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

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

If a sequence like \(x\) cannot grow too quickly — we say it is controlled — then the position \(i, j\) of the first increasing pair in \(x\) 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 \(\mathcal{IF}_\omega\) upper-bounds for the verification — e.g., termination and/or safety — of monotonic counter machines, and \(\mathcal{IF}_\omega \omega\) upper bounds for lossy channel systems.
IF YOU MISSED THE EARLIER EPISODES

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

If a sequence like \(x\) cannot grow too quickly — we say it is controlled — then the position \(i, j\) of the first increasing pair in \(x\) 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 \(F_{\omega}\) upper-bounds for the verification — e.g., termination and/or safety — of monotonic counter machines, and \(F_{\omega\omega}\) upper bounds for lossy channel systems

That was just the EASY part!!!
IF YOU MISSED THE EARLIER EPISODES

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

If a sequence like $x$ cannot grow too quickly —we say it is controlled— then the position $i, j$ of the first increasing pair in $x$ 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 $\mathsf{F}_\omega^\omega$ upper-bounds for the verification —e.g., termination and/or safety— of monotonic counter machines, and $\mathsf{F}_\omega^\omega$ 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 $\text{Ack}(n) \overset{\text{def}}{=} A(n,n)$ 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 $i$th prime (resp., Fibonacci) number

**Pb 5.** Input: $x$. Question: Does the Universal Turing machine halts on $x$ in at most $\text{Ack}(|x|)$ 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 \( \text{Ack}(n) \triangleq A(n,n) \) 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 \( i \)th prime (resp., Fibonacci) number?

**Pb 5.** Input: \( x \). Question: Does the Universal Turing machine halts on \( x \) in at most \( \text{Ack}(|x|) \) 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 $\text{Ack}(n) \defeq \Lambda(n, n)$ and $\Lambda$ 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 $i$th prime (resp., Fibonacci) number?

**Pb 5.** Input: $x$. Question: Does the Universal Turing machine halts on $x$ in at most $\text{Ack}(|x|)$ 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 $\text{Ack}(n) \overset{\text{def}}{=} A(n,n)$ 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 $\sum_{i \in K} p_{F_i}$? where $p_i$ and $F_i$ are the $i$th prime (resp., Fibonacci) number

Pb 5. Input: $x$. Question: Does the Universal Turing machine halts on $x$ in at most $\text{Ack}(|x|)$ 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 \( \text{Ack}(n) \triangleq A(n, n) \) 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 \( i \)th prime (resp., Fibonacci) number?

**Pb 5.** Input: \( x \). Question: Does the Universal Turing machine halts on \( x \) in at most \( \text{Ack}(|x|) \) steps?
We shall adopt the following strategy:

1. Compute unreliably a function in the Fast-Growing hierarchy
2. Use the result as an unreliable computational resource
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**

\[
\begin{align*}
F_0(n) & \overset{\text{def}}{=} n + 1 \\
F_{\alpha+1}(n) & \overset{\text{def}}{=} F_{\alpha}^{n+1}(n) = F_{\alpha}(F_{\alpha}(\ldots F_{\alpha}(n) \ldots)) \\
F_\lambda(n) & \overset{\text{def}}{=} F_\lambda(n)
\end{align*}
\]

with

\[
(\gamma + \omega^{\beta+1})_n \overset{\text{def}}{=} \gamma + \omega^\beta \cdot (n + 1)
\]

\[
(\gamma + \omega^\lambda)_n \overset{\text{def}}{=} \gamma + \omega^\lambda
\]

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 \( \alpha \) and \( n \)

Prop. \( H^\alpha(n) \leq H^{\alpha'}(n') \) and \( F_\alpha(n) \leq F_{\alpha'}(n') \) when \( \alpha \sqsubseteq \alpha' \) & \( n \leq n' \)
**Fast-Growing vs. Hardy Hierarchy**

\[
\begin{align*}
F_0(n) & \overset{\text{def}}{=} n + 1 \\
F_{\alpha+1}(n) & \overset{\text{def}}{=} F^{n+1}_\alpha(n) = F_\alpha(F_\alpha(...F_\alpha(n)...)) \\
F_\lambda(n) & \overset{\text{def}}{=} F_{\lambda n}(n)
\end{align*}
\]

with
\[
(\gamma + \omega^{\beta+1})_n \overset{\text{def}}{=} \gamma + \omega^\beta \cdot (n + 1)
\]

\[
\begin{align*}
H^0(n) & \overset{\text{def}}{=} n \\
H^{\alpha+1}(n) & \overset{\text{def}}{=} H_\alpha(n + 1) \\
H_\lambda(n) & \overset{\text{def}}{=} H^\lambda_n(n)
\end{align*}
\]

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 \( \alpha \) and \( n \)

Prop. \( H^\alpha(n) \preceq H^{\alpha'}(n') \) and \( F_\alpha(n) \preceq F_{\alpha'}(n') \) when \( \alpha \sqsubseteq \alpha' \) & \( n \leq n' \)
Computing Hardy Functions by Rewriting

\[ H^0(n) \overset{\text{def}}{=} n \quad H^{\alpha+1}(n) \overset{\text{def}}{=} H^\alpha(n + 1) \quad H^\lambda(n) \overset{\text{def}}{=} H^{\lambda n}(n) \]

seen as rewrite rules:

\[ \langle \alpha + 1, n \rangle \xrightarrow{H} \langle \alpha, n + 1 \rangle \quad \langle \lambda, n \rangle \xrightarrow{H} \langle \lambda n, n \rangle \]

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

\[ H^0(n) \overset{\text{def}}{=} n \quad H^{\alpha+1}(n) \overset{\text{def}}{=} H^\alpha(n + 1) \quad H^\lambda(n) \overset{\text{def}}{=} H^\lambda n(n) \]

seen as rewrite rules:

\[ \langle \alpha + 1, n \rangle \xrightarrow{H} \langle \alpha, n + 1 \rangle \quad \langle \lambda, n \rangle \xrightarrow{H} \langle \lambda n, n \rangle \]

**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 \( \alpha', n' \xrightarrow{H}^{-1} \alpha, n \)
\textbf{CM = COUNTER MACHINES}

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

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

\textbf{NB.} A counter machine like $M$ above is \textbf{not} monotonic.

Can test that a counter is zero $\Rightarrow$ steps \textbf{not compatible} with ordering (And we allow other guards/updates that break compatibility).

\textbf{In fact}, the ordering is used to model \textbf{unreliability}.
CM = COUNTER MACHINES

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

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

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.
A run of $M$: $(l_0,0,1,4) \rightarrow_{\text{rel}} (l_1,1,1,4) \rightarrow_{\text{rel}} (l_2,1,0,4) \rightarrow_{\text{rel}} (l_3,1,0,4)$

Ordering states: $(l_1,0,0,0) \leq (l_1,0,1,2)$ but $(l_1,0,0,0) \not\leq (l_2,0,1,2)$.

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

Ordering states: $(\ell_1,0,0,0) \preceq (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \npreceq (\ell_2,0,1,2)$.

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

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

**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

\[
\begin{align*}
\ell_0 & \rightarrow c_1++ \ell_1 \\
c_1 := c_3 & \rightarrow c_2>0? c_2-- \ell_2 \\
c_2=c_3? & \rightarrow c_2=0? \ell_3 \\
c_3 := 0 &
\end{align*}
\]

\[(\ell, a) \rightarrow (\ell', b) \overset{\text{def}}{\iff} (\ell, a) \geq (\ell, x) \rightarrow_{\text{rel}} (\ell', y) \geq (\ell', b) \text{ for some } x, y\]

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

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

\[
\ell_0 \xrightarrow{c_1++} \ell_1 \xrightarrow{c_2>0\?\ c_2--} \ell_2 \xrightarrow{c_2=0\?} \ell_3
\]

\[
\ell_0 = (0, 1, 4) \rightarrow (\ell_1, 1, 1, 2) \rightarrow (\ell_2, 1, 0, 2) \rightarrow (\ell_1, 1, 0, 0)
\]

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

\[(\ell, a) \rightarrow (\ell', b) \overset{\text{def}}{\iff} (\ell, a) \succeq (\ell, x) \rightarrow_{\text{rel}} (\ell', y) \succeq (\ell', b) \text{ for some } x, y\]

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

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

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

$\langle a_m, \ldots, a_0+1; n \rangle \xrightarrow{H} \langle a_m, \ldots, a_0; n+1 \rangle$

$\langle a_m, \ldots, a_k+1, 0, \ldots, 0; n \rangle \xrightarrow{H} \langle a_m, \ldots, a_k, n+1, 0, \ldots, 0; n \rangle$

$\%H^{\alpha+1}(n) = H^{\alpha}(n+1)$

$\%H^{\lambda}(n) = H^{\lambda n}(n)$
ENCODING ORDINALS $< \omega^\omega$ IN TUPLES OF NUMBERS

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

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

\[
\begin{align*}
\langle a_m, \ldots, a_0+1; n \rangle & \xrightarrow{H} \langle a_m, \ldots, a_0; n+1 \rangle \\
\langle a_m, \ldots, a_k+1, 0, \ldots, 0; n \rangle & \xrightarrow{H} \langle a_m, \ldots, a_k, n+1, 0, \ldots, 0; n \rangle \\
\end{align*}
\]

Recall: $(\gamma + \omega^{k+1})_n \overset{\text{def}}{=} \gamma + \omega^k \cdot (n + 1)$
ENCODING ORDINALS $< \omega^\omega$ IN TUPLES OF NUMBERS

Write $\alpha$ in CNF with coefficients

$\alpha = \omega^m a_m + \omega^{m-1} a_{m-1} + \cdots + \omega^0 a_0$

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

$\langle a_m, \ldots, a_0+1; n \rangle \xrightarrow{H} \langle a_m, \ldots, a_0; n+1 \rangle$

$\langle a_m, \ldots, a_k+1, 0, \ldots, 0; n \rangle \xrightarrow{H} \langle a_m, \ldots, a_k, n+1, 0, \ldots, 0; n \rangle$

$\%H^{\alpha+1}(n) = H^\alpha(n+1)$

$\%H^\lambda(n) = H^\lambda(n)$
NOW FOR $\xrightarrow{H^{-1}}$ (DENOTED $\xrightarrow{H^{-1}}$ FROM NOW ON)

$$\langle a_m, \ldots, a_0; n+1 \rangle \xrightarrow{H^{-1}} \langle a_m, \ldots, a_0+1; n \rangle$$

$$\langle a_m, \ldots, a_k, n+1, 0, \ldots, 0; n \rangle \xrightarrow{H^{-1}} \langle a_m, \ldots, a_k+1, 0, \ldots, 0; n \rangle$$

$\%H^{\alpha+1}(n) = H^\alpha(n - 1)$

$\%H^\lambda(n) = H^\lambda(n)$
NOW FOR $\frac{H}{H^{-1}}$ (DENOTED $\frac{H^{-1}}{}$ FROM NOW ON)

$$\langle a_m, \ldots, a_0; n+1 \rangle \xrightarrow{H^{-1}} \langle a_m, \ldots, a_0+1; n \rangle$$

$\%H^{\alpha+1}(n) = H^{\alpha}(n-1)$

$$\langle a_m, \ldots, a_k, n+1, 0, \ldots, 0; n \rangle \xrightarrow{H^{-1}} \langle a_m, \ldots, a_k+1, 0, \ldots, 0; n \rangle$$

$\%H^{\lambda}(n) = H^{\lambda n}(n)$

Prop. [Robustness] $a \preceq \times a'$ and $n \leq n'$ imply $H^{\alpha}(n) \leq H^{\alpha'}(n')$
Ensures:
1. $M^b \vdash (\ell, B, a) \xrightarrow{\text{rel}} (\ell', B', a')$ implies $B + |a| = B' + |a'|$
2. $M^b \vdash (\ell, B, a) \xrightarrow{\text{rel}} (\ell', B', a')$ implies $M \vdash (\ell, a) \xrightarrow{\text{rel}} (\ell', a')$
3. If $M \vdash (\ell, a) \xrightarrow{\text{rel}} (\ell', a')$ then $\exists B, B': M^b \vdash (\ell, B, a) \xrightarrow{\text{rel}} (\ell', B', a')$
4. If $M^b \vdash (\ell, B, a) \xrightarrow{\text{rel}} (\ell', B', a')$
   then $M^b \vdash (\ell, B, a) \xrightarrow{\text{rel}} (\ell', B', a')$ iff $B + |a| = B' + |a'|$
COUNTER MACHINES ON A BUDGET

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

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

iff $M$ has a reliable run from $\ell_{\text{ini}}$ to $\ell_{\text{fin}}$ where all counters are bounded by $H^ω(m)$, i.e., by $F_ω(m) \approx \text{Ackermann}(m)$

**Cor.** LCM verification is $F_ω$-hard, hence $F_ω$-complete
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 \( \ell_{\text{ini}} \) to \( \ell_{\text{fin}} \) where all counters are bounded by \( H^m(\omega)^m(m) \), i.e., by \( F_\omega(m) \approx \text{Ackermann}(m) \)

Cor. LCM verification is \( F_\omega \)-hard, hence \( F_\omega \)-complete
Prop. $M(m)$ has a lossy run

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

iff $M(m)$ has a reliable run

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

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

Cor. LCM verification is $F_\omega$-hard, hence $F_\omega$-complete
Prop. \( M(m) \) has a lossy run

\[
(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \rightarrow^* (\ell_{H-1}, 1, 0, \ldots, m, 0, \ldots)
\]

iff \( M(m) \) has a reliable run

\[
(\ell_H, a_m : 1, 0, \ldots, n : m, 0, \ldots) \rightarrow_{\text{rel}} (\ell_{H-1}, a_m : 1, 0, \ldots, n : m, 0, \ldots)
\]

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

Cor. LCM verification is \( F_\omega \)-hard, hence \( F_\omega \)-complete
Prop. $M(m)$ has a lossy run

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

iff $M(m)$ has a reliable run

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

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

Cor. LCM verification is $F_\omega$-hard, hence $F_\omega$-complete
RECALL: LCS / LOSSY CHANNEL SYSTEMS

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

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

Lossy steps: messages may be lost nondeterministically  
\[ \sigma \rightarrow \sigma' \iff \sigma \sqsubseteq \rho \rightarrow_{\text{rel}} \rho' \sqsubseteq \sigma' \text{ for some } \rho, \rho' \]  
where $(S, \sqsubseteq)$ is the wqo $(\text{Loc}_1, =) \times (\text{Loc}_2, =) \times (\Sigma^*, \leq_*)^{\{c_1, c_2\}}$  

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., $w_1 = \text{hup.ack.ack}$.

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

Lossy steps: messages may be lost nondeterministically

$$\sigma \rightarrow \sigma' \iff \sigma \sqsubseteq \rho \rightarrow_{\text{rel}} \rho' \sqsubseteq \sigma' \text{ for some } \rho, \rho'$$

where $(S, \sqsubseteq)$ is the wqo $(\text{Loc}_1, =) \times (\text{Loc}_2, =) \times (\Sigma^*, \leq_{\ast})^{\{c_1, c_2\}}$

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., \( w_1 = \text{hup.ack.ack} \).

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

Lossy steps: messages may be lost nondeterministically

\[
\sigma \rightarrow \sigma' \iff \sigma \preceq \rho \rightarrow_{\text{rel}} \rho' \preceq \sigma' \text{ for some } \rho, \rho'
\]

where \((S, \preceq)\) is the wqo \((\text{Loc}_1, =) \times (\text{Loc}_2, =) \times (\Sigma^*, \leq_*)^{\{c_1, c_2\}}\)

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

Two-level “differential” encoding:

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

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

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

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

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

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

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

2: $w \preceq_* w'$ implies $\alpha(w) \preceq \alpha(w')$ but not necessarily $\alpha(w) \sqsubseteq \alpha(w')$
ENCODING ORDINALS $< \omega^\omega$ IN CHANNELS

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

Two-level “differential” encoding:

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

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

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

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

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

E.g. $\alpha(III) = \omega^0 + \omega^0 + \omega^0 = 3$, $\alpha(a_1a_0|l|a_1l) = \omega^{\omega\cdot2} + \omega^{\omega+1} \cdot 2$

**Difficulties.**

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

2: $w \leq_* w'$ implies $\alpha(w) \leq \alpha(w')$ but not necessarily $\alpha(w) \subseteq \alpha(w')$
ENCODING ORDINALS $< \omega^\omega$ IN CHANNELS

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

Two-level “differential” encoding:

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

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

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

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

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

E.g. $\alpha(\text{III}) = \omega^0 + \omega^0 + \omega^0 = 3$ \quad $\alpha(a_1a_0|l|a_1l) = \omega^{\omega \cdot 2} + \omega^{\omega^1 \cdot 2}$

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

2: $w \leq^* w'$ implies $\alpha(w) \leq \alpha(w')$ but not necessarily $\alpha(w) \sqsubseteq \alpha(w')$
ENCODING ORDINALS $< \omega^\omega$ IN CHANNELS

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

Two-level “differential” encoding:

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

$\beta(a_{r_1} \ldots a_{r_k}) \triangleq \omega^{r_1} + \ldots + \omega^{r_k}$

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

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

$\alpha(a_1|a_2|\ldots|a_l) \triangleq \omega^{\beta(a_1a_2a_3)} + \ldots + \omega^{\beta(a_1a_2)} + \omega^{\beta(a_1)}$

E.g. $\alpha(III) = \omega^0 + \omega^0 + \omega^0 = 3$ \hspace{1cm} $\alpha(a_1a_0IIa_1I) = \omega^{\omega \cdot 2} + \omega^{\omega + 1} \cdot 2$

Difficulties.

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

2: $w \preceq w'$ implies $\alpha(w) \leq \alpha(w')$ but not necessarily $\alpha(w) \sqsubseteq \alpha(w')$
Weakly computing $\xrightarrow{H}$ with LCS's

\begin{align*}
(lw, n) & \xrightarrow{H} (w, n + 1) \\
(ua_0lw, n) & \xrightarrow{H} (u|^{n+1}a_0w, n) \\
(ua_{r+1}lw, n) & \xrightarrow{H} (ua_{r+1}|a_rw, n) \\
\cdots \text{similar rules for } \xrightarrow{H^{-1}} \cdots
\end{align*}

%$H^{\alpha+1}(n) = H^{\alpha}(n + 1)$

%$H^{\gamma+\omega^{k+1}}(n) = H^{\gamma+\omega^k \cdot (n+1)}(n)$

%$H^{\gamma+\omega^{\beta+\omega^{k+1}}}(n) = H^{\gamma+\omega^\beta+\omega^k \cdot (n+1)}(n)$

Prop. [Robustness]

$w \leq^* w'$ and $n \leq n'$ and $w'$ pure imply $H^{\alpha(w)}(n) \leq H^{\alpha(w')}(n')$

where purity means that $w'$ has no superfluous symbols
(a regular condition that can be enforced by LCS's)
Weakly computing $\xrightarrow{H} \text{ with LCS's}$

\[
(\text{lw}, n) \xrightarrow{H} (\text{w}, n + 1) \\
(\text{ua}_0\text{lw}, n) \xrightarrow{H} (\text{ul}^{n+1}a_0\text{w}, n) \\
(\text{ua}_{r+1}\text{lw}, n) \xrightarrow{H} (\text{ua}_r^{n+1}|a_r\text{w}, n) \\
(\cdots \text{ similar rules for } H^{-1} \cdots)
\]

Prop. [Robustness]

$\text{w} \leq_* \text{w}'$ and $n \leq n'$ and $\text{w}'$ pure imply $H^\alpha(\text{w})(n) \leq H^\alpha(\text{w}')(n')$

where purity means that $\text{w}'$ has no superfluous symbols

(a regular condition that can be enforced by LCS's)
We now store \( u \) and \( l^n \) as two strings (with endmarker \#) on two channels \( p \) and \( d \).

\[
\begin{align*}
\text{p:} & \quad |u#| \\
\text{d:} & \quad |n#|
\end{align*}
\]

\[
\begin{align*}
\text{u#} & \quad \rightarrow \\
\text{|n+1#|}
\end{align*}
\]
Computing with LCS's: Second Rule

\[ p : a_{i1} ... a_{ip} a_0 | u# \]
\[ d : \ |^{n\#} \]
\[ \rightarrow \]
\[ a_{i1} ... a_{ip} |^{n+1} a_0 u# \]
\[ \ |^{n\#} \]
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 $F_{\omega \omega}$, hence $F_{\omega \omega}$-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 $F_{\omega \omega}$, hence $F_{\omega \omega}$-complete

**Confirms:** the main parameter for complexity is the size of the message alphabet
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 $\mathcal{F}_{\omega \omega}$, hence $\mathcal{F}_{\omega \omega}$-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 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 $F_{\omega \omega \omega}$
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 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 $\mathbb{F}_{\omega^\omega}$.
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 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 $\mathcal{F}_{\omega^\omega}$.
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 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 $\mathcal{F}_{\omega^\omega}$