Deterministic complexity classes

Complexity Theory

Classify computable functions
be amount of resources needed
for their computation!

Resources:

- time
- space
- nondeterminism

Complexity Theory

Classify computable functions
be amount of resources needed
for their computation!

Resources:

- time
- space
- nondeterminism

Complexity class C_R:

- set of languages
- decidable with resource bound R(n)

Complexity Theory

Classify computable functions
be amount of resources needed
for their computation!

Resources:

- time
- space
- nondeterminism

Complexity class C_R:

- set of languages
- decidable with resource bound R(n)

the most famous classes:

- P: languages decidable in polynomial time
  - $O(n^k)$ for some k
- NP: languages decidable in nondeterministic polynomial time

Complexity Theory

Classify computable functions
be amount of resources needed
for their computation!

Resources:

- time
- space
- nondeterminism

Complexity class C_R:

- set of languages
- decidable with resource bound R(n)

the most famous classes:

- P: languages decidable in polynomial time
  - $O(n^k)$ for some k
- NP: languages decidable in nondeterministic polynomial time

since 1971: the most famous open problem of

- CS
- math

$\boxed{\textbf{P = NP ?}}$

1 Defining complexitiy classes

1.1 reminders:

$L \subseteq A^*$ language, M Turing machine, resource bounds

$$s, t : \mathbb{N}_0 \rightarrow \mathbb{R}^+$$

$s(n)$ and $t(n)$ will measure space and time as a function of the input length n. We are here only interested in machines, which read the entire input. Hence we will assume

$$s(n) \geq n \quad , \quad t(n) \geq n$$

1 Defining complexitiy classes

1.1 reminders:

$L \subseteq A^*$ language, M Turing machine, resource bounds

$$s, t : \mathbb{N}_0 \to \mathbb{R}^+$$

$s(n)$ and $t(n)$ will measure space and time as a function of the input length n. We are here only interested in machines, which read the entire input. Hence we will assume

$$s(n) \geq n \quad , \quad t(n) \geq n$$

- M accepts L iff
  $\forall w \in A^*$: M started with w halts in an accepting state $z \in Z_A$ iff $w \in L$

rejection possible by

- not halting
- halting in a non accepting (rejecting) state

1 Defining complexitiy classes

1.1 reminders:

$L \subseteq A^*$ language, M Turing machine, resource bounds

$$s, t : \mathbb{N}_0 \to \mathbb{R}^+$$

$s(n)$ and $t(n)$ will measure space and time as a function of the input length n. We are here only interested in machines, which read the entire input. Hence we will assume

$$s(n) \geq n \quad , \quad t(n) \geq n$$

- M accepts L iff
  $\forall w \in A^*$: M started with w halts in an accepting state $z \in Z_A$ iff $w \in L$

rejection possible by

- not halting
- halting in a non accepting (rejecting) state

- M decides L iff M computes χ_L, i.e.
  $\forall w \in A^*$: M started with w outputs

$$\chi_L(w) = \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$

and halts.

1 Defining complexitiy classes

1.1 reminders:

$L \subseteq A^*$ language, M Turing machine, resource bounds

$$s,t : \mathbb{N}_0 \to \mathbb{R}^+$$

$s(n)$ and $t(n)$ will measure space and time as a function of the input length n. We are here only interested in machines, which read the entire input. Hence we will assume

$$s(n) \geq n \quad , \quad t(n) \geq n$$

- M accepts L iff
  $\forall w \in A^*$: M started with w halts in an accepting state $z \in Z_A$ iff $w \in L$

rejection possible by

- not halting
- halting in a non accepting (rejecting) state

- M decides L iff M computes χ_L, i.e.
  $\forall w \in A^*$: M started with w outputs

$$\chi_L(w) = \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$

and halts.

- M is $t(n)$ time bounded if for all $w \in A^*$ holds:
  if $|w| = n$, then machine M started with w halts after at most $t(n)$ steps.
- M is $s(n)$ space bounded if for all $w \in A^*$ holds:
  if $|w| = n$, then the number of tape cells visited by the heads or occupied by the input is at most $s(n)$

attention: space bounded machines may not halt

1 Defining complexitiy classes

1.1 reminders:

$L \subseteq A^*$ language, M Turing machine, resource bounds

$$s,t : \mathbb{N}_0 \to \mathbb{R}^+$$

$s(n)$ and $t(n)$ will measure space and time as a function of the input length n. We are here only interested in machines, which read the entire input. Hence we will assume

$$s(n) \geq n \quad , \quad t(n) \geq n$$

- M accepts L iff
  $\forall w \in A^*$: M started with w halts in an accepting state $z \in Z_A$ iff $w \in L$

rejection possible by

- not halting
- halting in a non accepting (rejecting) state

- M decides L iff M computes χ_L, i.e.
  $\forall w \in A^*$: M started with w outputs

$$\chi_L(w) = \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$

and halts.

- M is $t(n)$ time bounded if for all $w \in A^*$ holds:
  if $|w| = n$, then machine M started with w halts after at most $t(n)$ steps.
- M is $s(n)$ space bounded if for all $w \in A^*$ holds:
  if $|w| = n$, then the number of tape cells visited by the heads or occupied by the input is at most $s(n)$

attention: space bounded machines may not halt

- complement of a language

$$\overline{L} = A^* \setminus L$$

1 Defining complexitiy classes

1.2 Deciding by end state

def: deciding by end state: an alternative definition for deciding

$$M = (Z, \Sigma, \delta, z_0, Z_A, Z_R)$$

- $Z_A \subseteq Z$: set of accepting end states
- $Z_R \subseteq Z$: set of rejecting end states

$$Z_A \cap Z_R = \emptyset$$

- M decides L by state if for all $w \in A^*$ machine M started with w halts in an accepting state if $w \in L$ and in a rejecting stete if $w \notin L$

1 Defining complexitiy classes

1.2 Deciding by end state

def: deciding by end state: an alternative definition for deciding

$$M = (Z, \Sigma, \delta, z_0, Z_A, Z_R)$$

- $Z_A \subseteq Z$: set of accepting end states
- $Z_R \subseteq Z$: set of rejecting end states

$$Z_A \cap Z_R = \emptyset$$

- M decides L by state if for all $w \in A^*$ machine M started with w halts in an accepting state if $w \in L$ and in a rejecting stete if $w \notin L$

Lemma 1. *If L is decided by a $t(n)$-time bounded k-tape TM, them L is decided by state by a $t(n) + O(1)$-time bounded k-tape TM.*

Proof. very easy exercise □

Lemma 2. *If L is decided by state by a $t(n)$-time bounded k-tape TM, them L is decided by a $t(n) + O(1)$-time bounded k-tape TM.*

O(t(n))

Proof. very easy exercise □

from now on: deciding by end state .

1 Defining complexitiy classes

1.3 Complexity classes

time completity classes defined as sets of languages

$TIME_k(t(n)) = \{L : L \text{ accepted by a deterministic } O(t(n)) - \text{time bounded } k \text{ tape } TM\}$

$$TIME(t(n)) = \bigcup_k TIME_k(t(n))$$

1.3 Complexity classes

time completity classes defined as sets of languages

$TIME_k(t(n)) = \{L : L \text{ accepted by a deterministic } O(t(n)) - \text{time bounded } k \text{ tape } TM\}$

$$TIME(t(n)) = \bigcup_k TIME_k(t(n))$$

Lemma 3. *Accepting and deciding by time bounded machines is the same, i.e.* $L \in TIME(t(n))$ *iff there is an* $O(t(n))$*-time bounded TM which decides L (by state).*

- by lemmas 1 and 2.
- $t(n)$ time bounded machines always halt

1.3 Complexity classes

time completity classes defined as sets of languages

$TIME_k(t(n)) = \{L : L \text{ accepted by a deterministic } O(t(n)) - \text{time bounded } k \text{ tape } TM\}$

$$TIME(t(n)) = \bigcup_k TIME_k(t(n))$$

Lemma 3. *Accepting and deciding by time bounded machines is the same, i.e. $L \in TIME(t(n))$ iff there is an $O(t(n))$-time bounded TM which decides L (by state).*

- by lemmas 1 and 2.
- $t(n)$ time bounded machines always halt

example : Let $x \notin \mathbb{B}$

$$L = \{wx^n w \ : w \in \mathbb{B}^n \ , \ n \in \mathbb{N}_0\}$$

then

$$L \in TIME_2(n) \quad L \in TIME_1(n^2)$$

proof: exercise

1 Defining complexitiy classes

1.3 Complexity classes

time completity classes defined as sets of languages

$TIME_k(t(n)) = \{L : L \text{ accepted by a deterministic } O(t(n)) - \text{time bounded } k \text{ tape } TM\}$

$$TIME(t(n)) = \bigcup_k TIME_k(t(n))$$

Lemma 3. *Accepting and deciding by time bounded machines is the same, i.e. $L \in TIME(t(n))$ iff there is an $O(t(n))$-time bounded TM which decides L (by state).*

- by lemmas 1 and 2.
- $t(n)$ time bounded machines always halt

example : Let $x \notin \mathbb{B}$

$$L = \{wx^n w \, : w \in \mathbb{B}^n \, , \, n \in \mathbb{N}_0\}$$

then

$$L \in TIME_2(n) \quad L \in TIME_1(n^2)$$

proof: exercise

Lemma 4.

$$\boxed{TIME(t(n)) \subseteq TIME_1(t^2(n))}$$

Proof. Shown in tape reduction theorem. □

1 Defining complexitiy classes

1.3 Complexity classes

time completity classes defined as sets of languages

$TIME_k(t(n)) = \{L : L \text{ accepted by a deterministic } O(t(n)) - \text{time bounded } k \text{ tape } TM\}$

$$TIME(t(n)) = \bigcup_k TIME_k(t(n))$$

Lemma 3. *Accepting and deciding by time bounded machines is the same, i.e. $L \in TIME(t(n))$ iff there is an $O(t(n))$-time bounded TM which decides L (by state).*

- by lemmas 1 and 2.
- $t(n)$ time bounded machines always halt

example : Let $x \notin \mathbb{B}$

$$L = \{wx^n w : w \in \mathbb{B}^n \ , \ n \in \mathbb{N}_0\}$$

then

$$L \in TIME_2(n) \quad L \in TIME_1(n^2)$$

proof: exercise

Lemma 4.

$$\boxed{TIME(t(n)) \subseteq TIME_1(t^2(n))}$$

Proof. Shown in tape reduction theorem. □

space completity classes defined as sets of languages

$SPACE_k(s(n)) = \{L : L \text{ accepted by a deterministic } O(s(n)) - \text{space bounded } k \text{ tape } TM\}$

$$SPACE(t(n)) = \bigcup_k SPACE_k(s(n))$$

1 Defining complexitiy classes

1.3 Complexity classes

time completity classes defined as sets of languages

$TIME_k(t(n)) = \{L : L \text{ accepted by a deterministic } O(t(n)) - \text{time bounded } k \text{ tape } TM\}$

$$TIME(t(n)) = \bigcup_k TIME_k(t(n))$$

Lemma 3. *Accepting and deciding by time bounded machines is the same, i.e. $L \in TIME(t(n))$ iff there is an $O(t(n))$-time bounded TM which decides L (by state).*

- by lemmas 1 and 2.
- $t(n)$ time bounded machines always halt

example : Let $x \notin \mathbb{B}$

$$L = \{wx^n w \ : w \in \mathbb{B}^n \ , \ n \in \mathbb{N}_0\}$$

then

$$L \in TIME_2(n) \quad L \in TIME_1(n^2)$$

proof: exercise

Lemma 4.

$$\boxed{TIME(t(n)) \subseteq TIME_1(t^2(n))}$$

Proof. Shown in tape reduction theorem. □

space completity classes defined as sets of languages

$SPACE_k(s(n)) = \{L : L \text{ accepted by a deterministic } O(s(n)) - \text{space bounded } k \text{ tape } TM\}$

$$SPACE(t(n)) = \bigcup_k SPACE_k(s(n))$$

if you care for $s(n) < n$: keep input on read only *input tape*; count with $s(n)$ only tape cells visited on other so called *work tapes*.

1 Defining complexitiy classes

1.3 Complexity classes

time completity classes defined as sets of languages

$TIME_k(t(n)) = \{L : L \text{ accepted by a deterministic } O(t(n)) - \text{time bounded } k \text{ tape } TM\}$

$$TIME(t(n)) = \bigcup_k TIME_k(t(n))$$

Lemma 3. *Accepting and deciding by time bounded machines is the same, i.e. $L \in TIME(t(n))$ iff there is an $O(t(n))$-time bounded TM which decides L (by state).*

- by lemmas 1 and 2.
- $t(n)$ time bounded machines always halt

example : Let $x \notin \mathbb{B}$

$$L = \{wx^n w : w \in \mathbb{B}^n , n \in \mathbb{N}_0\}$$

then

$$L \in TIME_2(n) \quad L \in TIME_1(n^2)$$

proof: exercise

Lemma 4.

$$\boxed{TIME(t(n)) \subseteq TIME_1(t^2(n))}$$

Proof. Shown in tape reduction theorem. □

space completity classes defined as sets of languages

$SPACE_k(s(n)) = \{L : L \text{ accepted by a deterministic } O(s(n)) - \text{space bounded } k \text{ tape } TM\}$

$$SPACE(t(n)) = \bigcup_k SPACE_k(s(n))$$

if you care for $s(n) < n$: keep input on read only *input tape*; count with $s(n)$ only tape cells visited on other so called *work tapes*.

Lemma 5.

$$\boxed{SPACE(t(n)) \subseteq SPACE_1(t(n))}$$

Proof. Shown in tape reduction theorem. □

not so obvious:

1 Defining complexitiy classes

Lemma 6. *Accepting and deciding by space bounded machines is the same, i.e. $L \in SPACE(t(n))$ iff there is an $O(t(n))$-space bounded TM which decides L (by state).*

- $\leftarrow$: trivial.

not so obvious:

1 Defining complexitiy classes

Lemma 6. *Accepting and deciding by space bounded machines is the same, i.e. $L \in SPACE(t(n))$ iff there is an $O(t(n))$-space bounded TM which decides L (by state).*

- $\leftarrow$: trivial.
- for $s \in \mathbb{N}$ count number $K(s)$ of configurations on space s

  $$K(s) \leq s \cdot \; |Z| \cdot |\Sigma|^s$$

  counting head positions, states, tape inscriptions with s cells.

  $$\begin{aligned} \log(K(s)) &\leq \log(s) + \log(|Z|) + s \cdot \log(|\Sigma|) \\ &= O(s) \end{aligned}$$

1 Defining complexitiy classes

not so obvious:

Lemma 6. *Accepting and deciding by space bounded machines is the same, i.e. $L \in SPACE(t(n))$ iff there is an $O(t(n))$-space bounded TM which decides L (by state).*

- $\leftarrow$: trivial.
- for $s \in \mathbb{N}$ count number $K(s)$ of configurations on space s

  $$K(s) \leq s \cdot \ |Z| \cdot |\Sigma|^s$$

  counting head positions, states, tape inscriptions with s cells.

  $$\begin{aligned} \log(K(s)) &\leq \log(s) + \log(|Z|) + s \cdot \log(|\Sigma|) \\ &= O(s) \end{aligned}$$

- deciding if M started with inputs $w\#bin(s)$ if M accepts w using space s:

  1. run M with input s counting number σ of cells visited and (in binary) number τ of steps made.
  2. if $\sigma > s$ reject; space bound exceeded.
  3. if $\tau > K(s)$ reject; a configuration repeats on space s, it will never halt
  4. accept if M accepts in space and time bound.

  This consumes space $n + O(s)$

not so obvious:

1 Defining complexitiy classes

Lemma 6. *Accepting and deciding by space bounded machines is the same, i.e. $L \in SPACE(t(n))$ iff there is an $O(t(n))$-space bounded TM which decides L (by state).*

- $\leftarrow$: trivial.
- for $s \in \mathbb{N}$ count number $K(s)$ of configurations on space s

$$K(s) \leq s \cdot \ |Z| \cdot |\Sigma|^s$$

  counting head positions, states, tape inscriptions with s cells.

$$\begin{aligned} \log(K(s)) &\leq \log(s) + \log(|Z|) + s \cdot \log(|\Sigma|) \\ &= O(s) \end{aligned}$$

- deciding if M started with inputs $w\#bin(s)$ if M accepts w using space s:
  1. run M with input s counting number σ of cells visited and (in binary) number τ of steps made.
  2. if $\sigma > s$ reject; space bound exceeded.
  3. if $\tau > K(s)$ reject; a configuration repeats on space s, it will never halt
  4. accept if M accepts in space and time bound.

  This consumes space $n + O(s)$
- decider for L:
  1. run above test for $s = n, n+1, \ldots$ until space bound s is not exceeded
  2. this succeeds for $s = s(n)$ using space $n + O(s(n))$

not so obvious:

Lemma 6. *Accepting and deciding by space bounded machines is the same, i.e. $L \in SPACE(t(n))$ iff there is an $O(t(n))$-space bounded TM which decides L (by state).*

- $\leftarrow$: trivial.
- for $s \in \mathbb{N}$ count number $K(s)$ of configurations on space s

$$K(s) \leq s \cdot |Z| \cdot |\Sigma|^s$$

counting head positions, states, tape inscriptions with s cells.

$$\begin{aligned} \log(K(s)) &\leq \log(s) + \log(|Z|) + s \cdot \log(|\Sigma|) \\ &= O(s) \end{aligned}$$

- deciding if M started with inputs $w\#bin(s)$ if M accepts w using space s:
  1. run M with input s counting number σ of cells visited and (in binary) number τ of steps made.
  2. if $\sigma > s$ reject; space bound exceeded.
  3. if $\tau > K(s)$ reject; a configuration repeats on space s, it will never halt
  4. accept if M accepts in space and time bound.

  This consumes space $n + O(s)$
- decider for L:
  1. run above test for $s = n, n+1, \dots$ until space bound s is not exceeded
  2. this succeeds for $s = s(n)$ using space $n + O(s(n))$

1.4 Exponential Time

Lemma 7.

$$SPACE(s(n)) \subseteq \bigcup_{C \in \mathbb{N}} TIME(2^{cs(n)})$$

Proof. The machine constructed in lemma 6 is $O(2^{C \cdot s(n)})$ time bounded for some C. Why? □

def: exponential time

$$EXPTIME = \bigcup_{C \in \mathbb{N}} TIME(2^{Cn})$$

Lemma 8.

$$SPACE(n) \subseteq EXPTIME$$

Proof. $s(n) = n$ in lemma 7 □

2 Constant tape compression and constant speed up

- artefacts of arbitrary size of tape alphabet
- explains why O-notation in bounds of complexity classes cannot be avoided

2 Constant tape compression and constant speed up

- artefacts of arbitrary size of tape alphabet
- explains why O-notation in bounds of complexity classes cannot be avoided

Lemma 9. *Let L be accepted by an $s(n)$ tape bounded Turing machine and $C > 0$. Then L is accepted by an $n+3+s(n)/C$ tape bounded Turing machine.*

2 Constant tape compression and constant speed up

- artefacts of arbitrary size of tape alphabet
- explains why O-notation in bounds of complexity classes cannot be avoided

Lemma 9. *Let L be accepted by an $s(n)$ tape bounded Turing machine and $C > 0$. Then L is accepted by an $n+2+s(n)/C$ tape bounded Turing machine.*

Assume 1-tape machine M accepts L (why can we assume this?)

- Replace tape alphabet A of M by

$$A' = A \cup A^C$$

- partition tape of M into blocks of C tape cells each
- compress input w to length $\lceil n/C \rceil$ using. This works in space $n+2$ (have to find ends of input).
- now simulate M on space $(n + \lceil s(n)/C \rceil)$

2 Constant tape compression and constant speed up

- artefacts of arbitrary size of tape alphabet
- explains why O-notation in bounds of complexity classes cannot be avoided

Lemma 9. *Let L be accepted by an $s(n)$ tape bounded Turing machine and $C > 0$. Then L is accepted by an $n+2+s(n)/C$ tape bounded Turing machine.*

Lemma 10. *Let L be accepted by a $t(n)$ time bounded Turing machine and $C > 0$. Then L is accepted by an $n+\lceil n/C \rceil +4+t(n)/C$ time bounded Turing machine.*

4t(n)/C

2 Constant tape compression and constant speed up

- artefacts of arbitrary size of tape alphabet
- explains why O-notation in bounds of complexity classes cannot be avoided

Lemma 9. *Let L be accepted by an $s(n)$ tape bounded Turing machine and $C > 0$. Then L is accepted by an $n+2+s(n)/C$ tape bounded Turing machine.*

Lemma 10. *Let L be accepted by a $t(n)$ time bounded Turing machine and $C > 0$. Then L is accepted by an $n+\lceil n/C \rceil +4+t(n)/C$ time bounded Turing machine.*

4t(n)/C

Assume k-tape machine M accepts L (why can we assume this?)

- Replace tape alphabet A of M by

$$A' = A \cup (A^C \times [0:C])$$

allowing to code in 1 symbol on each tape

1. a block of M of length C.
2. the head position within the block: 0 means: no head.

- partition tape of M into blocks of C tape cells each

2 Constant tape compression and constant speed up

- artefacts of arbitrary size of tape alphabet
- explains why O-notation in bounds of complexity classes cannot be avoided

Lemma 9. *Let L be accepted by an $s(n)$ tape bounded Turing machine and $C > 0$. Then L is accepted by an $n+2+s(n)/C$ tape bounded Turing machine.*

Lemma 10. *Let L be accepted by a $t(n)$ time bounded Turing machine and $C > 0$. Then L is accepted by an $n+\lceil n/C\rceil+4+t(n)/C$ time bounded Turing machine.*

$4\lceil t/C \rceil$

Assume k-tape machine M accepts L (why can we assume this?)

- Replace tape alphabet A of M by

$$A' = A \cup (A^C \times [0:C])$$

  allowing to code in 1 symbol on each tape

  1. a block of M of length C.
  2. the head position within the block: 0 means: no head.

- partition tape of M into blocks of C tape cells each

- compress input w to length $\lceil n/C\rceil$.
  1. copy input and compress to tape 2 tape; erase tape 1.
  2. this works in time $n+2$
  3. move head on tape 2 to start of compressed input
  4. this works in time $\lceil n/C\rceil+2$
  5. from now on exchange roles of tapes 1 and 2

2 Constant tape compression and constant speed up

- artefacts of arbitrary size of tape alphabet
- explains why O-notation in bounds of complexity classes cannot be avoided

Lemma 9. *Let L be accepted by an $s(n)$ tape bounded Turing machine and $C > 0$. Then L is accepted by an $n + 2 + s(n)/C$ tape bounded Turing machine.*

Lemma 10. *Let L be accepted by a $t(n)$ time bounded Turing machine and $C > 0$. Then L is accepted by an $n + \lceil n/C \rceil + 4 + t(n)/C$ time bounded Turing machine.*

$4\lceil t/C \rceil$

Assume k-tape machine M accepts L (why can we assume this?)

- Replace tape alphabet A of M by

$$A' = A \cup (A^C \times [0 : C])$$

  allowing to code in 1 symbol on each tape

  1. a block of M of length C.
  2. the head position within the block: 0 means: no head.

- partition tape of M into blocks of C tape cells each

- compress input w to length $\lceil n/C \rceil$.
  1. copy input and compress to tape 2 tape; erase tape 1.
  2. this works in time $n + 2$
  3. move head on tape 2 to start of compressed input
  4. this works in time $\lceil n/C \rceil + 2$
  5. from now on exchange roles of tapes 1 and 2

- in C steps machine M can move on each tape at most 1 block left or right. $\rightarrow$

  4 steps

  1. can model k steps of M by 1 step of a transition function δ' operating on blocks/new alphabet symbols.
  2. simulation of t steps of M in time $4\lceil t/C \rceil$

2 Constant tape compression and constant speed up

- artefacts of arbitrary size of tape alphabet
- explains why O-notation in bounds of complexity classes cannot be avoided

Lemma 9. *Let L be accepted by an $s(n)$ tape bounded Turing machine and $C > 0$. Then L is accepted by an $n+2+s(n)/C$ tape bounded Turing machine.*

Lemma 10. *Let L be accepted by a $t(n)$ time bounded Turing machine and $C > 0$. Then L is accepted by an $n+\lceil n/C\rceil+4+t(n)/C$ time bounded Turing machine.*

$4\lceil t/C\rceil$

Assume k-tape machine M accepts L (why can we assume this?)

- Replace tape alphabet A of M by

$$A' = A \cup (A^C \times [0:C])$$

allowing to code in 1 symbol on each tape

1. a block of M of length C.
2. the head position within the block: 0 means: no head.

- partition tape of M into blocks of C tape cells each

- compress input w to length $\lceil n/C\rceil$.

1. copy input and compress to tape 2 tape; erase tape 1.
2. this works in time $n+2$
3. move head on tape 2 to start of compressed input
4. this works in time $\lceil n/C\rceil+2$
5. from now on exchange roles of tapes 1 and 2

- in C steps machine M can move on each tape at most 1 block left or right. $\rightarrow$

4 steps

1. can model k steps of M by 1 step of a transition function δ' operating on blocks/new alphabet symbols.
2. simulation of t steps of M in time $4\lceil t/C\rceil$

- read current block, store in state
- if block left in <= C steps
  - read neighbor block
  - store in state
- determine these 2 blocks after C steps
- update both and move head to current block

2 Constant tape compression and constant speed up

- artefacts of arbitrary size of tape alphabet
- explains why O-notation in bounds of complexity classes cannot be avoided

Lemma 9. *Let L be accepted by an $s(n)$ tape bounded Turing machine and $C > 0$. Then L is accepted by an $n+2+s(n)/C$ tape bounded Turing machine.*

Lemma 10. *Let L be accepted by a $t(n)$ time bounded Turing machine and $C > 0$. Then L is accepted by an $n+\lceil n/C \rceil + 4 + t(n)/C$ time bounded Turing machine.*

$4\lceil t/C \rceil$

Assume k-tape machine M accepts L (why can we assume this?)

- Replace tape alphabet A of M by

$$A' = A \cup (A^C \times [0:C])$$

allowing to code in 1 symbol on each tape

1. a block of M of length C.
2. the head position within the block: 0 means: no head.

- partition tape of M into blocks of C tape cells each

- compress input w to length $\lceil n/C \rceil$.
  1. copy input and compress to tape 2 tape; erase tape 1.
  2. this works in time $n+2$
  3. move head on tape 2 to start of compressed input
  4. this works in time $\lceil n/C \rceil + 2$
  5. from now on exchange roles of tapes 1 and 2

- in C steps machine M can move on each tape at most 1 block left or right. $\rightarrow$

4 steps

1. can model k steps of M by 1 step of a transition function δ' operating on blocks/new alphabet symbols.
2. simulation of t steps of M in time $4\lceil t/C \rceil$

steps :

- read current block, store in state 1
- if block left in <= C steps
  - read neighbor block
  - store in state 1
- determine these 2 blocks after C steps
- update both and move head to current block 2

3 Universal Turing machines revisited

def: universal TM U : very slightly adjusted
For all $u \in \mathbb{B}^*$ and v in $\{0,1,\#\}^*$ (instead of $v \in \mathbb{B}^*$) machine U started with $u\#v$ simulates M_u started with input v.

3 Universal Turing machines revisited

def: universal TM U : very slightly adjusted

For all $u \in \mathbb{B}^*$ and v in $\{0,1,\#\}^*$ (instead of $v \in \mathbb{B}^*$) machine U started with $u\#v$ simulates M_u started with input v.

recall: coding of 1-tape TM M

$$M = (Z, A, \delta, z_0, Z_A, Z_R)$$

- number states and alphabet symbols

$$Z = \{z_0, \ldots, z_r\} \quad , \quad A = \{a_1, \ldots, a_s\}$$

  with z_0 as start state.

- end states

$$Z_A = \{z_{r-1}\} \quad , \quad Z_R = \{z_r\}$$

def: universal TM U : very slightly adjusted

For all $u \in \mathbb{B}^*$ and v in $\{0,1,\#\}^*$ (instead of $v \in \mathbb{B}^*$) machine U started with $u\#v$ simulates M_u started with input v.

recall: coding of 1-tape TM M

$$M = (Z, A, \delta, z_0, Z_A, Z_R)$$

- number states and alphabet symbols

$$Z = \{z_0, \dots, z_r\} \quad , \quad A = \{a_1, \dots, a_s\}$$

  with z_0 as start state.

- end states

$$Z_A = \{z_{r-1}\} \quad , \quad Z_R = \{z_r\}$$

- Let

$$f = \lceil \log(\max\{r+1, s+1\}) \rceil$$

  code single function values

$$\delta(z_i, a_j) = (z_k, a_\ell, m)$$

  as

$$w_{ijk\ell m} = \#bin_f(i)\#bin_f(j)\#bin_f(k)\#bin_f(\ell)\#m'\#\}$$

$$m' = \begin{cases} 00 & m = L \\ 01 & m = N \\ 10 & m = R \end{cases}$$

  So codes of states and letters all have length f.

def: universal TM U : very slightly adjusted

For all $u \in \mathbb{B}^*$ and v in $\{0,1,\#\}^*$ (instead of $v \in \mathbb{B}^*$) machine U started with $u\#v$ simulates M_u started with input v.

recall: coding of 1-tape TM M

$$M = (Z, A, \delta, z_0, Z_A, Z_R)$$

- number states and alphabet symbols

$$Z = \{z_0, \dots, z_r\} \quad , \quad A = \{a_1, \dots, a_s\}$$

  with z_0 as start state.

- end states

$$Z_A = \{z_{r-1}\} \quad , \quad Z_R = \{z_r\}$$

- Let

$$f = \lceil \log(\max\{r+1, s+1\}) \rceil$$

  code single function values

$$\delta(z_i, a_j) = (z_k, a_\ell, m)$$

  as

$$w_{ijk\ell m} = \#bin_f(i)\#bin_f(j)\#bin_f(k)\#bin_f(\ell)\#m'\#\}$$

$$m' = \begin{cases} 00 & m = L \\ 01 & m = N \\ 10 & m = R \end{cases}$$

  So codes of states and letters all have length f.

- $code'(M)$: concatenate words $w_{ijk\ell m}$ and put one symbol # in front. Now all words $w_{ijk\ell m}$ can be found by searching for ##.
- compute

$$code(M) = h(code'(M))$$

  by replacing

$$0 \to 00\,, 1 \to 01\,,\ \# \to 10$$

- define M_u as the machine M with $code(M) = u$ if it exists, a trivial machine otherwise.

def: universal TM U : very slightly adjusted

For all $u \in \mathbb{B}^*$ and v in $\{0,1,\#\}^*$ (instead of $v \in \mathbb{B}^*$) machine U started with $u\#v$ simulates M_u started with input v.

recall: coding of 1-tape TM M

$$M = (Z, A, \delta, z_0, Z_A, Z_R)$$

- number states and alphabet symbols

$$Z = \{z_0, \ldots, z_r\} \quad , \quad A = \{a_1, \ldots, a_s\}$$

with z_0 as start state.

- end states

$$Z_A = \{z_{r-1}\} \quad , \quad Z_R = \{z_r\}$$

- Let

$$f = \lceil \log(\max\{r+1, s+1\}) \rceil$$

code single function values

$$\delta(z_i, a_j) = (z_k, a_\ell, m)$$

as

$$w_{ijk\ell m} = \#bin_f(i)\#bin_f(j)\#bin_f(k)\#bin_f(\ell)\#m'\#\}$$

$$m' = \begin{cases} 00 & m = L \\ 01 & m = N \\ 10 & m = R \end{cases}$$

So codes of states and letters all have length f.

- $code'(M)$: concatenate words $w_{ijk\ell m}$ and put one symbol # in front. Now all words $w_{ijk\ell m}$ can be found by searching for ##.

- compute

$$code(M) = h(code'(M))$$

by replacing

$$0 \to 00 \, , 1 \to 01 \, , \# \to 10$$

- define M_u as the machine M with $code(M) = u$ if it exists, a trivial machine otherwise.

Lemma 11. *There is a universal 1-tape Turing machine U which simulates t steps of M_u started with input v of length $|v| = n$ in time*

$$O((|u| \cdot n)^2 + |u|^2 \cdot t)$$

If M_u uses space s then U uses space $O(|u| \cdot s)$.

Lemma 11. *There is a universal 1-tape Turing machine U which simulates t steps of M_u started with input v of length $|v| = n$ in time*

$$O((|u| \cdot n)^2 + |u|^2 \cdot t)$$

If M_u uses space s then U uses space $O(|u| \cdot s)$.

Lemma 11. *There is a universal 1-tape Turing machine U which simulates t steps of M_u started with input v of length $|v| = n$ in time*

$$O((|u| \cdot n)^2 + |u|^2 \cdot t)$$

If M_u uses space s then U uses space $O(|u| \cdot s)$.

- for encoding configurations of M_u as in figure 1

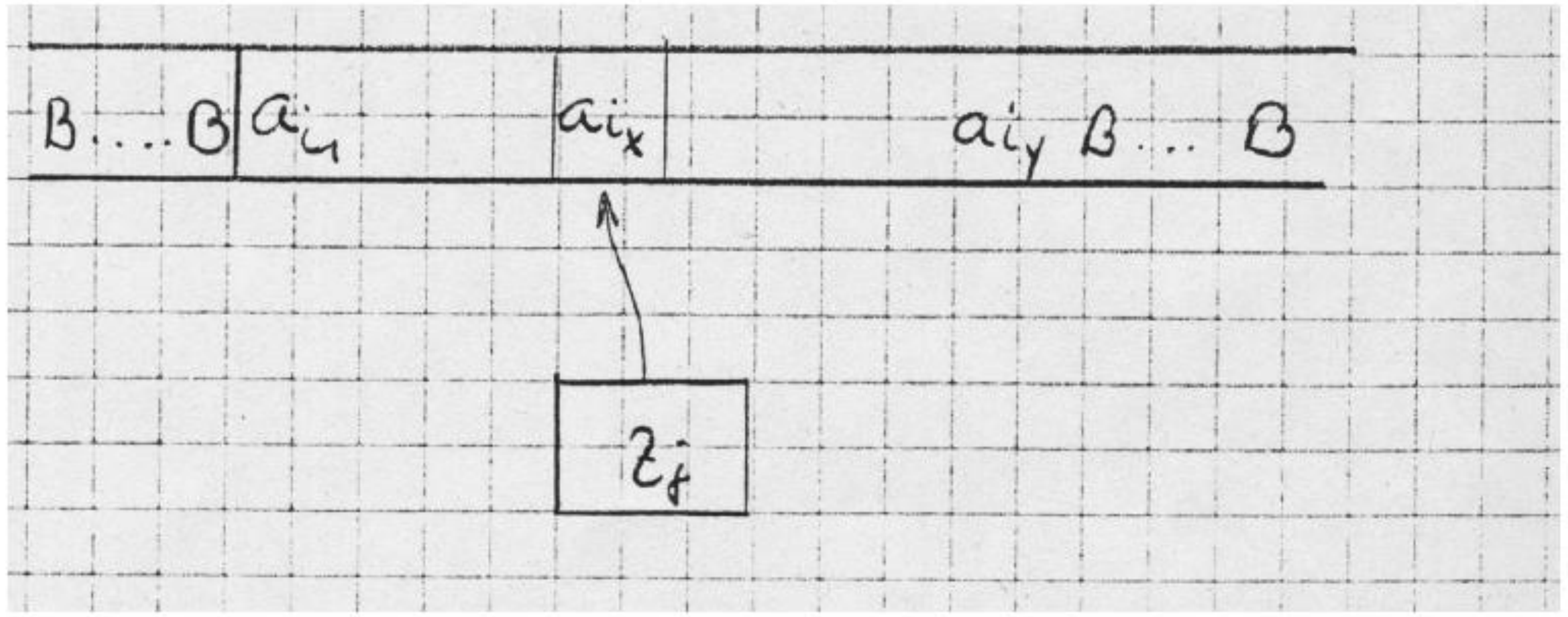

Figure 1: configuration k of 1-tape TM M

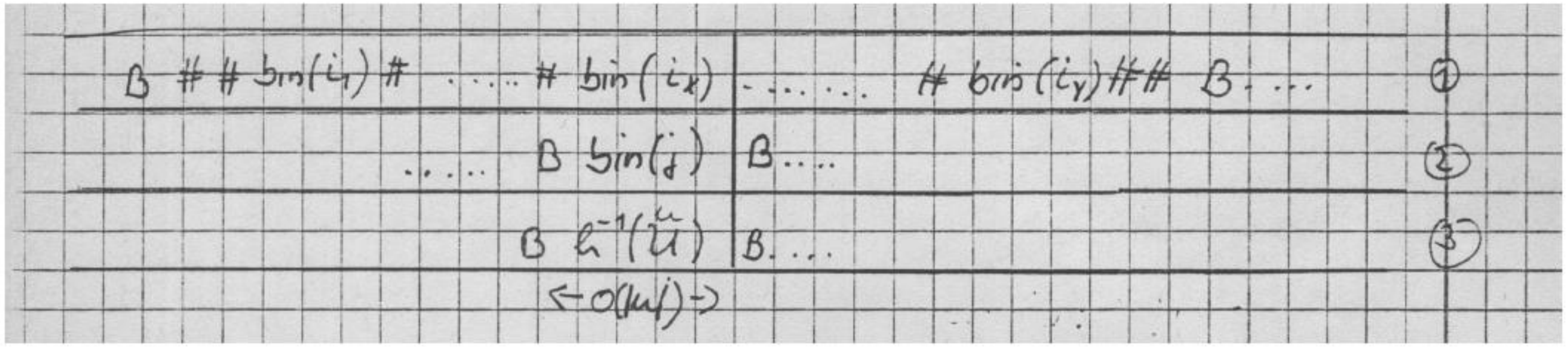

Figure 2: encoding configuration k on the first 3 tracks of universal machine U

1. track 1: encodes the tape of M_u in the obvious way.
2. track 2: codes the state of M in the obvious way
3. track 3: $code'(M_u)$ starting at head position of M_u
4. track 4: used for simulating steps
5. more tracks possible for comfort of programmming

Lemma 11. *There is a universal 1-tape Turing machine U which simulates t steps of M_u started with input v of length $|v| = n$ in time*

$$O((|u| \cdot n)^2 + |u|^2 \cdot t)$$

If M_u uses space s then U uses space $O(|u| \cdot s)$.

- for encoding configurations of M_u as in figure 1

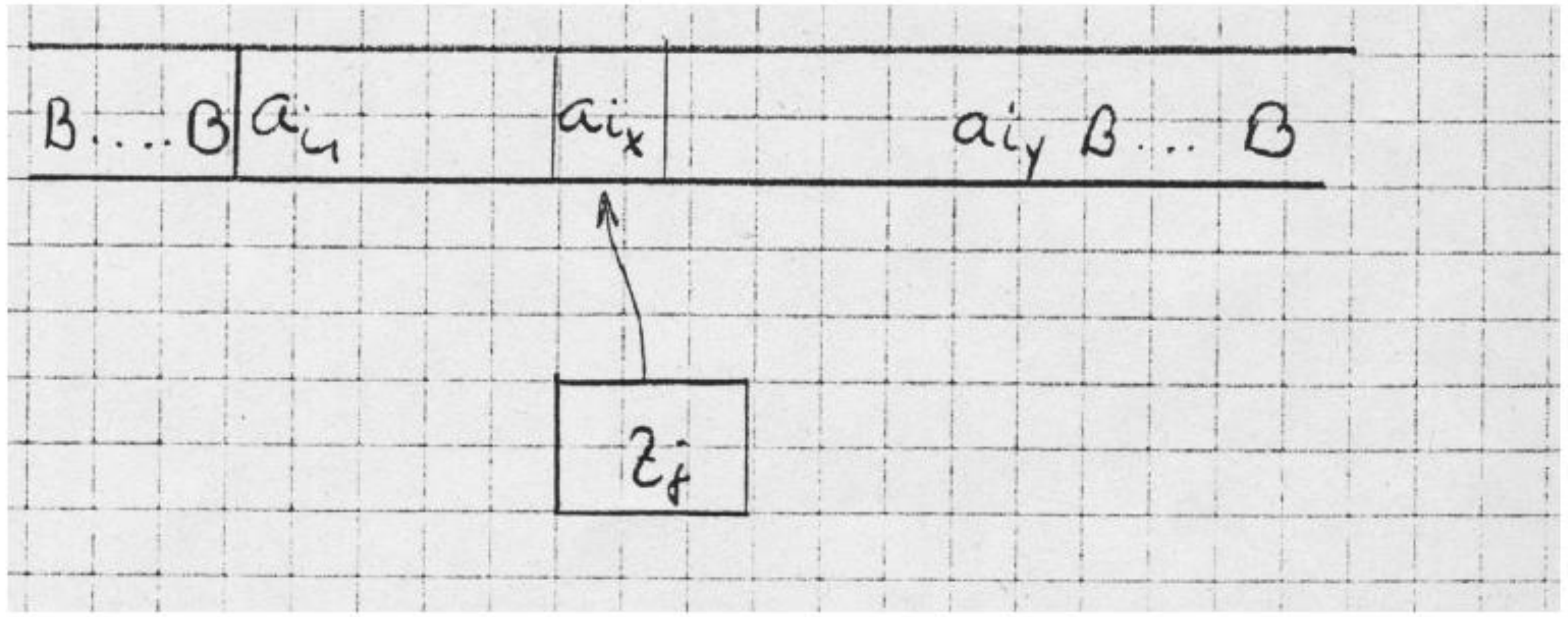

Figure 1: configuration k of 1-tape TM M

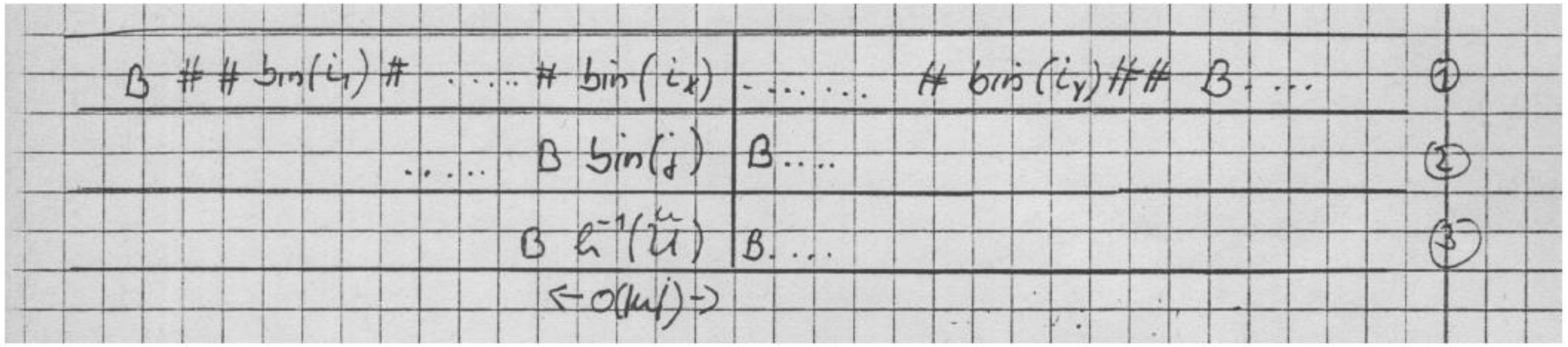

Figure 2: encoding configuration k on the first 3 tracks of universal machine U

1. track 1: encodes the tape of M_u in the obvious way.
2. track 2: codes the state of M in the obvious way
3. track 3: $code'(M_u)$ starting at head position of M_u
4. track 4: used for simulating steps
5. more tracks possible for comfort of programmming

- preprocessor
  1. unpack on track 2 $u = code(M_u)$ to
     $$code'(M_u) = h^{-1}(u)$$
     replacing in u from left to right
     $$00 \to 0\ ,\ 01 \to 1\ ,\ 10 \to \#$$
     time $O(|u|^2)$
  2. expand symbols $0, 1, \#$ in input v to length f and separate by symbols #.
     time $O((|u| \cdot n)^2)$ and space $O(|u| \cdot n)$.
  3. move $code'(M_u)$ under start of inscription of track 1
     time $O(|u|^2)$.

Lemma 11. *There is a universal 1-tape Turing machine U which simulates t steps of M_u started with input v of length $|v| = n$ in time*

$$O((|u| \cdot n)^2 + |u|^2 \cdot t)$$

If M_u uses space s then U uses space $O(|u| \cdot s)$.

- for encoding configurations of M_u as in figure 1

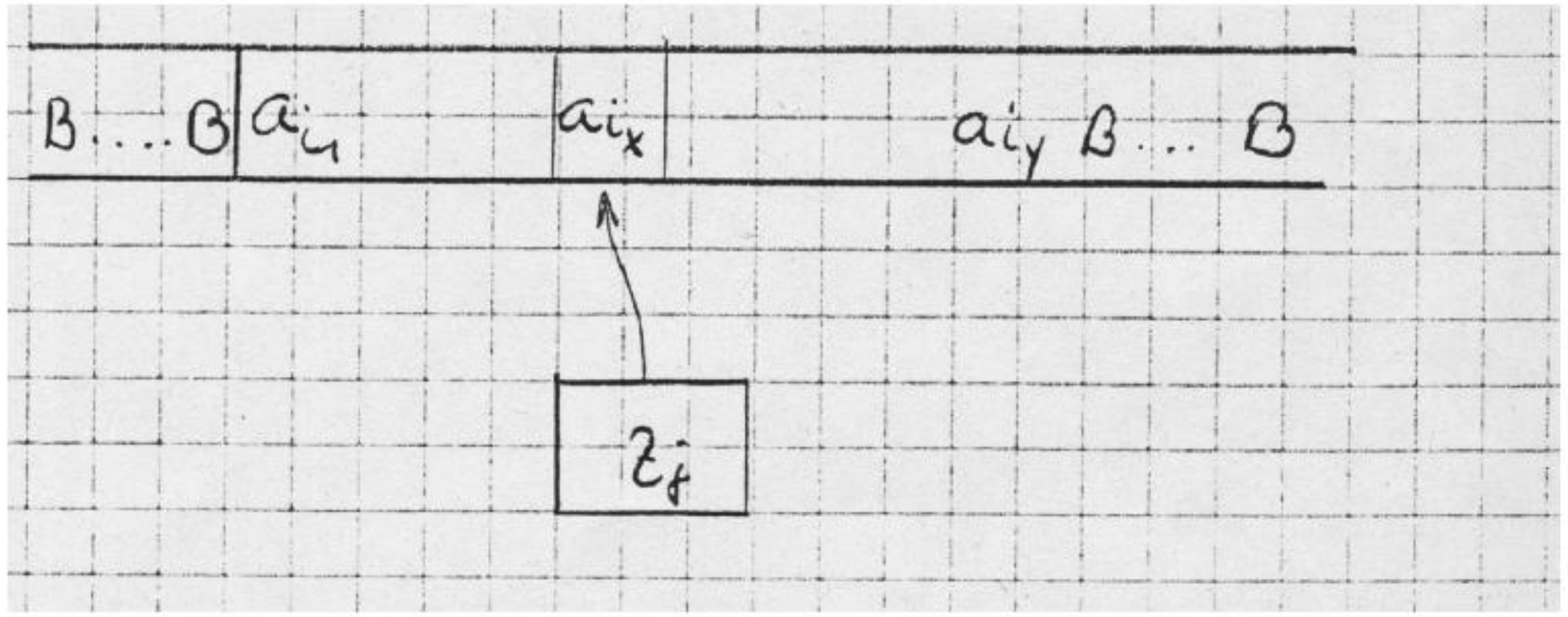

Figure 1: configuration k of 1-tape TM M

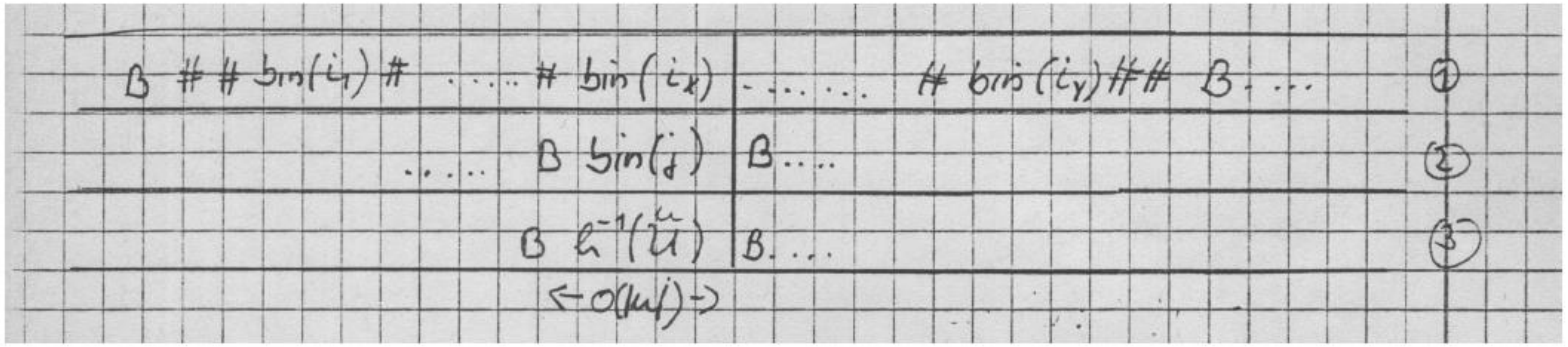

Figure 2: encoding configuration k on the first 3 tracks of universal machine U

1. track 1: encodes the tape of M_u in the obvious way.
2. track 2: codes the state of M in the obvious way
3. track 3: $code'(M_u)$ starting at head position of M_u
4. track 4: used for simulating steps
5. more tracks possible for comfort of programmming

- simulating a step. Notation from figure 2.

1. produce on track 4 $\#\#bin_f(j)\#bin(i_x)$.
2. search for this pattern in $code'(M_u)$ on track 3. If it is not found, state z_j is an end state. Accept if $j = r - 1$.
3. if it is followed by $bin_f(k)\#bin_f(\ell)\#m'$ (unless the computation ends), overwrite track 4 by it and move this back to head position (where inscription of track 2 starts)
4. overwrite state on track 2 with $bin_f(k)$ and symbol on track 1 with $bin_f(\ell)$.
5. depending on m move inscriptions on track 2 and 3 $f+1$ symbols left, right or leave them in place. Then erase track 4.

Lemma 11. *There is a universal 1-tape Turing machine U which simulates t steps of M_u started with input v of length $|v| = n$ in time*

$$O((|u| \cdot n)^2 + |u|^2 \cdot t)$$

If M_u uses space s then U uses space $O(|u| \cdot s)$.

- for encoding configurations of M_u as in figure 1

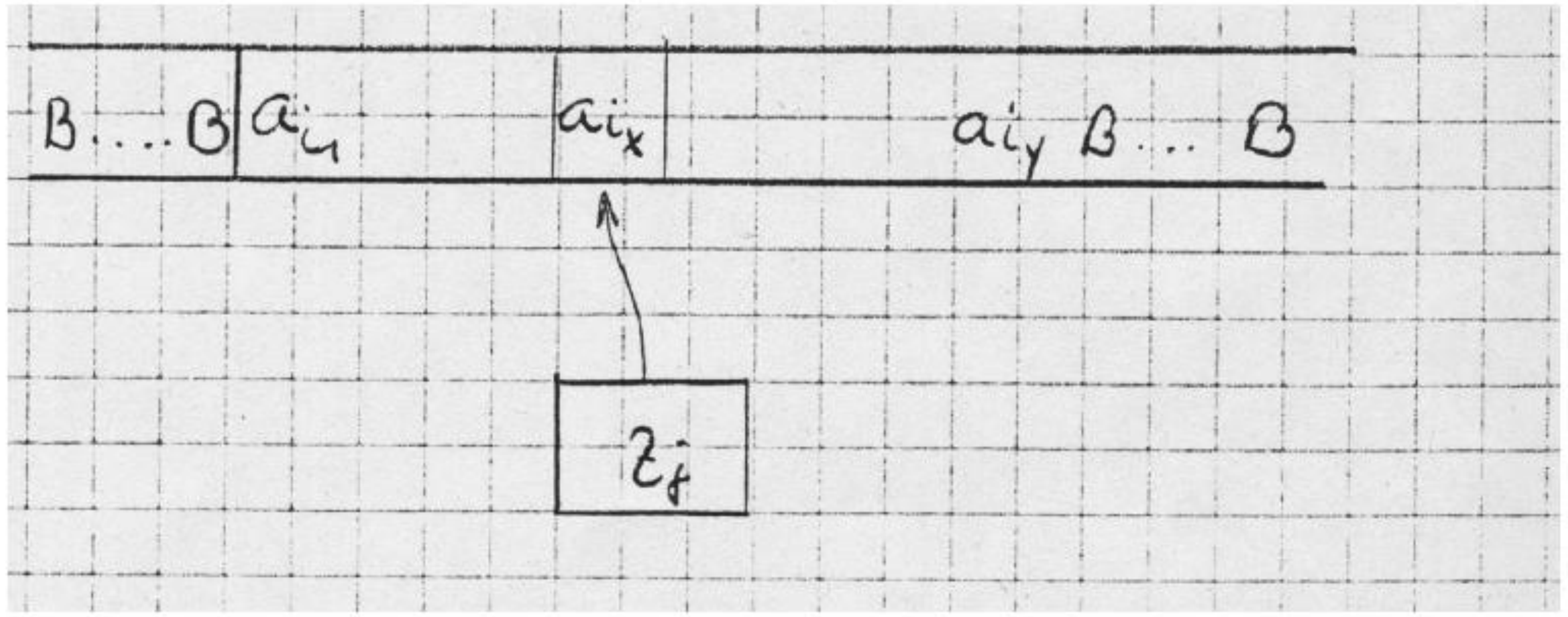

Figure 1: configuration k of 1-tape TM M

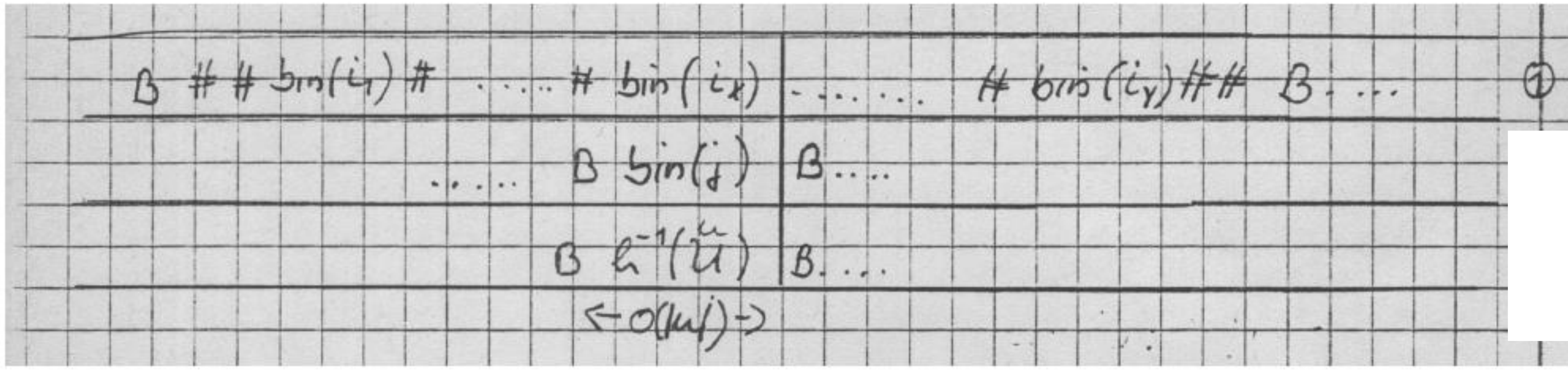

Figure 2: encoding configuration k on the first 3 tracks of universal machine U

1. track 1: encodes the tape of M_u in the obvious way.
2. track 2: codes the state of M in the obvious way
3. track 3: $code'(M_u)$ starting at head position of M_u
4. track 4: used for simulating steps
5. more tracks possible for comfort of programmming

- simulating a step. Notation from figure 2.
  1. produce on track 4 $\#\#bin_f(j)\#bin(i_x)$.
  2. search for this pattern in $code'(M_u)$ on track 3. If it is not found, state z_j is an end state. Accept if $j = r - 1$.
  3. if it is followed by $bin_f(k)\#bin_f(\ell)\#m'$ (unless the computation ends), overwrite track 4 by it and move this back to head position (where inscription of track 2 starts)
  4. overwrite state on track 2 with $bin_f(k)$ and symbol on track 1 with $bin_f(\ell)$.
  5. depending on m move inscriptions on track 2 and 3 $f+1$ symbols left, right or leave them in place. Then erase track 4.

Simulation of a step takes time $O(|u|^2)$. If the tape inscription of M_u has length s, then the inscription of track 1 of U has length

$$(f+1) \cdot s + 1 = O(|u| \cdot s)$$

Lemma 11. *There is a universal 1-tape Turing machine U which simulates t steps of M_u started with input v of length $|v| = n$ in time*

$$O((|u| \cdot n)^2 + |u|^2 \cdot t)$$

If M_u uses space s then U uses space $O(|u| \cdot s)$.

Lemma 12. *There is a universal 2-tape Turing machine U which simulates t steps of 1-tape TM M_u started with input v of length $|v| = n$ in time*

$$O(|u| \cdot (n + t)$$

If M_u uses space s then U uses space $O(|u| \cdot s)$.

Proof. exercise: why do the quadratic terms in the time bound disappear? □

4 Time and space constructible functions

setting a kitchen timer to 3 minutes should not take more than 3 minutes.

4 Time and space constructible functions

$\boxed{\textbf{setting a kitchen timer to 3 minutes should not take more than 3 minutes.}}$

Let

$$s,t : \mathbb{N}_0 \to \mathbb{N}_0$$

be functions intended as resource bounds.

def: time constructible Function t is *time constructible* if there is an $O(t(n))$-time bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(t(n))$ and halts.

def: space constructible Function s is *space constructible* if there is an $O(s(n))$-space bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(s(n))$ and halts.

4 Time and space constructible functions

setting a kitchen timer to 3 minutes should not take more than 3 minutes.

Let

$$s, t : \mathbb{N}_0 \to \mathbb{N}_0$$

be functions intended as resource bounds.

def: time constructible Function t is *time constructible* if there is an $O(t(n))$-time bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(t(n))$ and halts.

def: space constructible Function s is *space constructible* if there is an $O(s(n))$-space bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(s(n))$ and halts.

Lemma 13. *The identity function $t(n) = n$ is time and space constructible by a 2-tape TM.*

4 Time and space constructible functions

setting a kitchen timer to 3 minutes should not take more than 3 minutes.

Let

$$s, t : \mathbb{N}_0 \to \mathbb{N}_0$$

be functions intended as resource bounds.

def: time constructible Function t is *time constructible* if there is an $O(t(n))$-time bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(t(n))$ and halts.

def: space constructible Function s is *space constructible* if there is an $O(s(n))$-space bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(s(n))$ and halts.

Lemma 13. *The identity function $t(n) = n$ is time and space constructible by a 2-tape TM.*

- round 0: initialize tape 2=0.
- round $r > 0$:
  if head 1 does not read B :
  { tape 2 = tape 2 + 1 (binary); move head 1 right }
  else stop

4 Time and space constructible functions

setting a kitchen timer to 3 minutes should not take more than 3 minutes.

Let

$$s, t : \mathbb{N}_0 \to \mathbb{N}_0$$

be functions intended as resource bounds.

def: time constructible Function t is *time constructible* if there is an $O(t(n))$-time bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(t(n))$ and halts.

def: space constructible Function s is *space constructible* if there is an $O(s(n))$-space bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(s(n))$ and halts.

Lemma 13. *The identity function $t(n) = n$ is time and space constructible by a 2-tape TM.*

- round 0: initialize tape 2=0.
- round $r > 0$:
  if head 1 does not read B :
  { tape 2 = tape 2 + 1 (binary); move head 1 right }
  else stop

- for all i the number of rounds when i cells are visited in round i is $n/2^i$.
  Why? Time bound

$$\begin{aligned} t(n) &= O(\sum_{i=1}^{\lceil \log n \rceil} n \cdot i/2^i) \\ &\leq O(n \cdot (\sum_{i=1}^{\infty} i/2^i)) \\ &= O(n) \quad \text{(convergent series)} \end{aligned}$$

4 Time and space constructible functions

setting a kitchen timer to 3 minutes should not take more than 3 minutes.

Let

$$s,t:\mathbb{N}_0 \to \mathbb{N}_0$$

be functions intended as resource bounds.

def: time constructible Function t is *time constructible* if there is an $O(t(n))$-time bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(t(n))$ and halts.

def: space constructible Function s is *space constructible* if there is an $O(s(n))$-space bounded TM M such that for all n machine M started with the unary representation 1^n of n (or any string of length n) outputs $bin(s(n))$ and halts.

Lemma 13. *The identity function $t(n)=n$ is time and space constructible by a 2-tape TM.*

- round 0: initialize tape 2=0.
- round $r>0$:
  if head 1 does not read B :
  { tape 2 = tape 2 + 1 (binary); move head 1 right }
  else stop

- for all i the number of rounds when i cells are visited in round i is $n/2^i$.
  Why? Time bound

$$\begin{aligned} t(n) &= O(\sum_{i=1}^{\lceil \log n \rceil} n \cdot i/2^i) \\ &\leq O(n\cdot(\sum_{i=1}^{\infty} i/2^i)) \\ &= O(n) \quad \text{(convergent series)} \end{aligned}$$

Lemma 14. *Functions $n^k, 2^n, n\lceil \log n \rceil$ are time and space constructible*

Proof. Compute $bin(n)$ as in lemma 13, then use binary arithmetic. □

5 Hierarchy theorems

5.1 Time hierarchy

the crucial diagonalisation argument

Lemma 15. *Let*

$$n \leq t'(n) = o(T(n))$$

and let $T(n)$ be time constructible on 2 tapes. Then

$$TIME_1(t'(n)) \subsetneq TIME_2(T(n))$$

Hartmanis and Stearns 1965

created the field of complexity theory

Lemma 15. *Let*

$$n \leq t'(n) = o(T(n))$$

and let $T(n)$ be time constructible on 2 tapes. Then

$$TIME_1(t'(n)) \subsetneq TIME_2(T(n))$$

- 2 tape TM M started with input $u\#v$ with $u, v \in \mathbb{B}^*$ and length $n = |u\#v|$
    1. computes $bin(T(n))$ on tape 2 and $u\#u\#v$ on tape 1.
    2. behaves on tape 1 like a universal 1-tape TM U with input $u\#u\#v$, i.e. it simulates 1-tape TM M_u on input $u\#v$.
    3. it does this for $T(n)$ steps of U. If the simulation succeeds in this time bound and M_u accepts, thenn M rejects, otherwise M accepts.

Lemma 15. *Let*

$$n \leq t'(n) = o(T(n))$$

and let $T(n)$ be time constructible on 2 tapes. Then

$$TIME_1(t'(n)) \subsetneq TIME_2(T(n))$$

- 2 tape TM M started with input $u\#v$ with $u, v \in \mathbb{B}^*$ and length $n = |u\#v|$

  1. computes $bin(T(n))$ on tape 2 and $u\#u\#v$ on tape 1.
  2. behaves on tape 1 like a universal 1-tape TM U with input $u\#u\#v$, i.e. it simulates 1-tape TM M_u on input $u\#v$.
  3. it does this for $T(n)$ steps of U. If the simulation succeeds in this time bound and M_u accepts, thenn M rejects, otherwise M accepts.

- M counts $T(n)$ steps in time $O(T(n))$ by argument of lemma 13 (subtract instead of add).

$$L(M) \in TIME_2(T(n))$$

Lemma 15. *Let*

$$n \leq t'(n) = o(T(n))$$

and let $T(n)$ be time constructible on 2 tapes. Then

$$TIME_1(t'(n)) \subsetneq TIME_2(T(n))$$

- 2 tape TM M started with input $u\#v$ with $u, v \in \mathbb{B}^*$ and length $n = |u\#v|$

  1. computes $bin(T(n))$ on tape 2 and $u\#u\#v$ on tape 1.
  2. behaves on tape 1 like a universal 1-tape TM U with input $u\#u\#v$, i.e. it simulates 1-tape TM M_u on input $u\#v$.
  3. it does this for $T(n)$ steps of U. If the simulation succeeds in this time bound and M_u accepts, thenn M rejects, otherwise M accepts.

- M counts $T(n)$ steps in time $O(T(n))$ by argument of lemma 13 (subtract instead of add).

$$L(M) \in TIME_2(T(n))$$

- assume $L(M)$ is accepted by $O(t'(n))$-time bounded 1-tape TM M_u,i.e.

$$L(M) = L(M_u)$$

  Consider inputs of the form $u\#v$ with $|u\#v| = n$. By lemma 3: simulation of M_u with this input takes time

$$O(|u| \cdot (n + t'(n)) \leq C \cdot |u| \cdot 2t'(n) \quad \text{for all } n \geq n_0$$

  Simulation succeeds if

$$2C \cdot |u| \cdot t'(n) \leq T(n)$$

  resp.

$$t'(n)/T(n) \leq 1/(2C \cdot |u|)$$

  which holds for all $n \geq n_1$ as $t'(n) = o(T(n))$.

Lemma 15. *Let*

$$n \leq t'(n) = o(T(n))$$

and let $T(n)$ be time constructible on 2 tapes. Then

$$TIME_1(t'(n)) \subsetneq TIME_2(T(n))$$

- 2 tape TM M started with input $u\#v$ with $u, v \in \mathbb{B}^*$ and length $n = |u\#v|$
  1. computes $bin(T(n))$ on tape 2 and $u\#u\#v$ on tape 1.
  2. behaves on tape 1 like a universal 1-tape TM U with input $u\#u\#v$, i.e. it simulates 1-tape TM M_u on input $u\#v$.
  3. it does this for $T(n)$ steps of U. If the simulation succeeds in this time bound and M_u accepts, thenn M rejects, otherwise M accepts.

- M counts $T(n)$ steps in time $O(T(n))$ by argument of lemma 13 (subtract instead of add).

$$L(M) \in TIME_2(T(n))$$

- assume $L(M)$ is accepted by $O(t'(n))$-time bounded 1-tape TM M_u,i.e.

$$L(M) = L(M_u)$$

  Consider inputs of the form $u\#v$ with $|u\#v| = n$. By lemma 3: simulation of M_u with this input takes time

$$O(|u| \cdot (n + t'(n)) \leq C \cdot |u| \cdot 2t'(n) \quad \text{for all } n \geq n_0$$

  Simulation succeeds if

$$2C \cdot |u| \cdot t'(n) \leq T(n)$$

  resp.

$$t'(n)/T(n) \leq 1/(2C \cdot |u|)$$

  which holds for all $n \geq n_1$ as $t'(n) = o(T(n))$.

- For $u\#v$ with $n = |u\#v| \geq \max\{n_0, n_1\}$

$$\begin{aligned} u\#v \in L(M_u) &\leftrightarrow M_u \text{ started with } u\#v \text{ accepts} \\ &\leftrightarrow M \text{ started with } u\#v \text{ rejects} \\ &\leftrightarrow u\#v \notin L(M) \end{aligned}$$

Lemma 15. *Let*

$$n \leq t'(n) = o(T(n))$$

and let $T(n)$ be time constructible on 2 tapes. Then

$$TIME_1(t'(n)) \subsetneq TIME_2(T(n))$$

time hierarchy theorem

Lemma 16. *Let $T(n)$ be time constructible on 2 tapes and $t^2(n) = o(T(n))$, then*

$$TIME(t(n)) \subsetneq TIME(T(n))$$

Lemma 15. *Let*

$$n \le t'(n) = o(T(n))$$

and let $T(n)$ be time constructible on 2 tapes. Then

$$TIME_1(t'(n)) \subsetneq TIME_2(T(n))$$

time hierarchy theorem

Lemma 16. *Let $T(n)$ be time constructible on 2 tapes and $t^2(n) = o(T(n))$, then*

$$TIME(t(n)) \subsetneq TIME(T(n))$$

$$\begin{array}{rcll} TIME(t(n)) & \subseteq & TIME_1(t^2(n)) & \text{(lemma 4)} \\ & \subsetneq & TIME_2(T(n)) & \text{(lemma 18)} \\ & \subseteq & TIME(T(N)) & \end{array}$$

Lemma 15. *Let*

$$n \leq t'(n) = o(T(n))$$

and let $T(n)$ be time constructible on 2 tapes. Then

$$TIME_1(t'(n)) \subsetneq TIME_2(T(n))$$

time hierarchy theorem

Lemma 16. *Let $T(n)$ be time constructible on 2 tapes and $t^2(n) = o(T(n))$, then*

$$TIME(t(n)) \subsetneq TIME(T(n))$$

Hartmanis and Stearns 1965

created the field of complexity theory

$$\begin{aligned} TIME(t(n)) &\subseteq TIME_1(t^2(n)) && \text{(lemma 4)} \\ &\subsetneq TIME_2(T(n)) && \text{(lemma 18)} \\ &\subseteq TIME(T(N)) \end{aligned}$$

without proof: improved tape reduction theorem

Lemma 17. *Let t be time constructible. Then*

$$TIME(t(n)) \subseteq TIME_2(t(n)\log(t(n)))$$

Hennie and Stearns 1966

exercise : conclude a tighter time hierarchy theorem

5.2 Space hierarchy

Lemma 18. *Let*

$$n \leq s(n) = o(S(n))$$

and let $S(n)$ be space constructible. Then

$$SPACE_1(s(n)) \subsetneq SPACE_3(S(n))$$

Hartmanis, Lewis and Stearns 1965

like time hierarchy theorem but recycling the proof of the existence of tape bounded deciders

5.2 Space hierarchy

Lemma 18. *Let*

$$n \leq s(n) = o(S(n))$$

and let $S(n)$ be space constructible. Then

$$SPACE_1(s(n)) \subsetneq SPACE_3(S(n))$$

- 3 tape TM M started with input $u\#v$ with $u, v \in \mathbb{B}^*$ and length $n = |u\#v|$
  1. computes space bound $bin(S(n))$ on tape 2 and $u\#u\#v$ on tape 1.
  2. marks on tape 1 distance $S(n)$ both to the left and right of the original head position
  3. computes on tape 3 a bound

     $$\cancel{T(n) = 2^n} \qquad T(n) = 2^{S(n)}$$

     for the number of steps of the universal machine U. This takes space $O(S(n))$.
  4. behaves on tape 1 like a universal 1-tape TM U with input $u\#u\#v$, i.e. it simulates 1-tape TM M_u on input $u\#v$.
  5. it does this for at most $T(n)$ steps of U and only as long as the marked space is not exceeded. If the simulation succeeds in these bounds and M_u accepts, then M rejects, otherwise M accepts.

5.2 Space hierarchy

Lemma 18. *Let*

$$n \le s(n) = o(S(n))$$

and let $S(n)$ be space constructible. Then

$$SPACE_1(s(n)) \subsetneq SPACE_3(S(n))$$

- 3 tape TM M started with input $u\#v$ with $u,v \in \mathbb{B}^*$ and length $n = |u\#v|$
  1. computes space bound $bin(S(n))$ on tape 2 and $u\#u\#v$ on tape 1.
  2. marks on tape 1 distance $S(n)$ both to the left and right of the original head position
  3. computes on tape 3 a bound

     $$\cancel{T(n) = 2^n} \qquad T(n) = 2^{S(n)}$$

     for the number of steps of the universal machine U. This takes space $O(S(n))$.
  4. behaves on tape 1 like a universal 1-tape TM U with input $u\#u\#v$, i.e. it simulates 1-tape TM M_u on input $u\#v$.
  5. it does this for at most $T(n)$ steps of U and only as long as the marked space is not exceeded. If the simulation succeeds in these bounds and M_u accepts, then M rejects, otherwise M accepts.
- M is $O(S(n))$ space bounded, thus

$$L(M) \in SPACE_3(S(n))$$

5.2 Space hierarchy

Lemma 18. *Let*

$$n \leq s(n) = o(S(n))$$

and let $S(n)$ be space constructible. Then

$$SPACE_1(s(n)) \subsetneq SPACE_3(S(n))$$

- 3 tape TM M started with input $u\#v$ with $u, v \in \mathbb{B}^*$ and length $n = |u\#v|$
  1. computes space bound $bin(S(n))$ on tape 2 and $u\#u\#v$ on tape 1.
  2. marks on tape 1 distance $S(n)$ both to the left and right of the original head position
  3. computes on tape 3 a bound

     $$\cancel{T(n) = 2^n} \qquad T(n) = 2^{S(n)}$$

     for the number of steps of the universal machine U. This takes space $O(S(n))$.
  4. behaves on tape 1 like a universal 1-tape TM U with input $u\#u\#v$, i.e. it simulates 1-tape TM M_u on input $u\#v$.
  5. it does this for at most $T(n)$ steps of U and only as long as the marked space is not exceeded. If the simulation succeeds in these bounds and M_u accepts, then M rejects, otherwise M accepts.

- M is $O(S(n))$ space bounded, thus

  $$L(M) \in SPACE_3(S(n))$$

- assume $L(M)$ is accepted by $O(s(n)))$-space bounded 1-tape TM M_u,i.e.

  $$L(M) = L(M_u)$$

  Consider inputs of the form $u\#v$ with $|u\#v| = n$. By lemma 3: simulation of M_u with this input succeeds in space

  $$O(|u| \cdot s(n)) \leq C \cdot |u| \cdot s(n) \quad \text{for all } n \geq n_0$$

  Simulation does not run out of space if

  $$C \cdot |u| \cdot s(n) \leq S(n)$$

  resp.

  $$s(n)/S(n) \leq 1/(C \cdot |u|)$$

  which holds for all $n \geq n_1$ as $s(n) = o(S(n))$.

5.2 Space hierarchy

Lemma 18. *Let*

$$n \leq s(n) = o(S(n))$$

and let $S(n)$ be space constructible. Then

$$SPACE_1(s(n)) \subsetneq SPACE_3(S(n))$$

- 3 tape TM M started with input $u\#v$ with $u, v \in \mathbb{B}^*$ and length $n = |u\#v|$
  1. computes space bound $bin(S(n))$ on tape 2 and $u\#u\#v$ on tape 1.
  2. marks on tape 1 distance $S(n)$ both to the left and right of the original head position
  3. computes on tape 3 a bound

     $$\cancel{T(n) = 2^n} \qquad T(n) = 2^{S(n)}$$

     for the number of steps of the universal machine U. This takes space $O(S(n))$.
  4. behaves on tape 1 like a universal 1-tape TM U with input $u\#u\#v$, i.e. it simulates 1-tape TM M_u on input $u\#v$.
  5. it does this for at most $T(n)$ steps of U and only as long as the marked space is not exceeded. If the simulation succeeds in these bounds and M_u accepts, then M rejects, otherwise M accepts.

- M is $O(S(n))$ space bounded, thus

$$L(M) \in SPACE_3(S(n))$$

- assume $L(M)$ is accepted by $O(s(n)))$-space bounded 1-tape TM M_u,i.e.

$$L(M) = L(M_u)$$

  Consider inputs of the form $u\#v$ with $|u\#v| = n$. By lemma 3: simulation of M_u with this input succeeds in space

$$O(|u| \cdot s(n)) \leq C \cdot |u| \cdot s(n) \quad \text{for all } n \geq n_0$$

  Simulation does not run out of space if

$$C \cdot |u| \cdot s(n) \leq S(n)$$

  resp.

$$s(n)/S(n) \leq 1/(C \cdot |u|)$$

  which holds for all $n \geq n_1$ as $s(n) = o(S(n))$.

  - the number of configurations of M_u on space $s(n)$ is bounded by

$$\begin{aligned} s(n) \cdot |Z_u| \cdot |\Sigma_u|^{s(n)} &\leq s(n) \cdot |u| \cdot |u|^{s(n)} \\ &= 2^{\log(s(n)) + \log|u| \cdot (s(n)+1)} \\ &< 2^{S(n)} \quad \text{for } n \geq n_2 \end{aligned}$$

    Hence for $n > \max\{n_0, n_1, n_2\}$ if M accepts $u\#v$ because the time bound is exceeded, we have $u\#v \notin L(M_u)$ because M_u does not halt.

- For $u\#v$ with $n = |u\#v| \geq \max\{n_0, n_1, n_2\}$

$$\begin{aligned} u\#v \in L(M_u) &\leftrightarrow M_u \text{ started with } u\#v \text{ accepts} \\ &\leftrightarrow M \text{ started with } u\#v \text{ rejects} \\ &\leftrightarrow u\#v \notin L(M) \end{aligned}$$

- For $u\#v$ with $n = |u\#v| \geq \max\{n_0, n_1, n_2\}$

$$\begin{aligned} u\#v \in L(M_u) &\leftrightarrow M_u \text{ started with } u\#v \text{ accepts} \\ &\leftrightarrow M \text{ started with } u\#v \text{ rejects} \\ &\leftrightarrow u\#v \notin L(M) \end{aligned}$$

Lemma 19. *Let*

$$n \leq s(n) = o(S(n))$$

and let $S(n)$ be space constructible. Then

$$SPACE(s(n)) \subsetneq SPAC3(S(n))$$

Proof. easy exercise

- For $u\#v$ with $n = |u\#v| \geq \max\{n_0, n_1, n_2\}$

$$
\begin{aligned}
u\#v \in L(M_u) \quad &\leftrightarrow \quad M_u \text{ started with } u\#v \text{ accepts} \\
&\leftrightarrow \quad M \text{ started with } u\#v \text{ rejects} \\
&\leftrightarrow \quad u\#v \notin L(M)
\end{aligned}
$$

Lemma 19. *Let*

$$n \leq s(n) = o(S(n))$$

and let $S(n)$ be space constructible. Then

$$SPACE(s(n)) \subsetneq SPAC3(S(n))$$

Proof. easy exercise

if resource bounds are not time resp. space constructible
the hierarchy theorems do NOT hold

Borodin gap theorem

later