdot \frac{R}{R-1} = R
\end{array}
\]

counter program that, using ratio \( R \), computes ratio \( R! \)

1: \( i \) += 1 \( x \) += 1 \( y \) += 1
2: loop **nondeterministic init**
3: \( x \) += 1 \( y \) += 1
4: loop
5: loop
6: \( x \) -= i \( x' \) += i + 1
7: loop
8: \( x' \) -= 1 \( x \) += 1
9: \( i \) += 1
10: zero? \( i \)
11: loop
12: \( x \) -= i \( y \) -= 1
13: **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

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

Counter program that, using ratio \( R \), computes ratio \( R! \)

```plaintext
1: i += 1  x += 1  y += 1
2: loop
3:   x += 1  y += 1
4: loop
5:   loop
6:     x -= i  x' += i + 1
7:   loop
8:     x' -= 1  x += 1
9: i += 1
10: zero? i
11: loop
12:   x -= i  y -= 1
13: halt if y = 0
```

Initially equal \( R \)

A zero test

Weak multiplication by \( \frac{i+1}{i} \)

Further zero tests
Factorial amplifier - the idea

\[
\begin{array}{c}
\frac{2}{1} \cdot \frac{3}{2} \cdot \cdots \cdot \frac{R}{R-1} = R
\end{array}
\]

counter program that, using ratio \( R \), computes ratio \( R! \)

Initially equal \( R \)

A zero test

Tests if \( x \geq y \cdot R \)

Weak multiplication by \( \frac{i+1}{i} \)

Further zero tests

Non-deterministic init
Factorial amplifier - the idea

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

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

1: $i \gets 1$  \hspace{1em} $x \gets 1$  \hspace{1em} $y \gets 1$

2: **loop** \hspace{1em} **nondeterministic init**
   \hspace{1em} $x \gets 1$  \hspace{1em} $y \gets 1$

3: **loop**
   \hspace{1em} $x \gets i$  \hspace{1em} $x' \gets i + 1$

4: **loop**
   \hspace{1em} $x' \gets 1$  \hspace{1em} $x \gets 1$

5: \hspace{1em} $i \gets 1$

6: **zero? $i$**

7: **loop**
   \hspace{1em} $x \gets i$  \hspace{1em} $y \gets 1$

8: **halt if $y = 0$**

9: tests if $x \geq y \cdot R$

10: \hspace{1em} $x' \gets 1$

11: **loop**
   \hspace{1em} $i \gets 1$  \hspace{1em} $i' \gets 1$  \hspace{1em} $x' \gets 1$

12: **zero? $i$**

13: **loop**
   \hspace{1em} $i' \gets 1$  \hspace{1em} $i \gets 1$

14: **zero? $i'$**

initially equal $R$

a zero test

**exact**

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

**further zero tests**

counter program that, using ratio $R$, computes ratio $R!$
Factorial amplifier - the idea

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

initially equal \( R \)

a zero test

tests if \( x \geq y \cdot R \)

counter program that computes ratio \( R \)!

fine, but where is the factorial?

loop: \( i' \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i \)

loop: \( i' \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i' \)

loop: \( i \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i \)

loop: \( i' \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i' \)

loop: \( i \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i \)

loop: \( i' \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i' \)

loop: \( i \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i \)

loop: \( i' \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i' \)

loop: \( i \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i \)

loop: \( i' \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i' \)

loop: \( i \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i \)

loop: \( i' \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i' \)

loop: \( i \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i \)

loop: \( i' \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i' \)

loop: \( i \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i \)

loop: \( i' \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i' \)

loop: \( i \quad i' \quad x' \quad x \quad y \quad y \quad \)

zero? \( i \)
Factorial amplifier - the idea

\[
\begin{array}{cccccccccc}
\frac{2}{1} & \cdot & \frac{3}{2} & \cdot & \ldots & \cdot & \frac{R}{R-1} \\
= & R
\end{array}
\]

counter program that computes ratio \( R \) but where is the factorial?

20

initially equal \( R \)

a zero test

tests if \( x \geq y \cdot R \)

nondeterministic init

loop

tests if

loop

exact

weak multiplication by \( \frac{i+1}{i} \)

loop

loop

zero? \( i \)

zero? \( i' \)

zero? \( i \)

zero? \( i' \)
Factorial amplifier \( b = R! \quad c > 0 \quad d = c \cdot R! \)
Factorial amplifier  \( b = R! \)  \( c > 0 \)  \( d = c \cdot R! \)

counter program that, using ratio \( R \), computes ratio \( R! \)

1:  \( i += 1 \)  \( x += 1 \)  \( y += 1 \)  \( b += 1 \)  \( c += 1 \)  \( d += 1 \)
2:  \textbf{loop}
3:     \( x += 1 \)  \( y += 1 \)  \( c += 1 \)  \( d += 1 \)
4:  \textbf{loop}
5:     \textbf{loop}
6:         \( c -= i \)  \( c' += 1 \)
7:     \textbf{loop at most} \( b \) \textbf{times}
8:         \( x -= i \)  \( d -= i \)  \( x' += i + 1 \)
9:     \textbf{loop}
10:         \( b -= 1 \)  \( b' += i + 1 \)
11: \textbf{loop}
12:     \( b' -= 1 \)  \( b += 1 \)
13: \textbf{loop}
14:     \( c' -= 1 \)  \( c += 1 \)
15: \textbf{loop at most} \( b \) \textbf{times}
16:         \( x' -= 1 \)  \( x += 1 \)  \( d += 1 \)
17:  \( i += 1 \)
18:  \textbf{zero?}  \( \hat i \)
19: \textbf{loop}
20:     \( x -= i \)  \( y -= 1 \)
21: \textbf{halt if} \( y = 0 \)
Factorial amplifier \( b = R! \quad c > 0 \quad d = c \cdot R! \)

Counter program that, using ratio \( R \), computes ratio \( R! \)

```
1: i += 1  x += 1  y += 1  b += 1  c += 1  d += 1
2: loop
3:   x += 1  y += 1  c += 1  d += 1
4: loop
5:   loop
6:     c -= i  c' += 1
7:     loop at most b times
8:       x' -= i  d -= i  x' += i + 1
9:   loop
10:  b -= 1  b' += i + 1
11: loop
12:  b' -= 1  b += 1
13: loop
14:  c' -= 1  c += 1
15: loop at most b times
16:     x' -= 1  x += 1  d += 1
17: i += 1
18: zero? i
19: loop
20:   x -= i  y -= 1
21: halt if y = 0
```
Factorial amplifier $b = R!$, $c > 0$, $d = c \cdot R!$

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

counter program that, using ratio $R$, computes ratio $R!$
$b = R! \quad c > 0 \quad d = c \cdot R!$

Counter program that, using ratio $R$, computes ratio $R!$.

Factorial amplifier

1: $i \leftarrow i + 1 \quad x \leftarrow x + 1 \quad y \leftarrow y + 1 \quad b \leftarrow b + 1 \quad c \leftarrow c + 1 \quad d \leftarrow d + 1$

2: **loop**

3: $x \leftarrow x + 1 \quad y \leftarrow y + 1 \quad c \leftarrow c + 1 \quad d \leftarrow d + 1$

4: **loop**

5: **loop**

6: $c \leftarrow c - i \quad c' \leftarrow c' + 1$

7: **loop at most $b$ times**

8: $x \leftarrow x - i \quad d \leftarrow d - i \quad x' \leftarrow x' + i + 1$

9: **loop**

10: $b \leftarrow b - 1 \quad b' \leftarrow b' + i + 1$

11: **loop**

12: $b' \leftarrow b' - 1 \quad b \leftarrow b + 1$

13: **loop**

14: $c' \leftarrow c' - 1 \quad c \leftarrow c + 1$

15: **loop at most $b$ times**

16: $x' \leftarrow x' - 1 \quad x \leftarrow x + 1 \quad d \leftarrow d + 1$

17: $i \leftarrow i + 1$

18: zero? $\hat{i}$

19: **loop**

20: $x \leftarrow i \quad y \leftarrow 1$

21: halt if $y = 0$
Factorial amplifier \( b = R! \) \( c > 0 \) \( d = c \cdot R! \)

counter program that, using ratio \( R \), computes ratio \( R! \)

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

nondeterministic init

\( \text{invariant } d = c \cdot b \)

weak multiplication by \( \frac{i + 1}{i} \)
counter program that, using ratio $R$, computes ratio $R!$

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

1: $i += 1$ $x += 1$ $y += 1$ $b += 1$ $c += 1$ $d += 1$

2: \text{loop} \hspace{1cm} \text{nondeterministic init}

3: $x += 1$ $y += 1$ $c += 1$ $d += 1$

4: \text{loop}

5: \hspace{1cm} \text{loop}

6: $c -= i$ $c' += 1$

7: \hspace{1cm} \text{loop at most } b \text{ times}

8: $x -= i$ $d -= i$ $x' += i + 1$

9: \hspace{1cm} \text{loop}

10: $b -= 1$ $b' += i + 1$

11: \hspace{1cm} \text{loop}

12: $b' -= 1$ $b += 1$

13: \hspace{1cm} \text{loop}

14: $c' -= 1$ $c += 1$

15: \hspace{1cm} \text{loop at most } b \text{ times}

16: $x' -= 1$ $x += 1$ $d += 1$

17: $i += 1$

18: \text{zero? } \hat{i}$

19: \text{loop}

20: $x -= i$ $y -= 1$

21: \text{halt if } y = 0

tests if $x \geq y \cdot R$

\text{invariant} \hspace{1cm} d = c \cdot b

\text{exact} \hspace{1cm} \text{weak multiplication by } \frac{i+1}{i}$
Factorial amplifier: \( b = R! \), \( c > 0 \), \( d = c \cdot R! \)

counter program that, using ratio \( R \), computes ratio \( R! \)

```
1: i += 1  x += 1  y += 1  b += 1  c += 1  d += 1
2: loop  nondeterministic init
3:   x += 1  y += 1  c += 1  d += 1
4: loop
5:   loop  weak division by i
6:     c -= i  c' += 1
7:     loop at most b times
8:       x -= i  d -= i  x' += i + 1
9:     loop
10:    b -= 1  b' += i + 1
11: loop
12:   b' -= 1  b += 1
13: loop
14:   c' -= 1  c += 1
15:     loop at most b times
16:       x' -= 1  x += 1  d += 1
17: i += 1
18: zero? i
19: loop
20:   x -= i  y -= 1
21: halt if y = 0
```

Tests if \( x \geq y \cdot R \)
Factorial amplifier: \[ b = R! \quad c > 0 \quad d = c \cdot R! \]

```plaintext
1: i += 1 \quad x += 1 \quad y += 1 \quad b += 1 \quad c += 1 \quad d += 1

2: \textbf{loop}
   \quad \text{non-deterministic init}
   \quad x += 1 \quad y += 1 \quad c += 1 \quad d += 1

3: \textbf{loop}
   \quad \textbf{exact}
   \quad \text{weak division by } i
   \quad c -= i \quad c' += 1
   \quad \text{loop at most } b \text{ times}
   \quad x -= i \quad d -= i \quad x' += i + 1

4: \textbf{loop}
   \quad b -= 1 \quad b' += i + 1

5: \textbf{loop}
   \quad b' -= 1 \quad b += 1

6: \textbf{loop}
   \quad c' -= 1 \quad c += 1
   \quad \text{loop at most } b \text{ times}
   \quad x' -= 1 \quad x += 1 \quad d += 1

7: i += 1

8: \textbf{zero? } \hat{i}

9: \textbf{loop}
   \quad x -= i \quad y -= 1

10: \textbf{halt if } y = 0

\text{tests if } x \geq y \cdot R
```

counter program that, using ratio \( R \), computes ratio \( R! \)

\[ \text{invariant } d = c \cdot b \]
Factorial amplifier \( b = R! \) \( c > 0 \) \( d = c \cdot R! \)

counter program that, using ratio \( R \), computes ratio \( R! \)

\[
\begin{align*}
1: & \quad i \mathrel{+} = 1 \quad x \mathrel{+} = 1 \quad y \mathrel{+} = 1 \quad b \mathrel{+} = 1 \quad c \mathrel{+} = 1 \quad d \mathrel{+} = 1 \\
2: & \quad \text{loop} \quad \text{nondeterministic init} \\
3: & \quad x \mathrel{+} = 1 \quad y \mathrel{+} = 1 \quad c \mathrel{+} = 1 \quad d \mathrel{+} = 1 \\
4: & \quad \text{loop} \quad \text{exact} \\
5: & \quad \quad \text{loop} \quad \text{weak division by } i \\
6: & \quad \quad \quad c \mathrel{-} = i \quad c' \mathrel{+} = 1 \\
7: & \quad \quad \text{loop at most } b \text{ times} \\
8: & \quad \quad \quad x \mathrel{-} = i \quad d \mathrel{-} = i \quad x' \mathrel{+} = i + 1 \\
9: & \quad \text{loop} \\
10: & \quad b \mathrel{-} = 1 \quad b' \mathrel{+} = i + 1 \\
11: & \quad \text{loop} \\
12: & \quad b' \mathrel{-} = 1 \quad b \mathrel{+} = 1 \\
13: & \quad \text{loop} \\
14: & \quad c' \mathrel{-} = 1 \quad c \mathrel{+} = 1 \\
15: & \quad \text{loop at most } b \text{ times} \\
16: & \quad x' \mathrel{-} = 1 \quad x \mathrel{+} = 1 \quad d \mathrel{+} = 1 \\
17: & \quad i \mathrel{+} = 1 \\
18: & \quad \text{zero? } \hat{i} \\
19: & \quad \text{loop} \\
20: & \quad x \mathrel{-} = i \quad y \mathrel{-} = 1 \\
21: & \quad \text{halt if } y = 0 \\
\end{align*}
\]

tests if \( x \geq y \cdot R \)
counter program that, using ratio $R$, computes ratio $R!$

**Factorial amplifier** 
\[ b = R! \quad c > 0 \quad d = c \cdot R! \]

1: \( i += 1 \quad x += 1 \quad y += 1 \quad b += 1 \quad c += 1 \quad d += 1 \)

2: \textbf{loop} \hspace{1cm} \text{non-deterministic init}
3: \( x += 1 \quad y += 1 \quad c += 1 \quad d += 1 \)

4: \textbf{loop} \hspace{1cm} \text{exact}
5: \textbf{loop} \hspace{1cm} \text{weak division by } i
6: \( c += i \quad c' += 1 \)
7: \textbf{loop at most } b \textbf{ times}
8: \( x -= i \quad d -= i \quad x' += i + 1 \)
9: \textbf{loop}
10: \( b -= 1 \quad b' += i + 1 \)
11: \textbf{loop}
12: \( b' -= 1 \quad b += 1 \)
13: \textbf{loop}
14: \( c' -= 1 \quad c += 1 \)
15: \textbf{loop at most } b \textbf{ times}
16: \( x' -= 1 \quad x += 1 \quad d += 1 \)
17: \( i += 1 \)
18: \textbf{zero? } i
19: \textbf{loop}
20: \( x -= i \quad y -= 1 \)
21: \textbf{halt if } y = 0

**Invariant** 
\[ d = c \cdot b \]

\[ \text{exact} \quad \text{weak multiplication by } \frac{i+1}{i} \]

\[ \text{exact} \quad \text{weak multiplication by } i+1 \]
counter program that, using ratio $R$, computes ratio $R!$

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

1. $i += 1$  $x += 1$  $y += 1$  $b += 1$  $c += 1$  $d += 1$

2. **loop**
   - **nondeterministic init**
   - $x += 1$  $y += 1$  $c += 1$  $d += 1$

3. **loop**
   - **exact**
     - weak division by $i$
   - $c -= i$  $c' += 1$

4. **loop at most $b$ times**
   - $x -= i$  $d -= i$  $x' += i + 1$

5. **loop**
   - **exact**
     - weak multiplication by $\frac{i+1}{i}$
   - $b -= 1$  $b' += i + 1$

6. **loop**
   - $b' -= 1$  $b += 1$

7. **loop**
   - **exact**
     - weak multiplication by $i+1$
   - $c' -= 1$  $c += 1$

8. **loop at most $b$ times**
   - $x' -= 1$  $x += 1$  $d += 1$

9. $i += 1$

10. **zero? $\hat{i}$**

11. **loop**
    - $x -= i$  $y -= 1$

12. **halt if $y = 0$**

13. tests if $x \geq y \cdot R$
Future work
Future work

• TOWER…ACKERMANN gap

\( F_\omega \) gap
Future work

• TOWER…ACKERMANN gap

• improving the lower bound? TOWER amplifier?

F₀…Fₚ gap
Future work

• TOWER…ACKERMANN 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

\[ F_3 \ldots F_\omega \text{ gap} \]
Future work

• TOWER…ACKERMANN gap
  • 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
Future work

• TOWER…ACKERMANN 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

• 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

F₃…Fω gap

thank you!