Cryptocomplexity I

Kryptokomplexität I

Wintersemester 2023/2024

Chapter 7: Speed-Up, Compression, and Hierarchy Theorems

Dozent: Prof. Dr. J. Rothe

Linear Speed-Up

© autowp

Linear Speed-Up and Compression Theorems

Linear Speed-Up and Compression Theorems

The central question in this section is:

How much must a resource be increased in order to be able to compute strictly more?

Consider, for example, the deterministic time class $\mathrm{DTIME}(t_1)$, for some resource function t_1. How much stronger than t_1 must another function, t_2, grow in order to ensure that

$$\mathrm{DTIME}(t_1) \neq \mathrm{DTIME}(t_2)?$$

The linear tape-compression and speed-up theorems say that a linear increase of the given resource function does *not* suffice to get a strictly bigger complexity class.

Linear Compression Theorem

Theorem (Linear Compression)

For each total recursive function s,

$$\mathrm{DSPACE}(s) = \mathrm{DSPACE}(\mathbb{L}\mathrm{in}(s)).$$

Proof: It is enough to show that $\mathrm{DSPACE}(2s) \subseteq \mathrm{DSPACE}(s)$.

Let M be a DTM working, on any input x of length n, in space $2s(n)$.

It is convenient to make, without loss of generality, the following assumptions about M:

(a) M has only one tape that
(b) is infinite in just one direction,
(c) the tape cells are enumerated by 1, 2, etc., and
(d) M's head makes a left turn only on even-numbered cells.

Linear Compression Theorem

(If the given machine does not have these properties, it is not difficult to replace it by an equivalent one that does have the desired properties.)

Suppose further that Γ is the working alphabet of M.

The goal is to construct a new DTM N that, on input x of length n, simulates the computation of $M(x)$ but works in space $s(n)$.

Idea: N, which has more states than M and whose working alphabet is $\Gamma \times \Gamma$, "delays" the simulation of M: it will wait and see what M is going to do next before actually doing it.

To this end, view M's tape as being subdivided into blocks of two adjacent cells each, i.e., the blocks are the pairs of cells with numbers $(2i-1, 2i)$, for $i \geq 1$.

Linear Compression Theorem

Each such block is now considered to be *one* tape cell of N, and every ordered pair of symbols $(a, b) \in \Gamma \times \Gamma$ is now considered to be *one* symbol of N.

Then, $N(x)$ simulates the computation of $M(x)$, except that N moves its head to the right only when M's head crosses a boundary between two blocks to the right.

All steps of M within any one block can be simulated by N's finite control.

That is why N needs more states than M.

Clearly, $N(x)$ performs the exact same computation as $M(x)$ and needs only space $s(n)$. ❑

Linear Speed-Up Theorem

Theorem (Linear Speed-Up)

For each total recursive function t with $t \succ \mathrm{id}$,

$$\mathrm{DTIME}(t) = \mathrm{DTIME}(\mathbb{L}\mathrm{in}(t)).$$

Proof of Linear Speed-Up Theorem: Idea

Proof:

- Let $A \in \mathrm{DTIME}(t)$, and let M be a DTM such that $L(M) = A$ and M works in time $t(n)$ on inputs of length n.
- **Goal:** Construct a DTM N with $L(N) = A$ but at least m times as fast as M, for some constant $m > 1$.
  That is, m steps of M are to be simulated within just one step of N.
- Again, the idea is that patience will pay off:
  - N will "delay" the simulation of M, i.e., N will wait and see what M is going to do within the next m steps, then doing it all at once within a single step of its own.
  - N will compress the input using a larger alphabet and more states.

Proof of Linear Speed-Up Theorem: Idea

- However, N can use its compressed encoding not before it has scanned every input bit and has transformed the input into the compressed encoding to be used later on.

- In other words, the head moves on the input tape cannot be speeded up.

  Thus, the computation of N, on input x of length n, is done in two phases:

  - **Preparation Phase**;
  - **Simulation Phase**.

Proof of Linear Speed-Up Theorem: Preparation

- Subdivide the input string $x = x_1 x_2 \cdots x_n$ of length n into blocks of length m, where the ith block, $i \geq 1$, is represented by the string

$$\beta_i = x_{1+(i-1)m} x_{2+(i-1)m} \cdots x_{i \cdot m}.$$

- Then, N writes on its working tape the following redundant encoding of the input string:

$$(\Box^m, \beta_1, \beta_2)\,(\beta_1, \beta_2, \beta_3)\,(\beta_2, \beta_3, \beta_4)\,\cdots\,(\beta_{k-2}, \beta_{k-1}, \beta_k)\,(\beta_{k-1}, \beta_k, \Box^m)$$

- Every triple of the form $(\beta_{i-1}, \beta_i, \beta_{i+1})$, where $1 < i < k$, or $(\Box^m, \beta_1, \beta_2)$ or $(\beta_{k-1}, \beta_k, \Box^m)$ is considered to be just *one* symbol of N.

Proof of Linear Speed-Up Theorem: Preparation

- After N
  - has copied the input in this compressed (and somewhat redundant) form onto the working tape and
  - has moved the head back to the leftmost symbol, $(\Box^m, \beta_1, \beta_2)$,

  this working tape will henceforth be used as the input tape.
- The original input tape, which has been erased during the preparation phase, will henceforth be used as a working tape.
- The preparation phase requires

$$n + k = \left(1 + \frac{1}{m}\right) n$$

steps.

Proof of Linear Speed-Up Theorem: Simulation

- As above, N's encoding of $a = a_1 a_2 \cdots a_\ell$ is of the form

$$(\square^m, \alpha_1, \alpha_2)\,(\alpha_1, \alpha_2, \alpha_3)\,(\alpha_2, \alpha_3, \alpha_4)\,\cdots\,(\alpha_{z-2}, \alpha_{z-1}, \alpha_z)\,(\alpha_{z-1}, \alpha_z, \square^m)$$

where

(1) a is subdivided into $z+1$ blocks,

$$a = \alpha_1 \alpha_2 \cdots \alpha_{z+1},$$

(2) for each i with $1 \leq i \leq z$, block

$$\alpha_i = a_{1+(i-1)m} a_{2+(i-1)m} \cdots a_{i \cdot m}$$

has length m, and

(3) block α_{z+1} with $|\alpha_{z+1}| < m$ is handled by N's finite control.

Proof of Linear Speed-Up Theorem: Simulation

$N(x)$ simulates m steps of $M(x)$ as follows:

- If M's head is currently scanning α_j, then N's head scans $(\alpha_{j-1}, \alpha_j, \alpha_{j+1})$.
- After m steps, M's head has moved by at most m tape cells. Hence, it must scan either α_{j-1} or α_j or α_{j+1}, and none of the other blocks has been changed by M.
- Since N's head scans $(\alpha_{j-1}, \alpha_j, \alpha_{j+1})$, it can do all of M's changes within a single step of its own, and it moves its head to the symbol:

  $(\alpha_{j-2}, \alpha_{j-1}, \alpha_j)$ if M scans α_{j-1}

  $(\alpha_{j-1}, \alpha_j, \alpha_{j+1})$ if M scans α_j

  $(\alpha_j, \alpha_{j+1}, \alpha_{j+2})$ if M scans α_{j+1}

  after these m steps.

Proof of Linear Speed-Up Theorem: Simulation

- If M accepts or rejects x within these m steps, then so does N. Hence, $L(N) = L(M)$.
- The simulation phase requires at most

$$\left\lceil \frac{t(n)}{m} \right\rceil$$

steps.

Proof of Linear Speed-Up Theorem: Analysis

- Recall that $\mathrm{id} \prec t$, i.e., $n \in o(t(n))$. Thus,

$$(\forall c > 0)\,[n <_{\mathrm{ae}} c \cdot t(n)]. \tag{1}$$

- Summing up the time spent in both phases, $N(x)$ needs no more than

$$\begin{aligned}\left(1+\frac{1}{m}\right)n + \left\lceil \frac{t(n)}{m} \right\rceil &<_{\mathrm{ae}} \left(1+\frac{1}{m}\right)\frac{1}{m\left(1+\frac{1}{m}\right)}t(n) + \left\lceil \frac{t(n)}{m} \right\rceil \\ &\leq \left\lceil \frac{2t(n)}{m} \right\rceil + 1\end{aligned}$$

steps, where the first inequality follows from (1) for the specific constant

$$\hat{c} = \frac{1}{m\left(1+\frac{1}{m}\right)} = \frac{1}{m+1}.$$

Proof of Linear Speed-Up Theorem: Analysis

- The finitely many exceptions allowed in the $\leq_{\mathrm{ae}}$-notation can be handled by table-lookup.
- Thus, an arbitrary linear speed-up is possible by suitably choosing m. ❑

Proof of Linear Speed-Up Theorem: Illustration

- Suppose $t(n) = d \cdot n$, for some constant $d > 1$, and $N(x)$ has running time

$$\begin{aligned} T(n) &= \left(1 + \frac{1}{m}\right) n + \frac{t(n)}{m} \\ &= \left(1 + \frac{1}{m}\right) n + \frac{d \cdot n}{m} \\ &= \left(1 + \frac{d+1}{m}\right) n, \end{aligned}$$

where we assume for convenience that m divides both n and $t(n)$.

Proof of Linear Speed-Up Theorem: Illustration

- Since $d > 1$, choosing
$$m > \frac{d+1}{d-1}$$
implies $T(n) < d \cdot n = t(n)$ and thus a genuine speed-up.

- This example also suggests that the above proof does not work for $d = 1$, i.e., it does not work for $t = \mathrm{id}$.

Existence of Arbitrarily Complex Problems

Fact

For each $t \in \mathbb{R}$, there exists a problem A_t such that $A_t \notin \mathrm{DTIME}(t)$.

Proof: The proof is by diagonalization. Let $M_0, M_1, M_2, \ldots$ be a Gödelization (i.e., an effective enumeration) of all DTMs. Define

$$A_t = \{0^i \mid M_i \text{ does not accept } 0^i \text{ within } t(i) \text{ steps}\}.$$

Suppose $A_t \in \mathrm{DTIME}(t)$. Then, there exists a j such that $L(M_j) = A_t$ and $\mathrm{time}_{M_j}(n) \leq t(n)$ for each $n \in \mathbb{N}$. Hence,

$$\begin{aligned} 0^j \in A_t \quad &\Longleftrightarrow \quad M_j \text{ does not accept } 0^j \text{ within } t(j) \text{ steps} \\ &\Longleftrightarrow \quad 0^j \notin L(M_j) = A_t, \end{aligned}$$

which is a contradiction. It follows that $A_t \notin \mathrm{DTIME}(t)$. ❑

Existence of Arbitrarily Complex Problems

Since complexity classes such as $\mathrm{DTIME}(t)$ are closed under finite invariance, "$A_t \in \mathrm{DTIME}(t)$" means:

"For some DTM M, $L(M) = A_t$ and $\mathrm{time}_M(n) \leq_{\mathrm{ae}} t(n)$."

Hence, "$A_t \notin \mathrm{DTIME}(t)$" in the above fact means:

"For each DTM M with $L(M) = A_t$, $\mathrm{time}_M(n) >_{\mathrm{io}} t(n)$."

However, "$A_t \notin \mathrm{DTIME}(t)$" does not exclude that, for infinitely many other $n \in \mathbb{N}$, $\mathrm{time}_M(n) \leq_{\mathrm{io}} t(n)$ may nonetheless be true.

In this sense, Rabin's theorem on the next slide is much stronger than the above fact.

The (omitted) proof uses a clever priority argument in its diagonalization.

Existence of Arbitrarily Complex Problems

Theorem (Rabin)

For each $t \in \mathbb{R}$, there exists a decidable set D_t such that for each DTM M deciding D_t, it holds that

$$\mathrm{time}_M(n) >_{\mathrm{ae}} t(n).$$

without proof

Space-Constructibility

Definition (Space-Constructibility)

Let s be a function in $\mathbb{R}$ mapping from $\mathbb{N}$ to $\mathbb{N}$.

We say that s is *space-constructible* if and only if there exists a DTM M such that, for each n,

- M on any input of length n uses no more than $s(n)$ tape cells and
- prints the string

$$\#1^{s(n)-2}\$$$

  on one of its tapes, where $\#$ and $\$$ are special symbols marking the left and right boundaries.

We then say that M has *marked the space $s(n)$*.

Time-Constructibility

Definition (Time-Constructibility)

Let f and t be functions in $\mathbb{R}$ mapping from $\mathbb{N}$ to $\mathbb{N}$.

We say that f is *constructible in time t* if and only if there exists a DTM M such that, for each n,

- M on any input of length n runs for exactly $t(n)$ steps and
- prints the string

$$\#1^{f(n)-2}\$$$

  on its tape, where $\#$ and $\$$ are special symbols marking the left and right boundaries.

We say that t is *time-constructible* if and only if t is constructible in time t.

Space- and Time-Constructibility

Remark: Constructibility of resource functions is necessary to obtain an effective enumeration of the Turing machines representing a complexity class. For example, the set

$$\{M \mid L(M) \in \mathrm{DSPACE}(s)\}$$

- is decidable if s is space-constructible;
- otherwise it is not even recursively enumerable.

Example: $\log n, \quad n^k, \quad 2^n, \quad 2^{n^k}, \quad \ldots$ are space- and (except $\log n$, which doesn't make sense for DTMs) time-constructible.

Effective Enumerations of Turing Machines

Goal: We want to effectively enumerate all DTMs or NTMs that always work within a given time bound t (or space bound s).

For example, for $\mathrm{DTIME}(t)$:

- Let t be time-constructible via DTM M.
- Let $M_1, M_2, \ldots$ be a fixed Gödelization of all DTMs.
- Construct an enumeration $M'_1, M'_2, \ldots$ for $\mathrm{DTIME}(t)$ as follows.
  M'_i on input x:
  - simulates $M_i(x)$ and $M(1^{|x|})$ in parallel;
  - if $M_i(x)$ stops first or at the same time as $M(1^{|x|})$ then:

$$M'_i \text{ accepts } x \iff M_i \text{ accepts } x.$$

  - if $M(1^{|x|})$ stops first then M'_i rejects x.

Space Hierarchy Theorem

Again:

How much must a resource be increased in order to be able to compute strictly more?

Consider, for example, the deterministic space class $\mathrm{DSPACE}(s_1)$, for some resource function s_1. How much stronger than s_1 must another function, s_2, grow in order to ensure that

$$\mathrm{DSPACE}(s_1) \neq \mathrm{DSPACE}(s_2)?$$

Space Hierarchy Theorem

From the linear tape-compression theorem we know that

$$s_2 \in \mathcal{O}(s_1) \iff \exists c > 0 : s_2 \leq_{\text{ae}} c \cdot s_1$$

is not enough.

However, the negation:

$$s_2 \succ_{\text{io}} s_1 \iff \forall c > 0 : s_2 >_{\text{io}} c \cdot s_1$$

does suffice.

Theorem (Space Hierarchy Theorem)

If $s_1 \prec_{\text{io}} s_2$ and s_2 is space-constructible, then

$$\text{DSPACE}(s_2) \not\subseteq \text{DSPACE}(s_1).$$

Space Hierarchy Theorem: Proof

Proof: We prove the theorem only for the case of $s_1 \geq \log$.
(Using a result of Sipser (TCS 1980), one can get rid of this simplifying assumption.)

To construct a set A in the difference

$$\mathrm{DSPACE}(s_2) - \mathrm{DSPACE}(s_1)$$

by diagonalization, fix a Gödelization $M_0, M_1, M_2, \ldots$ of all DTMs having one working tape. (It is easy to see that it is enough to consider, without loss of generality, only one-tape DTMs.)

Define a DTM N with an input tape and three working tapes.

Space Hierarchy Theorem: Proof

On input $x \in \{0,1\}^*$ of length n, DTM N works as follows:

1. N marks the space $s_2(n)$ on all three working tapes.
2. Suppose x is of the form $x = 1^i y$, $0 \leq i \leq n$, $y \in \{\varepsilon\} \cup 0\{0,1\}^*$.

   That is, x starts with a (possibly empty) prefix of i ones
   - followed either by the empty string ε (in which case $x = 1^n$), or
   - followed by a zero and a (possibly empty) string from $\{0,1\}^*$.

   DTM N interprets i as a machine number, and it writes the suitably encoded program of M_i onto its first working tape.

   If this is not possible, since M_i's program is too large to fit in the marked space $s_2(n)$, then N aborts the computation and rejects x.

Space Hierarchy Theorem: Proof

Otherwise, N proceeds by

- simulating the computation of $M_i(x)$ on the second working tape,
- using the program of M_i on its first working tape and
- reading the symbols of x from its own input tape.

3. The third working tape contains a binary counter that is initially set to zero and is incremented by one in each step of the simulation of $M_i(x)$.

   If the simulation of $M_i(x)$ succeeds on N's second working tape before the counter on N's third working tape overflows, then $N(x)$ accepts if and only if $M_i(x)$ rejects.

   Otherwise, N rejects x.

Space Hierarchy Theorem: Proof

Some technical explanations are in order:

- The counter on N's third working tape guarantees that $N(x)$ halts, even if $M_i(x)$ would never terminate.
- There exists a constant c_i such that the simulation of $M_i(x)$ on N's second working tape can be done in space at most

$$c_i \cdot \mathrm{space}_{M_i}(n).$$

Why?

DTM N must be able to simulate *every* DTM M_i, $i \in \mathbb{N}$.

If for some i, M_i has z_i states and ℓ_i symbols in its working alphabet, then N can encode these states and symbols in binary, i.e., by strings over $\{0, 1\}$ of length $\lceil \log z_i \rceil$ and $\lceil \log \ell_i \rceil$.

Space Hierarchy Theorem: Proof

This encoding causes a constant space overhead for the simulating machine N, where the constant c_i depends only on M_i.

Define $A = L(N)$. Clearly, $A \in \mathrm{DSPACE}(s_2)$.

To prove that $A \notin \mathrm{DSPACE}(s_1)$, suppose for a contradiction that $A \in \mathrm{DSPACE}(s_1)$.

Thus, there exists some i such that $A = L(M_i)$ and

$$\mathrm{space}_{M_i}(n) \leq s_1(n) \prec_{\mathrm{io}} s_2(n).$$

Recall what $s_1 \prec_{\mathrm{io}} s_2$ means:

$$(\forall c > 0)\,[s_2(n) >_{\mathrm{io}} c \cdot s_1(n)]. \tag{2}$$

Space Hierarchy Theorem: Proof

Hence, for each real constant $c > 0$, there exist infinitely many arguments n_0, n_1, n_2, ... in $\mathbb{N}$ such that

$$s_2(n_k) > c \cdot s_1(n_k) \quad \text{for each } k.$$

From this infinite sequence of arguments, choose n_j large enough such that the following three conditions hold:

(a) M_i's program can be computed and written onto N's second working tape in space $s_2(n_j)$;

(b) the simulation of $M_i(1^i 0^{n_j - i})$ succeeds in space $s_2(n_j)$;

(c) $\mathrm{time}_{M_i}(n_j) \leq 2^{s_2(n_j)}$.

Space Hierarchy Theorem: Proof

Condition (a) can be satisfied for a large enough n_j, since the size of the program of M_i is a constant not depending on the machine's input.

Condition (b) can be satisfied for a large enough n_j, since the simulation of $M_i(1^i 0^{n_j - i})$ succeeds in space:

$$c_i \cdot \mathrm{space}_{M_i}(n_j) \leq c_i \cdot s_1(n_j) < s_2(n_j),$$

where c_i is the above constant that is due to N having to encode M_i's states and symbols, and where the last inequality follows from (2).

Condition (c) can be satisfied for a large enough n_j, since for $s_1 \geq \log$:

$$\begin{aligned} \mathrm{time}_{M_i}(n_j) &\leq 2^{d \cdot \mathrm{space}_{M_i}(n_j)} && \text{for a suitable constant } d \\ &\leq 2^{d \cdot s_1(n_j)} \\ &< 2^{s_2(n_j)}, && \text{again by (2).} \end{aligned}$$

Space Hierarchy Theorem: Proof

Hence, the simulation of $M_i(1^i 0^{n_j - i})$ succeeds before the binary counter of length $s_2(n_j)$ on N's third working tape is full.

Conditions (a), (b), and (c) and the construction of N imply that for the string $x = 1^i 0^{n_j - i}$,

$$\begin{aligned} x \in A \quad &\Longleftrightarrow \quad N \text{ accepts } x \\ &\Longleftrightarrow \quad M_i \text{ rejects } x. \end{aligned}$$

Thus, $A \neq L(M_i)$, contradicting our supposition.

Hence, $A \notin \mathrm{DSPACE}(s_1)$. ❑

Time Hierarchy Theorem

Theorem (Time Hierarchy Theorem)

If $t_2 \geq \mathrm{id}$ *and* $t_1 \prec_{\mathrm{io}} t_2$ *and* t_2 *is constructible in time* $t_2 \log t_2$*, then*

$$\mathrm{DTIME}(t_2 \log t_2) \not\subseteq \mathrm{DTIME}(t_1).$$

without proof

Upper Bounds and Lower Bounds

- **Upper bound** for a problem Π: *There exists some algorithm* (of the specified type) that solves Π within the given complexity bound.
- **Lower bound** for a problem Π: *All algorithms* (of the specified type) solving Π require at least / more than the given complexity.

A Lower Bound Proof via Crossing Sequences

Theorem

For each DTM M with only one working tape and no separate input tape that decides the problem

$$S = \{x2^{|x|}x \mid x \in \{0,1\}^*\}$$

there exists a constant $c > 0$ such that

$$\mathrm{time}_M(n) > c \cdot n^2.$$

A Lower Bound Proof via Crossing Sequences

Proof: Let M be a DTM with only one working tape and no separate input such that

$$L(M) = S = \{x2^{|x|}x \mid x \in \{0,1\}^*\}.$$

Let $w = uv$ be any input string.

A sequence of states of $M(w)$, denoted by

$$\mathrm{cs}(u|v) = (s_1, s_2, \ldots, s_k),$$

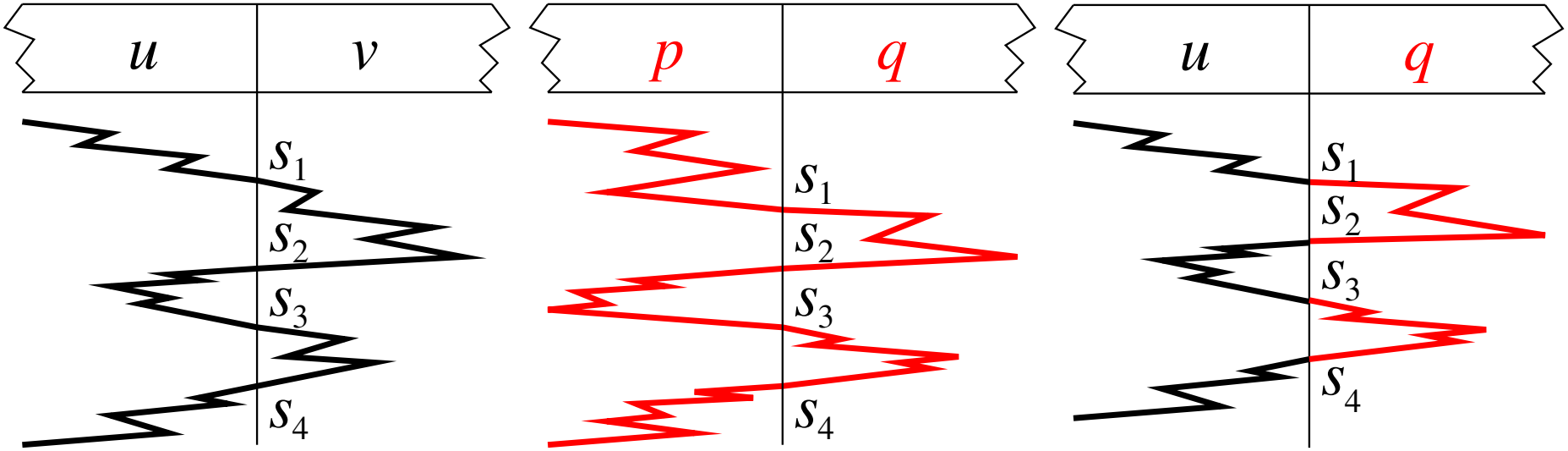
is called the *crossing sequence of $M(x)$ at the cell-boundary between u and v* if M's head crosses this cell-boundary exactly k times during the computation of $M(x)$ and M is in state s_i during the ith crossing.

A Lower Bound Proof via Crossing Sequences

Lemma

If $uv \in L(M)$ *and* $pq \in L(M)$ *and* $\mathrm{cs}(u|v) = \mathrm{cs}(p|q)$, *then* $uq \in L(M)$ *and* $pv \in L(M)$.

Proof Sketch of Lemma.



❑

A Lower Bound Proof via Crossing Sequences

By this lemma, for strings x and y with $x \neq y$, all crossing sequences of $x2^{|x|}x$ and $y2^{|y|}y$ in the block of 2s are pairwise distinct.

Why?

Because otherwise, M would accept strings *not in* S.

For example, consider

$$w_1 = uv = 101222101 \quad \text{with} \quad u = 1012 \quad \text{and} \quad v = 22101,$$
$$w_2 = pq = 001222001 \quad \text{with} \quad p = 0012 \quad \text{and} \quad q = 22001.$$

Clearly, $x = 101 \neq 001 = y$ and $w_1, w_2 \in S = L(M)$.

By the lemma, if $\mathrm{cs}(u|v) = \mathrm{cs}(p|q)$, then $uq = 101222001 \in L(M) = S$ and $pv = 001222101 \in L(M) = S$, contradicting the definition of S.

A Lower Bound Proof via Crossing Sequences

Let $z > 1$ be the number of M's states.

Then the number of distinct crossing sequences of length at most ℓ is:

$$z^0 + z^1 + \cdots + z^\ell = \frac{z^{\ell+1} - 1}{z - 1} < z^{\ell+1}.$$

Consider strings $x2^{|x|}x$ in S with $x \in \{0,1\}^*$ and $|x| = n$.
There are exactly 2^n such strings of length $3n$ in S.

We say a crossing sequence is *short with respect to n* if its length is shorter than ℓ_0, where

$$z^{\ell_0+1} = 2^n, \quad \text{i.e.,} \quad \ell_0 = \frac{n}{\log z} - 1.$$

A Lower Bound Proof via Crossing Sequences

Thus there are fewer short crossing sequences with respect to n than strings in $S \cap \{0,1,2\}^{3n}$.

Since for distinct strings in $S \cap \{0,1,2\}^{3n}$ all crossing sequences in the block of 2s are pairwise distinct, there exists a string $w \in S$, $|w| = 3n$, that has no short crossing sequences with respect to n in the block of 2s, i.e., all its crossing sequences in this block are of length at least ℓ_0.

It follows that M needs time at least

$$(n-1)\left(\frac{n}{\log z} - 1\right). \quad \square$$