

Lecture 1: Introduction

Lecturer: Kristoffer Arnsfelt Hansen

Scribe: Kasper Borup

Computational Model: 1-Tape Sequential Turing Maching

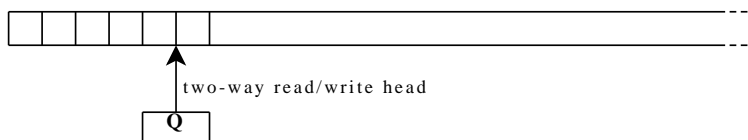


Figure 1:

- Q Finite set of states
- Σ Input alphabet
- Γ Tape alphabet
- δ Transition function
- $Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L\}$
- $s \in Q$ Start state
- $yes \in Q$ Accept state
- $no \in Q$ Reject state

The 1-tape sequential turing machine has one input/work tape which is finite to the left and infinite to the right. The transition function of the 1-tape sequential turing machine is a mathematical notion of the two-way read-write head of the turing maching - which for a symbol on the tape can move to the left or right and optionally choose to write another symbol on the current tape position.

Time Complexity

Let M be a Turing Machine.

Definition 1 $time_m(x) = \#steps$ M makes before halting on input x .

Definition 2 M is $T(n)$ time bounded if $time_m(x) \leq T(n), \forall x \in \Sigma^n$.

Crossing sequences

Definition 3 $C_i(x) = \{q_{i1}, q_{i2} \dots, q_{ik}\}$ - Sequence of states M is in on input $x \in \Sigma^*$, when crossing the line between cell i and $i + 1$, see figur 2.

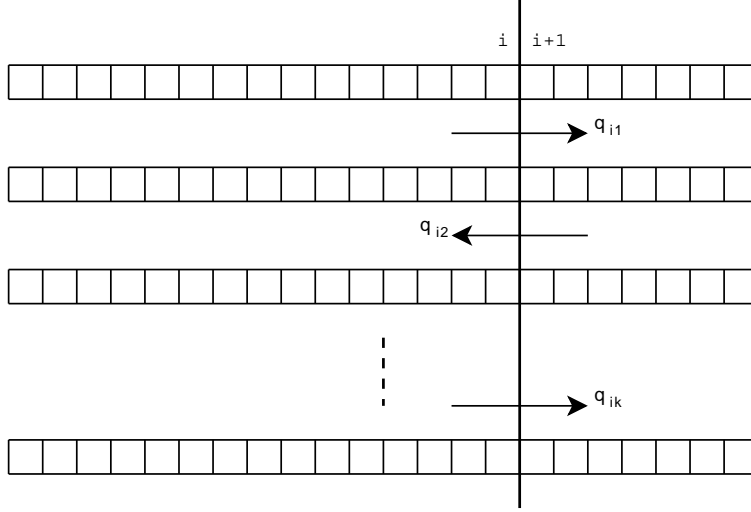


Figure 2: The states q_{i1}, \dots, q_{ik} that M is in when crossing cell i and $i + 1$, note that the first crossing is always a right crossing, and the next left and so on.

Lemma 4 Suppose M accepts $x = x_1x_2$ and $y = y_1y_2$, $|x_1| = i$ $|y_1| = j$. $C_i(x) = C_j(y)$, then M accepts x_1y_2 .

Proof The first crossing is always from left to right and note that because of the crossing sequence

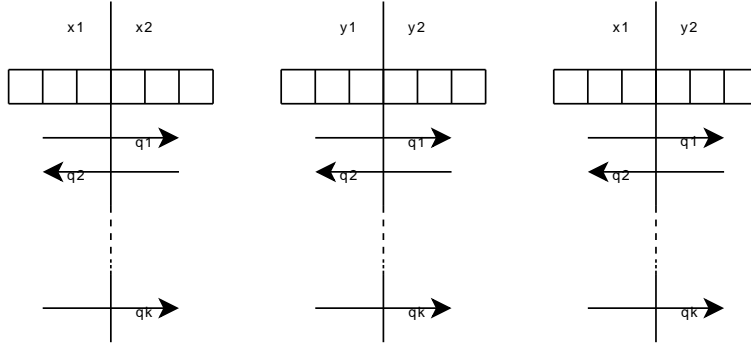


Figure 3:

$C_i(x) = C_i(x_1y_2) = C_j(y)$ we will always reenter and exit x_1 in the same state and with the same changes in both x_1x_2 and x_1y_2 . □

Language PAL_n

Definition 5 Given an n , let $k = \frac{n}{4}$, then $PAL_n = \{x\#^{n-2k}rev(x) \mid x \in \{0, 1\}^k\} \subseteq PAL$.

Lemma 6 Let $C(x) = \{C_i(x) \mid k \leq i \leq n - k\}$. If $x = x'\#^{n-2k}rev(x')$, $y = y'\#^{n-2k}rev(y') \in PAL_n$ where $|x| = |y|$ and $x \neq y$ then $C(x) \cap C(y) = \emptyset$.

Proof Assume that $C(x) \cap C(y) \neq \emptyset$, and note that $x \neq y \Rightarrow x' \neq y'$. Then there exists a $C_i(x) \in C(x)$ and $C_j(y) \in C(y)$ such that $C_i(x) = C_j(y)$. We know that

$$x = x'\#^{n-2k}rev(x'), y = y'\#^{n-2k}rev(y') \in PAL_n \subseteq PAL,$$

if we apply lemma 4, we now have that $z = x' \#^{n-2k+i-j} rev(y') \in PAL^1$. But now $z = x' \#^{n-2k+i-j} rev(y') \in PAL$ and $|x'| = |y'|$ implies $x' = y' \#$, which contradicts $x' \neq y'$. \square

Theorem 7 *Let M be a 1-tape sequential turing machine accepting the language PAL_n , with time complexity $time_m(x) \leq T(n)$. Then $T(n) = \Omega(n^2)$.*

Proof We know that $T(n) \geq time_m(x) \geq \sum_i |C_i(x)|$, because every time a boundary is crossed, at least one move is made.

Let $m_x = \min_{k \leq i \leq n-k} |C_i(x)|$, the length of the minimal crossing sequence C_i , where $i \in [k, n-k]$ and $x \in PAL_n$. Let $m = \max_x(m_x)$, the length of the largest of the minimal crossing sequences, $\forall x \in PAL_n$.

We want to show that $m = \Omega(n)$. Let $N = \text{"\#crossing sequences of size at most } m\text{"}$, that is

$$N = \sum_{j=0}^m |Q|^j = \frac{|Q|^{m+1} - 1}{|Q| - 1}.$$

Now "distinct elements in PAL_n " = $2^{\lfloor \frac{n}{4} \rfloor}$, so $N \geq 2^{\lfloor \frac{n}{4} \rfloor}$, because of the $\sum_{j=0}^m |Q|^j$ possible crossing sequences and because there is a minimal length crossing sequence for every $x \in PAL_n$ and from lemma 6 we know that no two strings in PAL_n shares these.

$$\begin{aligned} \frac{|Q|^{m+1} - 1}{|Q| - 1} &\geq 2^{\lfloor \frac{n}{4} \rfloor} \\ \log \left(\frac{|Q|^{m+1} - 1}{|Q| - 1} \right) &\geq \log \left(2^{\lfloor \frac{n}{4} \rfloor} \right) \\ \log (|Q|^{m+1} - 1) - \log (|Q| - 1) &\geq \lfloor \frac{n}{4} \rfloor \\ (m + 1) \log |Q| &\geq \lfloor \frac{n}{4} \rfloor \\ m &= \Omega(n) \end{aligned}$$

We now have that $m = \Omega(n) \Rightarrow T(n) \geq \sum_i m = \Omega(n^2)$.

\square

Computational Model: k -Tape Sequential Turing Maching

The k -tape turing machine is a five tuple $(Q, \Sigma, \Gamma, s, \delta)$, see figur 4, as before, which is the same as for the 1-tape turing machine except the transition function, which is extended to handle k tape heads so

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{R, L\}^k.$$

The language PAL is $time_m(x) = O(n)$, by copying the input to a second tape, and then making a scan comparing position $i = 1, 2, \dots, n$ on the input tape with position $j = n, n - 1, \dots, 1$ on the second tape. We have an algorithm working in $O(n)$ time.

¹We do not have $z \in PAL_n$ as $|z|$ can be different from n

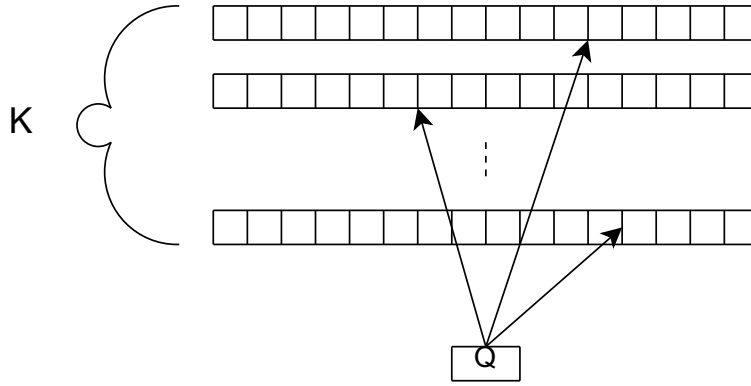


Figure 4:

Tape reduction

Theorem 8 (k -tapes \rightarrow 1-tape reduction) Suppose L is computed by a k -tape $T(n)$ time-bounded turing machine. Then L is computed by a 1-tape bounded turing machine, in time $O(T^2(n))$.

Proof We can simulate a k -tape turing machine M_1 by a 1-tape turing machine M_2 having k -tracks, one for each tape and k cells containing the head markers of that tape, see fig 5. A simulation of

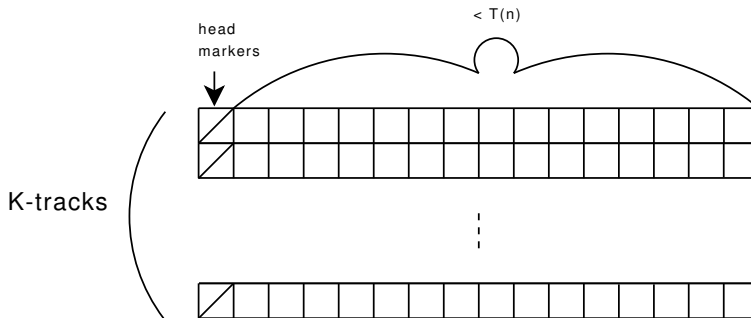


Figure 5: We simulate M_1 's k -tapes by packing them into 1-tape, in k -tracks, on M_2 , by enlarging the alphabet Σ of the new machine M_2 .

a move on M_1 is done sweeping the tape, visiting the head markers and record the symbol scanned by the head, after sweeping the tape M_2 can decide the move of M_1 which is performed in addition on the next sweep.

A sweep takes $O(kT(n)+k) = O(T(n))$ time, which results in the desired time bound $time_{M_2}(x) = O(T^2(n))$. Which is the best we can do as we have already shown that the language PAL is computed in $T(n) = \Omega(n^2)$ on a 1-tape sequential turing machine, and we know that we can compute PAL in $T(n) = O(n)$ on a k -tape turing machine, for $k \geq 2$. \square

Theorem 9 (k -tapes \rightarrow 2-tape reduction) Suppose L is computed by a k -tape $T(n)$ time-bounded turing machine. Then L is computed by a 2-tape $O(T(n) \log(T(n)))$ time-bounded turing machine.

Proof Sketch Let M be a two taped turing machine, where the tapes is divided into blocks off size $|B_i| = |B-i| = 2^{i-1}$ cells.

Tape 1 is a two-way infinite tape with 2 tracks for each of the k -tapes, see fig 6. The numbers

	b_{-3}	b_{-2}	b_{-1}	b_0	b_1	b_2	b_3
2-tracks	-5	-3	-1	/	1	3	5
	-6	-4	-2	0	2	4	6

Figure 6:

on the block parts indicates how the simulated tape is placed on the 2-tracks, only full parts is an actual part of the tape, the original tape is placed $\{\dots, -6, -5, -4-3-, 2-, -1, 0, 1, 2, 3, 4, 5, 6, \dots\}$, parts numbered < 0 is to the left of the current position of the tape head, and parts > 0 is to the right of the current position of the tape head.

Tape 2 is a scratch tape. The intuition behind the reduction is: move data not tape heads $- b_0$



holds the storage symbols scanned by each of the tape heads. When simulating a move data is then transported in the opposite direction of the move for the 2-tracks of the tape head being moved.

Invariant

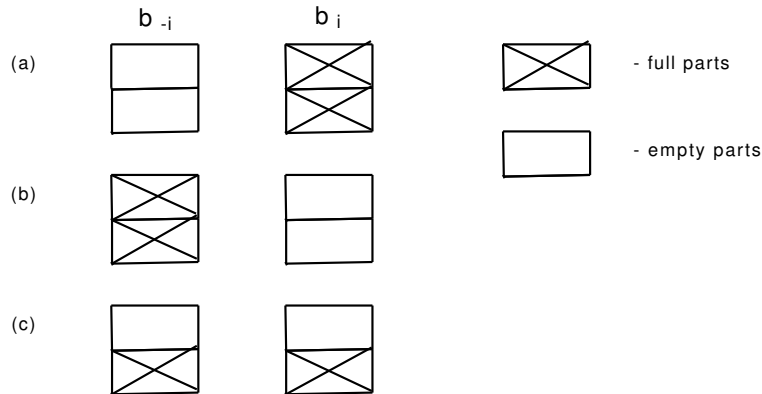


Figure 7: NB: a empty part is completely empty – containing a another blank symbol not in the alphabet of the k -tape turing machine.

Example: how to move data (right one step)

See figure 8 for reference:

- find first b_i not completely full.
- copy data at $b_0, b_1, \dots, b_{i-1} \rightarrow scratchtape$.
- copy back to lower parts of b_1, \dots, b_{i-1} and bottom/top most empty trunk of b_i .

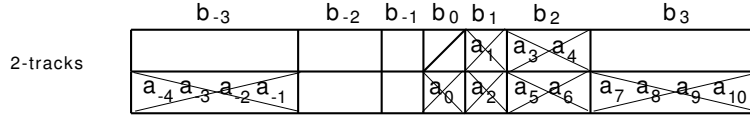


Figure 8: The a_i 's are the cells of the old machine that is contained in the blocks of M , where block b_i is of size 2^{i-1} cells of M .

See figure 9 for reference:

- goto b_{-i} and copy top/bottom most full part \rightarrow scratch tape.
- copy back to lower parts of $b_{-i+1}, b_{-i+2}, \dots, b_0$, note that b_{-i+1}, \dots, b_0 is empty because b_0, \dots, b_{i-1} was full because b_i was the first half full/empty block.

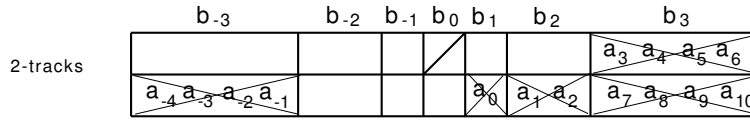


Figure 9:

See figure 10 for reference:

- The five steps executed above is called a b_i operation.
- A b_i operation is done a most every 2^{i-1} steps.
- A B_i operation can first be done after at least 2^{i-1} steps of computation. So there is at most $\frac{T(n)}{2^{i-1}}$ numbers of b_i -operations.
- Largest i used is $\log_2(T(n)) + 1$.
- Every b_i operation takes $O(2^i)$ time.

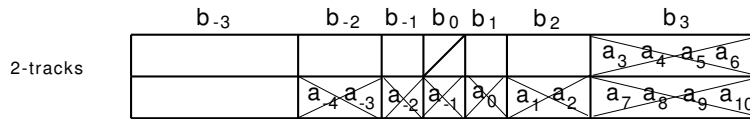


Figure 10:

The time complexity for the new machine M is:

$$\begin{aligned}
 time_m(x) &= O\left(\sum_{i=0}^{\log(T(n))+1} \frac{T(n)}{2^{i-1}} 2^i\right) \\
 &= O\left(T(n) \sum_{i=0}^{\log(T(n))+1} 2\right) \\
 &= O(T(n) \log(T(n)))
 \end{aligned}$$

□

Time complexity classes

Definition 10 $DTIME(T(n)) = \{ \text{Class of languages computed by a multitape } O(T(n)) \text{ time-bounded turing machine } \}$.

$$P = \bigcup_{k>0} DTIME(n^k)$$
$$EXP = \bigcup_{k>0} DTIME(2^{n^k})$$

Universal Turing Machine

- Takes a description of a Turing Machine M (and input) as input.
- Simulates a run of M .

U_k is defined as: has $k + 1$ tapes – put encoding of M , a k -tape turing machine on tape $k + 1$ – then U_k can simulate M on input x with a constant overhead.

If M is $T(n)$ time-bounded then U_k runs for $O(T(n))$ steps – where the constant only depends on M .

Time-constructible

Definition 11 $T(n)$ is time-constructible iff there is a turing machine that on input 1^n halts with $1^{T(n)}$ on a given tape in time $O(T(n))$.

Time Hierachy Theorem

Theorem 12 Let $T_2(n)$ be time-constructable, $T_1(n)$ fulfills $T_1(n) \log(T_1(n)) = o(T_2(n))$ and $T_1(n), T_2(n) \geq n$. Then

$$DTIME(T_1(n)) \subsetneq DTIME(T_2(n)),$$

meaning something that can be computed in $DTIME(T_2(n))$ cannot be computed in $DTIME(T_1(n))$.

Proof We want to prove that the time class $DTIME(T_2(n))$ is strictly larger than the time class $DTIME(T_1(n))$. We do this by constructing a machine M which cannot be in $DTIME(T_1(n))$, by diagonalization, and we then show that M is in $DTIME(T_2(n))$.

Contruction of M

For a input x , where $|x| = n$, we use the time-constructibility to put a clock $1^{T_2(n)}$ on the same tape, and then we run M for at most $T_2(n)$ steps.

We can write x on the form $x = 0^i 1 w$, for $i \in \mathbb{N}$, now M does:

- test if w is an encoding of a 2-tape turing machine M' , if not accept.
- simulate M' on input x .
- if M' halts then accept iff M' rejects (diagonalization).
- if time runs out, then accept.

Analysis

Assume $L(M)$ is computed by a $T_1(n)$ time-bounded turing machine. By 2-tape reduction we get a 2-tape TM M' , which is $O(T_1(n) \log(T_1(n)))$ time bounded, that computes $L(M)$.

Look at the simulation of M' with input $x = 0^i 1 w$, on M . Since $T_1(n) \log(T_1(n)) = o(T_2(n))$ the simulation of M' will succeed for all sufficiently large i . For these inputs the machines M and M' disagree, and hence M' cannot compute the language $L(M)$. \square

From this theorem we now directly get

$$P = \bigcup_{k>0} DTIME(n^k) \subseteq DTIME(2^n) \subsetneq DTIME(2^{n^2}) \subseteq EXP.$$

because $n2^n = o(2^{n^2})$.