Algorithmic Aspects of WQO (Well-Quasi-Ordering) Theory

Part IV: Fast-growing complexity 2

Philippe Schnoebelen

LSV, CNRS & ENS Cachan

Chennai Mathematical Institute, Jan. 2017

Based on joint work with Sylvain Schmitz, Prateek Karandikar, K. Narayan Kumar, Alain Finkel, ..

Lecture notes & exercises available via `www.lsv.ens-cachan.fr/~phs`

If you missed the earlier episodes

$(\mathbb{N}^k, \leqslant_\times)$ and $(\Sigma^*, \leqslant_*)$ are well-quasi-orderings: any infinite sequence $\mathbf{x} = x_0, x_1, x_2, \ldots$ contains an increasing pair $x_i \leqslant x_j$ —we say it is good—

If a sequence like $\mathbf{x}$ cannot grow too quickly —we say it is controlled— then the position i, j of the first increasing pair in $\mathbf{x}$ can be bounded by some length function $L_{X,\text{control}}(|x_0|)$

This gave us upper bounds for the complexity of wqo-based algorithms. Furthermore, these length functions can be precisely pinned down inside elegant subrecursive hierarchies

For example, it gave $\mathbf{F}_\omega$ upper-bounds for the verification —e.g., termination and/or safety— of monotonic counter machines, and $\mathbf{F}_{\omega^\omega}$ upper bounds for lossy channel systems

IF YOU MISSED THE EARLIER EPISODES

$(\mathbb{N}^k, \leqslant_\times)$ and $(\Sigma^*, \leqslant_*)$ are well-quasi-orderings: any infinite sequence $\mathbf{x} = x_0, x_1, x_2, \ldots$ contains an increasing pair $x_i \leqslant x_j$ —we say it is good—

If a sequence like $\mathbf{x}$ cannot grow too quickly —we say it is controlled— then the position i, j of the first increasing pair in $\mathbf{x}$ can be bounded by some length function $L_{X,\text{control}}(|x_0|)$

This gave us upper bounds for the complexity of wqo-based algorithms. Furthermore, these length functions can be precisely pinned down inside elegant subrecursive hierarchies

For example, it gave $\mathbf{F}_\omega$ upper-bounds for the verification —e.g., termination and/or safety— of monotonic counter machines, and $\mathbf{F}_{\omega^\omega}$ upper bounds for lossy channel systems

That was just the EASY part!!!

IF YOU MISSED THE EARLIER EPISODES

$(\mathbb{N}^k, \leqslant_\times)$ and $(\Sigma^*, \leqslant_*)$ are well-quasi-orderings: any infinite sequence $\mathbf{x} = x_0, x_1, x_2, \ldots$ contains an increasing pair $x_i \leqslant x_j$ —we say it is good—

If a sequence like $\mathbf{x}$ cannot grow too quickly —we say it is controlled— then the position i, j of the first increasing pair in $\mathbf{x}$ can be bounded by some length function $L_{X,\text{control}}(|x_0|)$

This gave us upper bounds for the complexity of wqo-based algorithms. Furthermore, these length functions can be precisely pinned down inside elegant subrecursive hierarchies

For example, it gave $\mathbf{F}_\omega$ upper-bounds for the verification —e.g., termination and/or safety— of monotonic counter machines, and $\mathbf{F}_{\omega^\omega}$ upper bounds for lossy channel systems

Today we consider the “hardness” question: are these upper bounds optimal?, or equivalently: do we have matching lowing bounds?

—the answer is often “positive”

OUTLINE FOR TODAY

- What is the question exactly? And why isn't it obvious?
- A strategy for proving hardness
- Hardness for Lossy Counter Machines
- Hardness for Lossy Channel Systems

Problem Statement

We have upper bounds on the complexity of verification for lossy counter machines and lossy channel systems

Do we have matching lower bounds?

"Could be" for the simple-minded algorithms we presented in Part II

"**No**" for the underlying decision problems (witness: VASS's)

Exercise. Give a decision problem solvable in Ackermannian time of its input that requires Ackermannian time (where $Ack(n) \stackrel{\text{def}}{=} A(n,n)$ and A is the usual binary Ackermann function).

Pb 1. Input: x,y,z. Question: Does $A(x,y)=z$?

Pb 2. Input: x,y,x',y'. Question: Is $A(x,y)<A(x',y')$?

Pb 3. Input: x,y. Question: Is $A(x,y)$ prime?

Pb 4. Input: x,y. Question: Is $A(x,y)$ a sum $\sum_{i\in K} p_i^{F_i}$? where p_i and F_i are the ith prime (resp., Fibonacci) number

Pb 5. Input: x. Question: Does the Universal Turing machine halts on x in at most $Ack(|x|)$ steps?

PROBLEM STATEMENT

We have upper bounds on the complexity of verification for lossy counter machines and lossy channel systems

Do we have matching lower bounds?

"Could be" for the simple-minded algorithms we presented in Part II

"**No**" for the underlying decision problems (witness: VASS's)

Exercise. Give a decision problem solvable in Ackermannian time of its input that requires Ackermannian time (where $Ack(n) \stackrel{\text{def}}{=} A(n,n)$ and A is the usual binary Ackermann function).

Pb 1. Input: x, y, z. Question: Does $A(x,y) = z$?

Pb 2. Input: x, y, x', y'. Question: Is $A(x,y) < A(x',y')$?

Pb 3. Input: x, y. Question: Is $A(x,y)$ prime?

Pb 4. Input: x, y. Question: Is $A(x,y)$ a sum $\sum_{i \in K} p_i^{F_i}$? where p_i and F_i are the ith prime (resp., Fibonacci) number

Pb 5. Input: x. Question: Does the Universal Turing machine halts on x in at most $Ack(|x|)$ steps?

Problem Statement

We have upper bounds on the complexity of verification for lossy counter machines and lossy channel systems

Do we have matching lower bounds?

“Could be” for the simple-minded algorithms we presented in Part II

“**No**” for the underlying decision problems (witness: VASS’s)

Exercise. Give a decision problem solvable in Ackermannian time of its input that requires Ackermannian time (where $Ack(n) \stackrel{\text{def}}{=} A(n,n)$ and A is the usual binary Ackermann function).

Pb 1. Input: x,y,z. Question: Does $A(x,y) = z$?

Pb 2. Input: x,y,x',y'. Question: Is $A(x,y) < A(x',y')$?

Pb 3. Input: x,y. Question: Is $A(x,y)$ prime?

Pb 4. Input: x,y. Question: Is $A(x,y)$ a sum $\sum_{i \in K} p_i^{F_i}$? where p_i and F_i are the ith prime (resp., Fibonacci) number

Pb 5. Input: x. Question: Does the Universal Turing machine halts on x in at most $Ack(|x|)$ steps?

Problem Statement

We have upper bounds on the complexity of verification for lossy counter machines and lossy channel systems

Do we have matching lower bounds?

"Could be" for the simple-minded algorithms we presented in Part II

"**No**" for the underlying decision problems (witness: VASS's)

Exercise. Give a decision problem solvable in Ackermannian time of its input that requires Ackermannian time (where $Ack(n) \stackrel{\text{def}}{=} A(n,n)$ and A is the usual binary Ackermann function).

Pb 1. Input: x, y, z. Question: Does $A(x,y) = z$?

Pb 2. Input: x, y, x', y'. Question: Is $A(x,y) < A(x',y')$?

Pb 3. Input: x, y. Question: Is $A(x,y)$ prime?

Pb 4. Input: x, y. Question: Is $A(x,y)$ a sum $\sum_{i \in K} p_i^{F_i}$? where p_i and F_i are the ith prime (resp., Fibonacci) number

Pb 5. Input: x. Question: Does the Universal Turing machine halts on x in at most $Ack(|x|)$ steps?

Problem Statement

We have upper bounds on the complexity of verification for lossy counter machines and lossy channel systems

Do we have matching lower bounds?

"Could be" for the simple-minded algorithms we presented in Part II

"**No**" for the underlying decision problems (witness: VASS's)

Exercise. Give a decision problem solvable in Ackermannian time of its input that requires Ackermannian time (where $Ack(n) \stackrel{\text{def}}{=} A(n,n)$ and A is the usual binary Ackermann function).

Pb 1. Input: x,y,z. Question: Does $A(x,y) = z$?

Pb 2. Input: x,y,x',y'. Question: Is $A(x,y) < A(x',y')$?

Pb 3. Input: x,y. Question: Is $A(x,y)$ prime?

Pb 4. Input: x,y. Question: Is $A(x,y)$ a sum $\sum_{i \in K} p_i^{F_i}$? where p_i and F_i are the ith prime (resp., Fibonacci) number

Pb 5. Input: x. Question: Does the Universal Turing machine halts on x in at most $Ack(|x|)$ steps?

PROVING LOWER BOUNDS FOR MONOTONIC MODELS

We shall adopt the following strategy:

1. Compute unreliably a function in the Fast-Growing hierarchy
2. Use the result as an unreliable computational ressource
3. “Check” in the end that everything was done reliably
4. NB: Need computing unreliably the inverses of Fast-Growing functions

Great technical improvement: use Hardy hierarchy!

Fast-Growing vs. Hardy Hierarchy

$$F_0(n) \stackrel{\text{def}}{=} n+1 \qquad H^0(n) \stackrel{\text{def}}{=} n$$

$$F_{\alpha+1}(n) \stackrel{\text{def}}{=} F_\alpha^{n+1}(n) = \overbrace{F_\alpha(F_\alpha(\ldots F_\alpha}^{n+1 \text{ times}}(n)\ldots)) \qquad H^{\alpha+1}(n) \stackrel{\text{def}}{=} H^\alpha(n+1)$$

$$F_\lambda(n) \stackrel{\text{def}}{=} F_{\lambda_n}(n) \qquad H^\lambda(n) \stackrel{\text{def}}{=} H^{\lambda_n}(n)$$

with

$$(\gamma+\omega^{\beta+1})_n \stackrel{\text{def}}{=} \gamma+\omega^\beta\cdot(n+1) \qquad (\gamma+\omega^\lambda)_n \stackrel{\text{def}}{=} \gamma+\omega^{\lambda_n}$$

Prop. $H^{\alpha+\beta}(n) = H^\alpha(H^\beta(n))$ for all $\alpha+\beta$ and n

Prop. $F_\alpha(n) = H^{\omega^\alpha}(n)$ for all α and n

Prop. $H^\alpha(n) \leqslant H^{\alpha'}(n')$ and $F_\alpha(n) \leqslant F_{\alpha'}(n')$ when $\alpha \sqsubseteq \alpha'$ & $n \leqslant n'$

Fast-Growing vs. Hardy Hierarchy

$$F_0(n) \stackrel{\text{def}}{=} n+1 \qquad\qquad H^0(n) \stackrel{\text{def}}{=} n$$

$$F_{\alpha+1}(n) \stackrel{\text{def}}{=} F_\alpha^{n+1}(n) = \overbrace{F_\alpha(F_\alpha(\ldots F_\alpha}^{n+1 \text{ times}}(n)\ldots)) \qquad\qquad H^{\alpha+1}(n) \stackrel{\text{def}}{=} H^\alpha(n+1)$$

$$F_\lambda(n) \stackrel{\text{def}}{=} F_{\lambda_n}(n) \qquad\qquad H^\lambda(n) \stackrel{\text{def}}{=} H^{\lambda_n}(n)$$

with

$$(\gamma+\omega^{\beta+1})_n \stackrel{\text{def}}{=} \gamma+\omega^\beta\cdot(n+1) \qquad\qquad (\gamma+\omega^\lambda)_n \stackrel{\text{def}}{=} \gamma+\omega^{\lambda_n}$$

Prop. $H^{\alpha+\beta}(n) = H^\alpha(H^\beta(n))$ for all $\alpha+\beta$ and n

Prop. $F_\alpha(n) = H^{\omega^\alpha}(n)$ for all α and n

Prop. $H^\alpha(n) \leqslant H^{\alpha'}(n')$ and $F_\alpha(n) \leqslant F_{\alpha'}(n')$ when $\alpha \sqsubseteq \alpha'$ & $n \leqslant n'$

Computing Hardy functions by rewriting

$$H^0(n) \stackrel{\text{def}}{=} n \qquad H^{\alpha+1}(n) \stackrel{\text{def}}{=} H^{\alpha}(n+1) \qquad H^{\lambda}(n) \stackrel{\text{def}}{=} H^{\lambda_n}(n)$$

seen as rewrite rules:

$$\langle \alpha+1, n\rangle \xrightarrow{H} \langle \alpha, n+1\rangle \qquad \langle \lambda, n\rangle \xrightarrow{H} \langle \lambda_n, n\rangle$$

Note (Tail-recursive implementation)
$H^{\alpha}(n)$ can be evaluated by rewriting a pair
$\alpha, n = \alpha_0, n_0 \xrightarrow{H} \alpha_1, n_1 \xrightarrow{H} \alpha_2, n_2 \xrightarrow{H} \cdots \xrightarrow{H} \alpha_k, n_k$ with
$\alpha_0 > \alpha_1 > \alpha_2 > \cdots$ until eventually $\alpha_k = 0$ and $n_k = H^{\alpha}(n)$

Below we compute fast-growing functions and their inverses
by encoding $\alpha, n \xrightarrow{H} \alpha', n'$ and $\alpha', n' \xrightarrow{H}{}^{-1} \alpha, n$

Computing Hardy functions by rewriting

$$H^0(n) \stackrel{\text{def}}{=} n \qquad H^{\alpha+1}(n) \stackrel{\text{def}}{=} H^{\alpha}(n+1) \qquad H^{\lambda}(n) \stackrel{\text{def}}{=} H^{\lambda_n}(n)$$

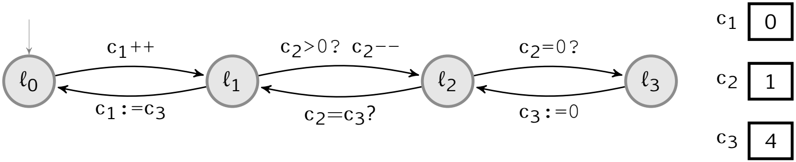
seen as rewrite rules:

$$\langle \alpha+1, n\rangle \xrightarrow{H} \langle \alpha, n+1\rangle \qquad \langle \lambda, n\rangle \xrightarrow{H} \langle \lambda_n, n\rangle$$

Note (Tail-recursive implementation)
$H^{\alpha}(n)$ can be evaluated by rewriting a pair
$\alpha, n = \alpha_0, n_0 \xrightarrow{H} \alpha_1, n_1 \xrightarrow{H} \alpha_2, n_2 \xrightarrow{H} \cdots \xrightarrow{H} \alpha_k, n_k$ with
$\alpha_0 > \alpha_1 > \alpha_2 > \cdots$ until eventually $\alpha_k = 0$ and $n_k = H^{\alpha}(n)$

Below we compute fast-growing functions and their inverses
by encoding $\alpha, n \xrightarrow{H} \alpha', n'$ and $\alpha', n' \xrightarrow{H}{}^{-1} \alpha, n$

CM = COUNTER MACHINES



A run of M: $(\ell_0,0,1,4) \to_{\text{rel}} (\ell_1,1,1,4) \to_{\text{rel}} (\ell_2,1,0,4) \to_{\text{rel}} (\ell_3,1,0,4)$

Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \not\leqslant (\ell_2,0,1,2)$.

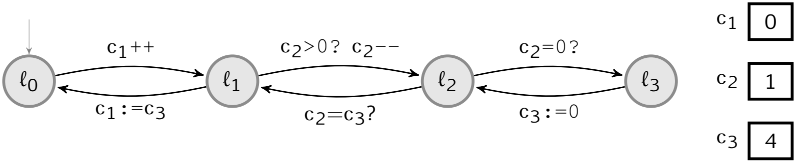
NB. A counter machine like M above is not monotonic.

Can test that a counter is zero $\Rightarrow$ steps not compatible with ordering

(And we allow other guards/updates that break compatibility).

In fact, the ordering is used to model unreliability.

CM = Counter Machines



A run of M: $(\ell_0,0,1,4) \to_{\text{rel}} (\ell_1,1,1,4) \to_{\text{rel}} (\ell_2,1,0,4) \to_{\text{rel}} (\ell_3,1,0,4)$

Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \not\leqslant (\ell_2,0,1,2)$.

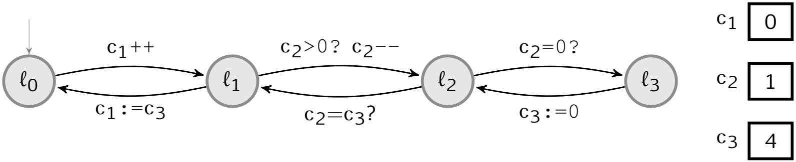
NB. A counter machine like M above is not monotonic.

Can test that a counter is zero $\Rightarrow$ steps not compatible with ordering

(And we allow other guards/updates that break compatibility).

In fact, the ordering is used to model unreliability.

CM = Counter Machines



A run of M: $(\ell_0,0,1,4) \to_{\text{rel}} (\ell_1,1,1,4) \to_{\text{rel}} (\ell_2,1,0,4) \to_{\text{rel}} (\ell_3,1,0,4)$

Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \not\leqslant (\ell_2,0,1,2)$.

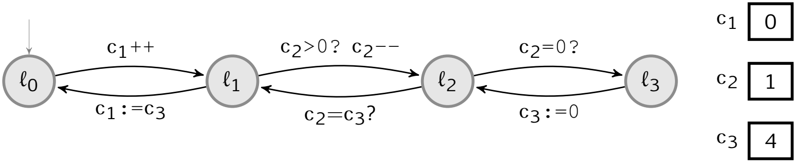
NB. A counter machine like M above is not monotonic.

Can test that a counter is zero $\Rightarrow$ steps not compatible with ordering

(And we allow other guards/updates that break compatibility).

In fact, the ordering is used to model unreliability.

CM = Counter Machines



A run of M: $(\ell_0,0,1,4) \rightarrow_{\text{rel}} (\ell_1,1,1,4) \rightarrow_{\text{rel}} (\ell_2,1,0,4) \rightarrow_{\text{rel}} (\ell_3,1,0,4)$

Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \not\leqslant (\ell_2,0,1,2)$.

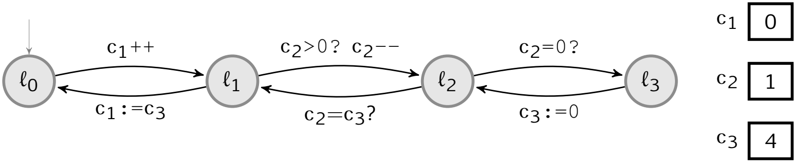
NB. A counter machine like M above is not monotonic.

Can test that a counter is zero $\Rightarrow$ steps not compatible with ordering

(And we allow other guards/updates that break compatibility).

In fact, the ordering is used to model unreliability.

CM = COUNTER MACHINES



A run of M: $(\ell_0,0,1,4) \rightarrow_{\text{rel}} (\ell_1,1,1,4) \rightarrow_{\text{rel}} (\ell_2,1,0,4) \rightarrow_{\text{rel}} (\ell_3,1,0,4)$

Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \not\leqslant (\ell_2,0,1,2)$.

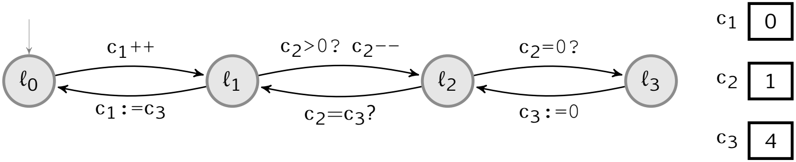
NB. A counter machine like M above is not monotonic.

Can test that a counter is zero $\Rightarrow$ steps not compatible with ordering

(And we allow other guards/updates that break compatibility).

In fact, the ordering is used to model unreliability.

LCM = *Lossy* Counter Machines



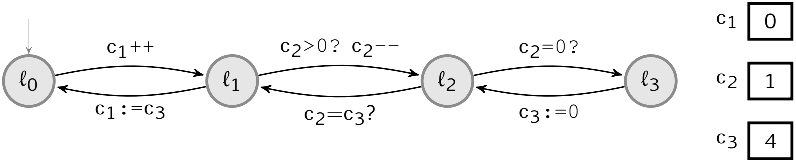
$$(\ell,\mathbf{a}) \to (\ell',\mathbf{b}) \overset{\text{def}}{\Leftrightarrow} (\ell,\mathbf{a}) \geqslant (\ell,\mathbf{x}) \to_{\text{rel}} (\ell',\mathbf{y}) \geqslant (\ell',\mathbf{b}) \text{ for some } \mathbf{x},\mathbf{y}$$

A run of M: $(\ell_0,0,1,4) \to (\ell_1,1,1,2) \to (\ell_2,1,0,2) \to (\ell_1,1,0,0)$

The unreliable counter machine is a WSTS

Paradox: It does much more than its reliable twin but can compute much less

LCM = *Lossy* COUNTER MACHINES



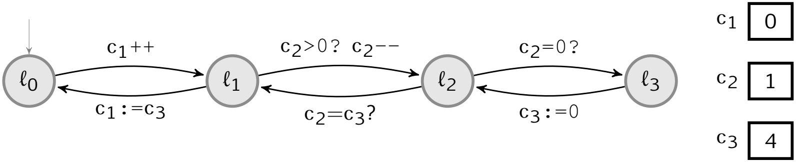
$$(\ell,\mathbf{a}) \to (\ell',\mathbf{b}) \overset{\text{def}}{\Leftrightarrow} (\ell,\mathbf{a}) \geqslant (\ell,\mathbf{x}) \to_{\text{rel}} (\ell',\mathbf{y}) \geqslant (\ell',\mathbf{b}) \text{ for some } \mathbf{x},\mathbf{y}$$

A run of M: $(\ell_0,0,1,4) \to (\ell_1,1,1,2) \to (\ell_2,1,0,2) \to (\ell_1,1,0,0)$

The unreliable counter machine is a WSTS

Paradox: It does much more than its reliable twin but can compute much less

LCM = *Lossy* Counter Machines



$$(\ell,\mathbf{a}) \to (\ell',\mathbf{b}) \overset{\text{def}}{\Leftrightarrow} (\ell,\mathbf{a}) \geqslant (\ell,\mathbf{x}) \to_{\text{rel}} (\ell',\mathbf{y}) \geqslant (\ell',\mathbf{b}) \text{ for some } \mathbf{x},\mathbf{y}$$

A run of M: $(\ell_0,0,1,4) \to (\ell_1,1,1,2) \to (\ell_2,1,0,2) \to (\ell_1,1,0,0)$

The unreliable counter machine is a WSTS

Paradox: It does much more than its reliable twin but can compute much less

ENCODING ORDINALS $< \omega^\omega$ IN TUPLES OF NUMBERS

Write α in CNF with coefficients
$\alpha = \omega^m.a_m + \omega^{m-1}.a_{m-1} + \cdots + \omega^0 a_0$

Encoding of α, n is $\langle a_m, \ldots, a_0; n\rangle \in \mathbb{N}^{m+2}$.

$$\langle a_m, \ldots, a_0 + 1; n\rangle \xrightarrow{H} \langle a_m, \ldots, a_0; n+1\rangle \qquad \%H^{\alpha+1}(n) = H^\alpha(n+$$

$$\langle a_m, \ldots, a_k + 1, \overbrace{0, \ldots, 0}^{k>0}; n\rangle \xrightarrow{H} \langle a_m, \ldots, a_k, n+1, \overbrace{0, \ldots, 0}^{k-1}; n\rangle \quad \%H^\lambda(n) = H^{\lambda_n}(n)$$

ENCODING ORDINALS $< \omega^\omega$ IN TUPLES OF NUMBERS

Write α in CNF with coefficients
$\alpha = \omega^m.a_m + \omega^{m-1}.a_{m-1} + \cdots + \omega^0 a_0$

Encoding of α, n is $\langle a_m, \ldots, a_0; n\rangle \in \mathbb{N}^{m+2}$.

$$\langle a_m, \ldots, a_0 + 1; n\rangle \xrightarrow{H} \langle a_m, \ldots, a_0; n+1\rangle \qquad \%H^{\alpha+1}(n) = H^{\alpha}(n+$$

$$\langle a_m, \ldots, a_k + 1, \overbrace{0, \ldots, 0}^{k>0}; n\rangle \xrightarrow{H} \langle a_m, \ldots, a_k, n+1, \overbrace{0, \ldots, 0}^{k-1}; n\rangle \quad \%H^{\lambda}(n) = H^{\lambda_n}(n)$$

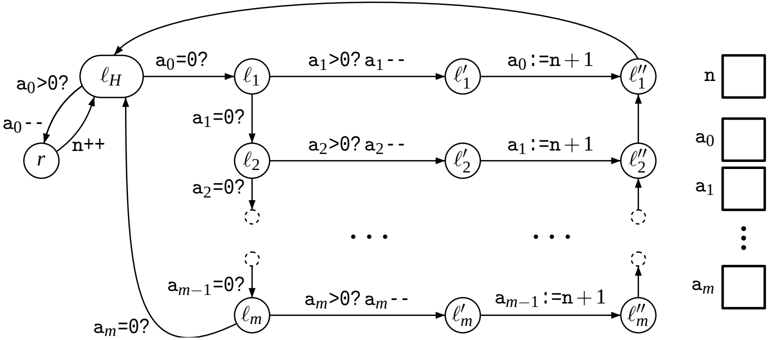
Recall: $(\gamma + \omega^{k+1})_n \stackrel{\text{def}}{=} \gamma + \omega^k \cdot (n+1)$

ENCODING ORDINALS $< \omega^\omega$ IN TUPLES OF NUMBERS

Write α in CNF with coefficients
$\alpha = \omega^m.a_m + \omega^{m-1}.a_{m-1} + \cdots + \omega^0 a_0$
Encoding of α, n is $\langle a_m, \ldots, a_0; n \rangle \in \mathbb{N}^{m+2}$.

$$\langle a_m, \ldots, a_0 {+} 1; n \rangle \xrightarrow{H} \langle a_m, \ldots, a_0; n{+}1 \rangle \qquad \% H^{\alpha+1}(n) = H^\alpha(n+$$

$$\langle a_m, \ldots, a_k {+} 1, \overbrace{0, \ldots, 0}^{k>0}; n \rangle \xrightarrow{H} \langle a_m, \ldots, a_k, n{+}1, \overbrace{0, \ldots, 0}^{k-1}; n \rangle \quad \% H^\lambda(n) = H^{\lambda_n}(n)$$



Now for $\xrightarrow{H}^{-1}$ (denoted $\xrightarrow{H^{-1}}$ from now on)

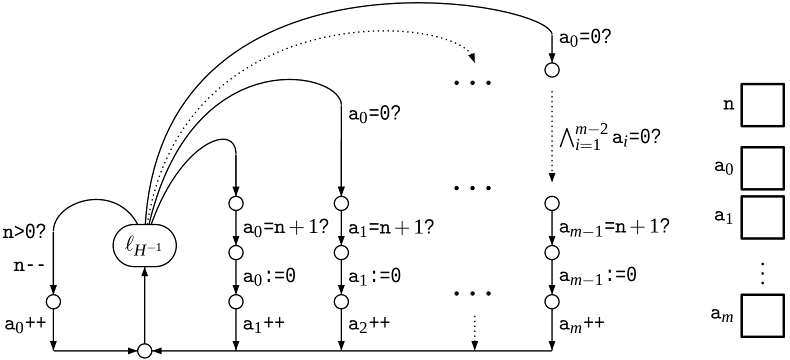
$$\langle a_m, \ldots, a_0; n+1\rangle \xrightarrow{H^{-1}} \langle a_m, \ldots, a_0+1; n\rangle \qquad \%H^{\alpha+1}(n) = H^{\alpha}(n$$

$$\langle a_m, \ldots, a_k, n+1, \overbrace{0, \ldots, 0}^{k-1}; n\rangle \xrightarrow{H^{-1}} \langle a_m, \ldots, a_k+1, \overbrace{0, \ldots, 0}^{k}; n\rangle \quad \%H^{\lambda}(n) = H^{\lambda_n}(n)$$

Now for $\xrightarrow{H}^{-1}$ (denoted $\xrightarrow{H^{-1}}$ from now on)

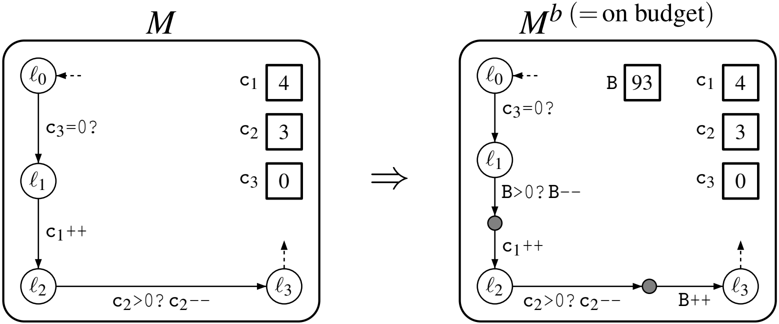
$$\langle a_m,\dots,a_0;n+1\rangle \xrightarrow{H^{-1}} \langle a_m,\dots,a_0+1;n\rangle \qquad \%H^{\alpha+1}(n)=H^{\alpha}(n -$$

$$\langle a_m,\dots,a_k,n+1,\overbrace{0,\dots,0}^{k-1};n\rangle \xrightarrow{H^{-1}} \langle a_m,\dots,a_k+1,\overbrace{0,\dots,0}^{k};n\rangle \quad \%H^{\lambda}(n)=H^{\lambda_n}(n)$$



Prop. [Robustness] $\mathbf{a} \leqslant_\times \mathbf{a}'$ and $n \leqslant n'$ imply $H^{\alpha}(n) \leqslant H^{\alpha'}(n')$

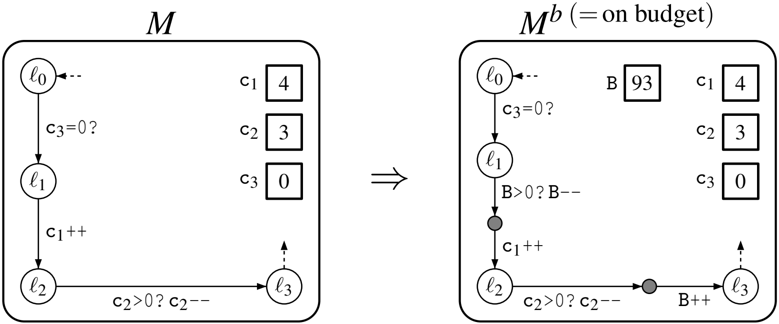
Counter machines on a budget



Ensures:

1. $\mathrm{M}^{\mathrm{b}} \vdash (\ell, \mathrm{B}, \mathbf{a}) \xrightarrow{*}_{\text{rel}} (\ell', \mathrm{B}', \mathbf{a}')$ implies $\mathrm{B} + |\mathbf{a}| = \mathrm{B}' + |\mathbf{a}'|$
2. $\mathrm{M}^{\mathrm{b}} \vdash (\ell, \mathrm{B}, \mathbf{a}) \xrightarrow{*}_{\text{rel}} (\ell', \mathrm{B}', \mathbf{a}')$ implies $\mathrm{M} \vdash (\ell, \mathbf{a}) \xrightarrow{*}_{\text{rel}} (\ell', \mathbf{a}')$
3. If $\mathrm{M} \vdash (\ell, \mathbf{a}) \xrightarrow{*}_{\text{rel}} (\ell', \mathbf{a}')$ then $\exists \mathrm{B}, \mathrm{B}'$: $\mathrm{M}^{\mathrm{b}} \vdash (\ell, \mathrm{B}, \mathbf{a}) \xrightarrow{*}_{\text{rel}} (\ell', \mathrm{B}', \mathbf{a}')$
4. If $\mathrm{M}^{\mathrm{b}} \vdash (\ell, \mathrm{B}, \mathbf{a}) \xrightarrow{*} (\ell', \mathrm{B}', \mathbf{a}')$
   then $\mathrm{M}^{\mathrm{b}} \vdash (\ell, \mathrm{B}, \mathbf{a}) \xrightarrow{*}_{\text{rel}} (\ell', \mathrm{B}', \mathbf{a}')$ iff $\mathrm{B} + |\mathbf{a}| = \mathrm{B}' + |\mathbf{a}'|$

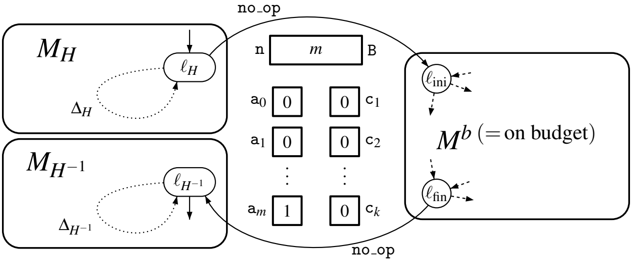
Counter machines on a budget



Ensures:

1. $M^b \vdash (\ell, B, \mathbf{a}) \xrightarrow{*}_{\mathsf{rel}} (\ell', B', \mathbf{a}')$ implies $B + |\mathbf{a}| = B' + |\mathbf{a}'|$
2. $M^b \vdash (\ell, B, \mathbf{a}) \xrightarrow{*}_{\mathsf{rel}} (\ell', B', \mathbf{a}')$ implies $M \vdash (\ell, \mathbf{a}) \xrightarrow{*}_{\mathsf{rel}} (\ell', \mathbf{a}')$
3. If $M \vdash (\ell, \mathbf{a}) \xrightarrow{*}_{\mathsf{rel}} (\ell', \mathbf{a}')$ then $\exists B, B'$: $M^b \vdash (\ell, B, \mathbf{a}) \xrightarrow{*}_{\mathsf{rel}} (\ell', B', \mathbf{a}')$
4. If $M^b \vdash (\ell, B, \mathbf{a}) \xrightarrow{*} (\ell', B', \mathbf{a}')$
   then $M^b \vdash (\ell, B, \mathbf{a}) \xrightarrow{*}_{\mathsf{rel}} (\ell', B', \mathbf{a}')$ iff $B + |\mathbf{a}| = B' + |\mathbf{a}'|$

$\mathrm{M}(m)$: Wrapping it up



Prop. $\mathrm{M}(m)$ has a lossy run

$$(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \xrightarrow{*} (\ell_{H-1}, 1, 0, \ldots, m, 0, \ldots)$$

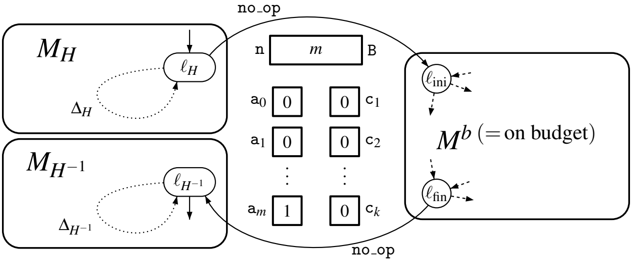
iff $\mathrm{M}(m)$ has a reliable run

$$(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \xrightarrow{*}_{\text{rel}} (\ell_{H-1}, a_m : 1, 0, \ldots, n : m, 0, \ldots)$$

iff M has a reliable run from ℓ_{ini} to ℓ_{fin} where all counters are bounded by $H^{\omega^m}(m)$, i.e., by $F_\omega(m) \approx$ *Ackermann*(m)

Cor. LCM verification is $\mathbb{F}_\omega$-hard, hence $\mathbb{F}_\omega$-complete

$\mathrm{M}(\mathrm{m})$: WRAPPING IT UP



Prop. $\mathrm{M}(\mathrm{m})$ has a lossy run

$$(\ell_H, \mathtt{a}_\mathrm{m} : 1, 0, \ldots, \mathtt{n} : \mathrm{m}, 0, \ldots) \xrightarrow{*} (\ell_{H-1}, 1, 0, \ldots, \mathrm{m}, 0, \ldots)$$

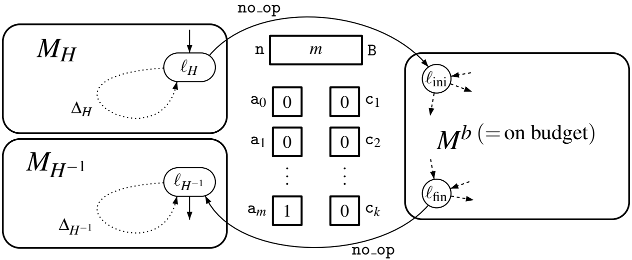
iff $\mathrm{M}(\mathrm{m})$ has a reliable run

$$(\ell_H, \mathtt{a}_\mathrm{m} : 1, 0, \ldots, \mathtt{n} : \mathrm{m}, 0, \ldots) \xrightarrow{*}_{\text{rel}} (\ell_{H-1}, \mathtt{a}_\mathrm{m} : 1, 0, \ldots, \mathtt{n} : \mathrm{m}, 0, \ldots)$$

iff M has a reliable run from ℓ_{ini} to ℓ_{fin} where all counters are bounded by $\mathrm{H}^{\omega^\mathrm{m}}(\mathrm{m})$, i.e., by $\mathrm{F}_\omega(\mathrm{m}) \approx$ *Ackermann*(m)

Cor. LCM verification is $\mathbb{F}_\omega$-hard, hence $\mathbb{F}_\omega$-complete

$M(m)$: WRAPPING IT UP



Prop. $M(m)$ has a lossy run

$$(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \xrightarrow{*} (\ell_{H-1}, 1, 0, \ldots, m, 0, \ldots)$$

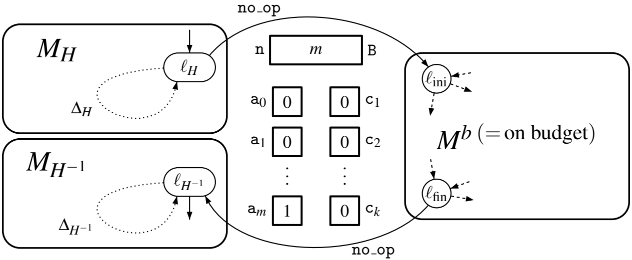
iff $M(m)$ has a reliable run

$$(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \xrightarrow{*}_{\text{rel}} (\ell_{H-1}, a_m : 1, 0, \ldots, n : m, 0, \ldots)$$

iff M has a reliable run from ℓ_{ini} to ℓ_{fin} where all counters are bounded by $H^{\omega^m}(m)$, i.e., by $F_\omega(m) \approx \textit{Ackermann}(m)$

Cor. LCM verification is $\mathbb{F}_\omega$-hard, hence $\mathbb{F}_\omega$-complete

$M(m)$: Wrapping it up



Prop. $M(m)$ has a lossy run

$$(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \xrightarrow{*} (\ell_{H-1}, 1, 0, \ldots, m, 0, \ldots)$$

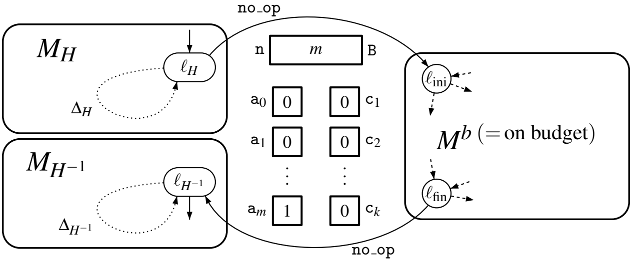
iff $M(m)$ has a reliable run

$$(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \xrightarrow{*}_{\text{rel}} (\ell_{H-1}, a_m : 1, 0, \ldots, n : m, 0, \ldots)$$

iff M has a reliable run from ℓ_{ini} to ℓ_{fin} where all counters are bounded by $H^{\omega^m}(m)$, i.e., by $F_\omega(m) \approx$ *Ackermann*(m)

Cor. LCM verification is $\mathbb{F}_\omega$-hard, hence $\mathbb{F}_\omega$-complete

$\mathcal{M}(m)$: WRAPPING IT UP



Prop. $\mathcal{M}(m)$ has a lossy run

$$(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \xrightarrow{*} (\ell_{H-1}, 1, 0, \ldots, m, 0, \ldots)$$

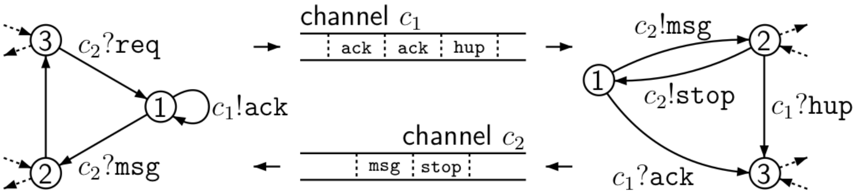
iff $\mathcal{M}(m)$ has a reliable run

$$(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \xrightarrow{*}_{\text{rel}} (\ell_{H-1}, a_m : 1, 0, \ldots, n : m, 0, \ldots)$$

iff M has a reliable run from ℓ_{ini} to ℓ_{fin} where all counters are bounded by $H^{\omega^m}(m)$, i.e., by $F_\omega(m) \approx \textit{Ackermann}(m)$

Cor. LCM verification is $\mathbf{F}_\omega$-hard, hence $\mathbf{F}_\omega$-complete

RECALL: LCS / LOSSY CHANNEL SYSTEMS



A configuration $\sigma = (\ell_1, \ell_2, w_1, w_2)$ with $w_i \in \Sigma^*$.
E.g., $w_1 = \mathtt{hup.ack.ack}$.

Reliable steps: $\sigma \rightarrow_{\text{rel}} \rho$ read in front of channels, write at end (FIFO)

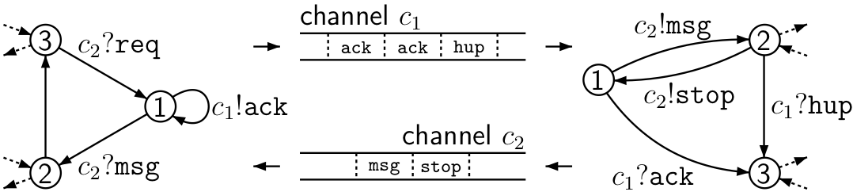
Lossy steps: messages may be lost nondeterministically

$$\sigma \rightarrow \sigma' \overset{\text{def}}{\Leftrightarrow} \sigma \sqsupseteq \rho \rightarrow_{\text{rel}} \rho' \sqsupseteq \sigma' \text{ for some } \rho, \rho'$$

where $(S, \sqsubseteq)$ is the wqo $(Loc_1, =) \times (Loc_2, =) \times (\Sigma^*, \leqslant_*)^{\{c_1, c_2\}}$

A model useful for concurrent protocols but also timed automata, metric temporal logic, products of modal logics, ...

Recall: LCS / Lossy Channel Systems



A configuration $\sigma = (\ell_1, \ell_2, w_1, w_2)$ with $w_i \in \Sigma^*$.
E.g., $w_1 = \texttt{hup.ack.ack}$.

Reliable steps: $\sigma \to_{\text{rel}} \rho$ read in front of channels, write at end (FIFO)

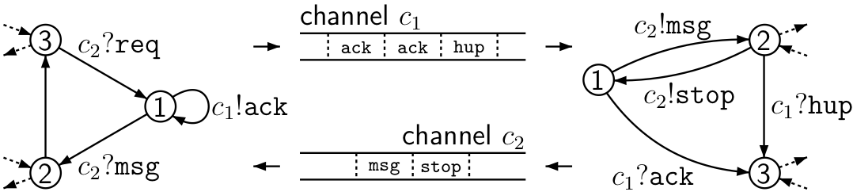
Lossy steps: messages may be lost nondeterministically

$$\sigma \to \sigma' \overset{\text{def}}{\Leftrightarrow} \sigma \sqsupseteq \rho \to_{\text{rel}} \rho' \sqsupseteq \sigma' \text{ for some } \rho, \rho'$$

where $(S, \sqsubseteq)$ is the wqo $(Loc_1, =) \times (Loc_2, =) \times (\Sigma^*, \leqslant_*)^{\{c_1, c_2\}}$

A model useful for concurrent protocols but also timed automata, metric temporal logic, products of modal logics, ...

Recall: LCS / Lossy Channel Systems



A configuration $\sigma = (\ell_1, \ell_2, w_1, w_2)$ with $w_i \in \Sigma^*$.
E.g., $w_1 = \mathtt{hup.ack.ack}$.

Reliable steps: $\sigma \to_{\text{rel}} \rho$ read in front of channels, write at end (FIFO)

Lossy steps: messages may be lost nondeterministically

$$\sigma \to \sigma' \stackrel{\text{def}}{\Leftrightarrow} \sigma \sqsupseteq \rho \to_{\text{rel}} \rho' \sqsupseteq \sigma' \text{ for some } \rho, \rho'$$

where $(S, \sqsubseteq)$ is the wqo $(Loc_1, =) \times (Loc_2, =) \times (\Sigma^*, \leqslant_*)^{\{c_1, c_2\}}$

A model useful for concurrent protocols but also timed automata, metric temporal logic, products of modal logics, ...

Encoding ordinals $< \omega^{\omega^\omega}$ in channels

We use $\Sigma = \{a_0,\ldots,a_m\} \cup \{|\}$ to encode ordinals $\alpha < \omega^{\omega^{m+1}}$.

Two-level "differential" encoding:

$\beta : \{a_0,\ldots,a_m\}^* \to \omega^{m+1}$

$\beta(a_{r_1}\ldots a_{r_k}) \stackrel{\text{def}}{=} \omega^{r_1} + \cdots + \omega^{r_k}$

E.g. $\beta(\varepsilon) = 0$, $\beta(a_3a_0a_0) = \omega^3 + 2$, $\beta(a_0a_0a_3) = 2 + \omega^3 = \omega^3$

$\alpha : \Sigma^* \to \omega^{\omega^{m+1}}$

$\alpha(a_1|a_2|\ldots a_l|) \stackrel{\text{def}}{=} \omega^{\beta(a_1a_2\ldots a_l)} + \cdots + \omega^{\beta(a_1a_2)} + \omega^{\beta(a_1)}$

E.g. $\alpha(|||) = \omega^0 + \omega^0 + \omega^0 = 3$ $\qquad \alpha(a_1a_0||a_1|) = \omega^{\omega\cdot 2} + \omega^{\omega+1}\cdot 2$

Difficulties. 1: $\alpha(w)$ is not always a CNF

2: $w \leqslant_* w'$ implies $\alpha(w) \leqslant \alpha(w')$ but not necessarily $\alpha(w) \sqsubseteq \alpha(w')$

Encoding ordinals $< \omega^{\omega^\omega}$ in channels

We use $\Sigma = \{a_0, \ldots, a_m\} \cup \{|\}$ to encode ordinals $\alpha < \omega^{\omega^{m+1}}$.

Two-level "differential" encoding:

$\beta : \{a_0, \ldots, a_m\}^* \to \omega^{m+1}$
$\beta(a_{r_1} \ldots a_{r_k}) \stackrel{\text{def}}{=} \omega^{r_1} + \cdots + \omega^{r_k}$
E.g. $\beta(\varepsilon) = 0$, $\beta(a_3 a_0 a_0) = \omega^3 + 2$, $\beta(a_0 a_0 a_3) = 2 + \omega^3 = \omega^3$

$\alpha : \Sigma^* \to \omega^{\omega^{m+1}}$
$\alpha(a_1 | a_2 | \ldots a_l |) \stackrel{\text{def}}{=} \omega^{\beta(a_1 a_2 \ldots a_l)} + \cdots + \omega^{\beta(a_1 a_2)} + \omega^{\beta(a_1)}$

E.g. $\alpha(|||) = \omega^0 + \omega^0 + \omega^0 = 3$ $\qquad$ $\alpha(a_1 a_0 || a_1 |) = \omega^{\omega \cdot 2} + \omega^{\omega+1} \cdot 2$

Difficulties. 1: $\alpha(w)$ is not always a CNF

2: $w \leqslant_* w'$ implies $\alpha(w) \leqslant \alpha(w')$ but not necessarily $\alpha(w) \sqsubseteq \alpha(w')$

ENCODING ORDINALS $< \omega^{\omega^\omega}$ IN CHANNELS

We use $\Sigma = \{a_0, \ldots, a_m\} \cup \{|\}$ to encode ordinals $\alpha < \omega^{\omega^{m+1}}$.

Two-level "differential" encoding:

$\beta : \{a_0, \ldots, a_m\}^* \to \omega^{m+1}$
$\beta(a_{r_1} \ldots a_{r_k}) \stackrel{\text{def}}{=} \omega^{r_1} + \cdots + \omega^{r_k}$
E.g. $\beta(\varepsilon) = 0$, $\beta(a_3 a_0 a_0) = \omega^3 + 2$, $\beta(a_0 a_0 a_3) = 2 + \omega^3 = \omega^3$

$\alpha : \Sigma^* \to \omega^{\omega^{m+1}}$
$\alpha(\mathbf{a}_1 | \mathbf{a}_2 | \ldots \mathbf{a}_l |) \stackrel{\text{def}}{=} \omega^{\beta(\mathbf{a}_1 \mathbf{a}_2 \ldots \mathbf{a}_l)} + \cdots + \omega^{\beta(\mathbf{a}_1 \mathbf{a}_2)} + \omega^{\beta(\mathbf{a}_1)}$

E.g. $\alpha(|||) = \omega^0 + \omega^0 + \omega^0 = 3$ $\qquad \alpha(a_1 a_0 || a_1 |) = \omega^{\omega \cdot 2} + \omega^{\omega+1} \cdot 2$

Difficulties. 1: $\alpha(w)$ is not always a CNF

2: $w \leqslant_* w'$ implies $\alpha(w) \leqslant \alpha(w')$ but not necessarily $\alpha(w) \sqsubseteq \alpha(w')$

Encoding ordinals $< \omega^{\omega^\omega}$ in channels

We use $\Sigma = \{a_0, \ldots, a_m\} \cup \{|\}$ to encode ordinals $\alpha < \omega^{\omega^{m+1}}$.

Two-level "differential" encoding:

$\beta : \{a_0, \ldots, a_m\}^* \to \omega^{m+1}$

$\beta(a_{r_1} \ldots a_{r_k}) \stackrel{\text{def}}{=} \omega^{r_1} + \cdots + \omega^{r_k}$

E.g. $\beta(\varepsilon) = 0$, $\beta(a_3 a_0 a_0) = \omega^3 + 2$, $\beta(a_0 a_0 a_3) = 2 + \omega^3 = \omega^3$

$\alpha : \Sigma^* \to \omega^{\omega^{m+1}}$

$\alpha(\mathbf{a}_1 | \mathbf{a}_2 | \ldots \mathbf{a}_l |) \stackrel{\text{def}}{=} \omega^{\beta(\mathbf{a}_1 \mathbf{a}_2 \ldots \mathbf{a}_l)} + \cdots + \omega^{\beta(\mathbf{a}_1 \mathbf{a}_2)} + \omega^{\beta(\mathbf{a}_1)}$

E.g. $\alpha(|||) = \omega^0 + \omega^0 + \omega^0 = 3$ $\quad$ $\alpha(a_1 a_0 || a_1 |) = \omega^{\omega \cdot 2} + \omega^{\omega+1} \cdot 2$

Difficulties. 1: $\alpha(w)$ is not always a CNF

2: $w \leqslant_* w'$ implies $\alpha(w) \leqslant \alpha(w')$ but not necessarily $\alpha(w) \sqsubseteq \alpha(w')$

Weakly computing $\xrightarrow{H}$ with LCS's

$$
\begin{aligned}
(|w,n) &\xrightarrow{H} (w,n+1) && \% H^{\alpha+1}(n) = H^{\alpha}(n+1)\\
(ua_0|w,n) &\xrightarrow{H} (u|^{n+1}a_0w,n) && \% H^{\gamma+\omega^{k+1}}(n) = H^{\gamma+\omega^{k}\cdot(n+1)}(n)\\
(ua_{r+1}|w,n) &\xrightarrow{H} (ua_r^{n+1}|a_rw,n) && \% H^{\gamma+\omega^{\beta+\omega^{k+1}}}(n) = H^{\gamma+\omega^{\beta+\omega^{k}\cdot(n+1)}}(n)
\end{aligned}
$$

$(\cdots$ similar rules for $\xrightarrow{H^{-1}}$ $\cdots)$

Prop. [Robustness]
$w \leqslant_* w'$ and $n \leqslant n'$ and w' pure imply $H^{\alpha(w)}(n) \leqslant H^{\alpha(w')}(n')$

where purity means that w' has no superfluous symbols
(a regular condition that can be enforced by LCS's)

WEAKLY COMPUTING $\xrightarrow{H}$ WITH LCS'S

$$(|w,n) \xrightarrow{H} (w,n+1) \qquad \%H^{\alpha+1}(n) = H^{\alpha}(n+1)$$

$$(ua_0|w,n) \xrightarrow{H} (u|^{n+1}a_0w,n) \qquad \%H^{\gamma+\omega^{k+1}}(n) = H^{\gamma+\omega^{k}\cdot(n+1)}(n)$$

$$(ua_{r+1}|w,n) \xrightarrow{H} (ua_r^{n+1}|a_rw,n) \qquad \%H^{\gamma+\omega^{\beta+\omega^{k+1}}}(n) = H^{\gamma+\omega^{\beta+\omega^{k}\cdot(n+1)}}(n)$$

$(\cdots$ similar rules for $\xrightarrow{H^{-1}} \cdots)$

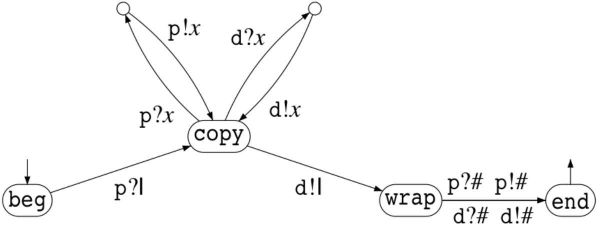
Prop. [Robustness]
$w \leqslant_* w'$ and $n \leqslant n'$ and w' pure imply $H^{\alpha(w)}(n) \leqslant H^{\alpha(w')}(n')$

where purity means that w' has no superfluous symbols
(a regular condition that can be enforced by LCS's)

COMPUTING $\xrightarrow{\text{H}}$ WITH LCS'S: FIRST RULE

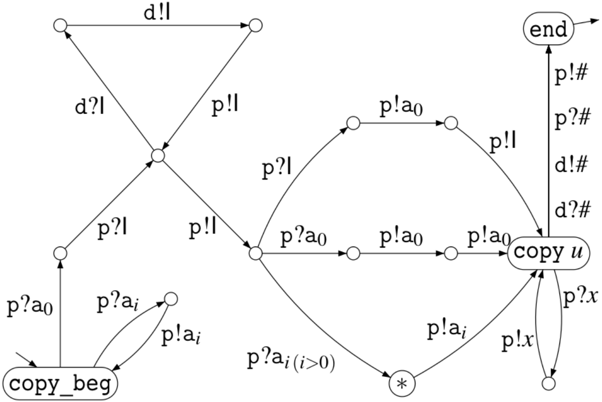
We now store u and $|^n$ as two strings (with endmarker $\#$) on two channels p and d.

$$\begin{array}{ll} p: & |u\# \\ d: & |^n\# \end{array} \xrightarrow{*} \begin{array}{l} u\# \\ |^{n+1}\# \end{array}$$



COMPUTING $\xrightarrow{\mathrm{H}}$ WITH LCS'S: SECOND RULE

$$\begin{array}{ll} \mathtt{p}: & a_{i_1}\ldots a_{i_p} a_0 \mathsf{I} u\# \\ & \\ \mathtt{d}: & \mathsf{I}^n\# \end{array} \quad \xrightarrow{*} \quad \begin{array}{l} a_{i_1}\ldots a_{i_p} \mathsf{I}^{n+1} a_0 u\# \\ \\ \mathsf{I}^n\# \end{array}$$



WRAPPING IT UP (SKETCHILY)

As we did for lossy counter machines, this time with channels

Bottom line: a LCS with $|\Sigma| = m + 3$
— can build a workspace of size
$H^{\omega^{\omega^{m+1}}}(m) = H^{\omega^{\omega^{\omega}}}(m) = F_{\omega^\omega}(m)$,
— use this as a computational resource,
— and fold back the workspace by computing the inverse of H

Checking that the above computation is performed reliably can be stated as (reduces to) a reachability (or termination) question

Cor. LCS verification is hard for $\mathbb{F}_{\omega^\omega}$, hence $\mathbb{F}_{\omega^\omega}$-complete

Confirms: the main parameter for complexity is the size of the message alphabet

Wrapping it up (sketchily)

As we did for lossy counter machines, this time with channels

Bottom line: a LCS with $|\Sigma| = m+3$
— can build a workspace of size
$H^{\omega^{\omega^{m+1}}}(m) = H^{\omega^{\omega^{\omega}}}(m) = F_{\omega^\omega}(m)$,
— use this as a computational resource,
— and fold back the workspace by computing the inverse of H

Checking that the above computation is performed reliably can be stated as (reduces to) a reachability (or termination) question

Cor. LCS verification is hard for $\mathbf{F}_{\omega^\omega}$, hence $\mathbf{F}_{\omega^\omega}$-complete

Confirms: the main parameter for complexity is the size of the message alphabet

Wrapping it up (sketchily)

As we did for lossy counter machines, this time with channels

Bottom line: a LCS with $|\Sigma| = m + 3$
— can build a workspace of size
$H^{\omega^{\omega^{m+1}}}(m) = H^{\omega^{\omega^{\omega}}}(m) = F_{\omega^\omega}(m)$,
— use this as a computational resource,
— and fold back the workspace by computing the inverse of H

Checking that the above computation is performed reliably can be stated as (reduces to) a reachability (or termination) question

Cor. LCS verification is hard for $\mathbf{F}_{\omega^\omega}$, hence $\mathbf{F}_{\omega^\omega}$-complete

Confirms: the main parameter for complexity is the size of the message alphabet

CONCLUSION FOR LAST TWO LECTURES

Length of bad sequences is key to bounding the complexity of WQO-based algorithms

Here computer scientists need results/theories from other fields: proof-theory and combinatorics

Proving matching lower bounds is not necessarily tricky (and is easy for LCM's or LCS's) but we still lack:

— a collection of hard problems: Post Embedding Problem, . . .

— a tutorial/textbook on subrecursive hierarchies (like fast-growing and Hardy hierarchies)

— a toolkit of coding tricks for computing with ordinals

— a large enough user community

The approach is workable: we could characterize the complexity of Timed-Arc Petri nets and Data Petri Nets at level $\mathbb{F}_{\omega^{\omega^{\omega}}}$

Conclusion for last two lectures

Length of bad sequences is key to bounding the complexity of WQO-based algorithms

Here computer scientists need results/theories from other fields: proof-theory and combinatorics

Proving matching lower bounds is not necessarily tricky (and is easy for LCM's or LCS's) but we still lack:

— a collection of hard problems: Post Embedding Problem, . . .

— a tutorial/textbook on subrecursive hierarchies (like fast-growing and Hardy hierarchies)

— a toolkit of coding tricks for computing with ordinals

— a large enough user community

The approach is workable: we could characterize the complexity of Timed-Arc Petri nets and Data Petri Nets at level $\mathbb{F}_{\omega^{\omega^\omega}}$

CONCLUSION FOR LAST TWO LECTURES

Length of bad sequences is key to bounding the complexity of WQO-based algorithms

Here computer scientists need results/theories from other fields: proof-theory and combinatorics

Proving matching lower bounds is not necessarily tricky (and is easy for LCM's or LCS's) but we still lack:

— a collection of hard problems: Post Embedding Problem, . . .

— a tutorial/textbook on subrecursive hierarchies (like fast-growing and Hardy hierarchies)

— a toolkit of coding tricks for computing with ordinals

— a large enough user community

The approach is workable: we could characterize the complexity of Timed-Arc Petri nets and Data Petri Nets at level $\mathbb{F}_{\omega^{\omega^{\omega}}}$

Conclusion for last two lectures

Length of bad sequences is key to bounding the complexity of WQO-based algorithms

Here computer scientists need results/theories from other fields: proof-theory and combinatorics

Proving matching lower bounds is not necessarily tricky (and is easy for LCM's or LCS's) but we still lack:

— a collection of hard problems: Post Embedding Problem, ...

— a tutorial/textbook on subrecursive hierarchies (like fast-growing and Hardy hierarchies)

— a toolkit of coding tricks for computing with ordinals

— a large enough user community

The approach is workable: we could characterize the complexity of Timed-Arc Petri nets and Data Petri Nets at level $\mathbb{F}_{\omega^{\omega^{\omega}}}$