Acta Informatica 14, 243–255 (1980)

Acta Informatica

On Alternation*

Wolfgang J. Paul, Ernst J. Prauß, Rüdiger Reischuk,
Fakultät für Mathematik, Universität Bielefeld, 4800 Bielefeld 1, Germany (Fed. Rep.)

Summary. Every alternating $t(n)$-time bounded multitape Turing machine can be simulated by an alternating $t(n)$-time bounded 1-tape Turing machine. Every nondeterministic $t(n)$-time bounded 1-tape Turing machine can be simulated by an alternating $(n+(t(n))^{1/2})$-time bounded 1-tape Turing machine. For wellbehaved functions $t(n)$ every nondeterministic $t(n)$-time bounded 1-tape Turing machine can be simulated by a deterministic $((n \log n)^{1/2} + (t(n))^{1/2})$-tape bounded off-line Turing machine. These results improve or extend results by Chandra-Stockmeyer, Lipton-Tarjan and Paterson.

1. Introduction

We investigate alternation. Alternation is at the same time a generalization of nondeterminism and a mechanism to model parallel computation. It was introduced independently in [3] and [10]. In 2 we repeat the basic definitions.

In the real world neither nondeterministic nor unbounded parallel machines exist, but although these machines are purely artificial constructs they help us to understand facts and to prove theorems about the real world. This is well known for nondeterministic machines. Similarly alternating machines have served to characterize the complexity of Presburger Arithmetic [1] and to derive lower bounds for the complexity of modal logic [5] and certain games [3]. Also they help to unify the theory and to design deterministic algorithms.

Exploring what can be done with the help of alternation we derive some simulation results. In [2] it was shown that every nondeterministic $t(n)$-time bounded multitape Turing machine (TM) can be simulated by a nondeterministic $t(n)$-time bounded 2-tape TM. In [3] the authors noted that the same result holds for alternating machines and that by the usual step by step simulation reduction to 1 tape can be achieved at the expense of squaring the time bound. Adapting the proof from [2] and exploiting in addition the power of alternation

* A preliminary version of this paper was presented at the 19th IEEE-FOCS

0001-5903/80/0014/0243/$02.60

we show in 3, that alternating multitape TM's can be simulated by alternating 1-tape TM's without loss of time.

In [13] it is shown that for $t(n) \geqq n^2$ every nondeterministic $t(n)$-time bounded 1-tape TM can be simulated by a deterministic $(t(n))^{1/2}$-tape bounded TM. In [11] the authors announced that every nondeterministic $t(n)$-time bounded 1-tape TM can be simulated by an alternating $(t(n))^c$-time bounded TM. This hints a simulation in the spirit of [9] and $c = 2/3$. In 4 we turn the divide-and-conquer simulation from [13] in a straight-forward way into a parallel simulation such that the space bound from [13] translates directly into a bound for alternating time, i.e. we obtain $c = 1/2$. Using moreover a lemma from [6] we can weaken the hypothesis $t(n) \geqq n^2$ in [13] to $t(n) \geqq n \log n$.

2. Definitions

One can describe all possible computations of a nondeterministic machine given some input as a tree: all nodes are instantaneous descriptions (ID's), the root is the initial ID of the machine for the given input and the sons of an ID c are exactly those ID's which can be reached from c in one move allowed by the transition relation. Leaves of the tree are the final configurations, they may be accepting or rejecting. Of course certain paths in the tree may be infinite.

(2.1) One can define an interior node of the tree to be accepting if at least one of its sons is accepting; the machine accepts the input iff the root is accepting.

Thus we may imagine the computation tree being built up level by level each time the machine makes a step and once there is an accepting leaf the information about this fact flows from that leaf to the root in a very simple way.

One may define a nondeterministic machine to be $t(n)$-time bounded if for every input of size n accepted by the machine there is an accepting node whose distance from the root is at most $t(n)$. All the above definitions are equivalent to the usual ones.

For *alternating machines* the mechanism for the flow of control from the leaves to the root is made more powerful. The states of a machine are partitioned into two classes: *existential states* and *universal states*. Depending on the class of a state ID's and hence nodes of computation trees may now be called existential or universal.

(2.2) An existential interior node is defined to be accepting if at least one of its sons is accepting, a universal interior node is defined to be accepting if all its sons are accepting.

An alternating machine M is defined to be *$t(n)$-time bounded* if for every accepted input w of length n the computation tree of M started with w stays accepting if it is pruned at depth $t(n)$. Clearly alternating machines without universal states are just nondeterministic machines.

For convenience denote by $ATIME_k(t(n))$ the set of languages accepted by alternating $O(t(n))$-time bounded k-tape TM's and by $ATIME(t(n))$ the union of

the classes $ATIME_k(t(n))$. In the same spirit we use the notations $NTIME_k(t(n))$, $NTIME(t(n))$, $DTIME_k(t(n))$, $DTIME(t(n))$ for nondeterministic and deterministic machines and $DTAPE(t(n))$ for determinisitc tape bounded machines. The latter notation remains meaningful for $t(n)<n$ if we refer to off-line machines.

3. Tape Reduction

It is well known that deterministic as well as nondeterministic 1-tape Turing machines cannot test quickly the equality of strings which are positioned somewhat apart on the tape [8]. This difficulty can be overcome by alternating machines.

Lemma 1. *Let A be a finite alphabet and $\# \notin A$ a marker.*
Let

$$L1 = \{x \# y \# z \mid x, y, z \in A^*, x = z\}.$$

Then

$$L1 \in ATIME_1(n).$$

Proof. We describe an algorithm for an alternating 1-tape TM.

Given input w with $|w| = n$ ($|w|$ denotes the length of w) accept if (3.1) to (3.3) hold.

(3.1) w has the format $x \# y \# z$, $x, y, z \in A^*$

(3.2) x is a prefix of z

(3.3) z is a prefix of x

(3.1) can be tested deterministically in time $O(n)$. We say how to test (3.2); (3.3) is tested in a similar way.

Use 4 tracks, copy the input on track 1.

Universally (i.e. in universal states) choose a tape square under x and print a marker b on track 2.

Then choose *existentially* (i.e. in existential states) a tape square under z and print a marker c on track 2. Accept iff (3.4) and (3.5) hold.

(3.4) The symbol above b equals the symbol above c.

(3.5) The distance d_b from the left end of x to b equals the distance d_c from the left end of z to c.

Clearly x is a prefix of z iff for all choices of the position for b there is a choice for the position of c such that (3.4) and (3.5) hold. (3.4) can be tested deterministically in time $O(n)$. It remains to show how to test (3.5).

Existentially guess on track 3 nonoverlapping binary representations m_1, $m_2, \ldots, m_t$ of natural numbers where $t \leqq n$ is arbitrary, such that $m_1 = 1$, m_1 stands under the first symbol of x and the first symbol of m_t stands under the last symbol of z. Similarly guess existentially on track 4 nonoverlapping binary

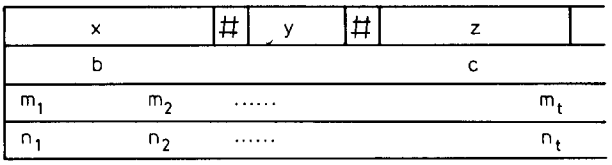

Fig. 1

representations $n_1, \ldots, n_t$ of numbers such that for all i the first symbol of n_i stands under the first symbol of m_i (see Fig. 1).

There will be subtrees of the computation tree, where the m_i and n_i have all length $O(\log n)$ and where the m_i are guessed at distance $O(n/\log n)$ from each other. Call such subtrees *good.*

Let $[m_i]$ denote the number represented by m_i. Choose universally $i \in \{2, \ldots, t\}$. Test deterministically if $[m_i] - [m_{i-1}]$ equals the distance between the left ends of m_i and m_{i-1} (this can be done by moving a counter over the distance $[m_i] - [m_{i-1}]$; in good subtrees the counter has length $O(\log n)$ and is moved over a distance $O(n/\log n)$), if not reject. Otherwise each m_i begins on cell $[m_i]$ of the tape inscription, thus the m_i may be used then as "milestones" to test (3.5):

Compute deterministically with the help of the milestones the binary representations of d_b and d_c. Compare each of these with some n_i nearby; if any of them differs from the n_i nearby reject. Otherwise choose universally $i \in \{2, \ldots, t\}$, test deterministically $n_i = n_{i-1}$, accept iff $n_i = n_{i-1}$. Clearly the part of the algorithm described last accepts iff (3.5) holds. Moreover if some input of length n is accepted, then any good accepting subtree of the computation tree has depth $O(n)$. □

Alternating 1-tape TM's can also quickly convert the binary representation of a number into its unary representation or vice versa. As the machines can guess the new representation and then verify the guess it suffices essentially to show

Lemma 2. *Let* $\# \notin A$ *be a marker and*

$$L2 = \{x \# y \mid x, y \in A^*, \ y \text{ is the binary representation of } |x|\}.$$

Then $L2 \in ATIME_1(n)$.

The *proof* is implicit in the last part of the proof of Lemma 1: guess the milestones below x, verify their correctness, compare the last one with y.

We remark that Lemma 1 gives us a fast way of copying a string (guess the copy and try to verify the correctness using Lemma 1) and Lemma 2 gives by the same method a fast way to determine the binary representation of the length of a string.

Theorem 1. $ATIME(t(n)) = ATIME_1(t(n))$ *for all* $t(n)$.

Proof. Let M be an alternating k-tape Turing machine with alphabet A and set of states Z. We describe an 1-tape Turing machine Q, which simulates M. A

path of length t in a computation tree of M defines in an obvious way a sequence of configurations $C=C_1,\ldots,C_t$ of M (this sequence looks like a computation of a conventional Turing machine). A *protocol* of this sequence of t *entries* $e_i \in Z \times A^k \times Z \times A^k \times \{\ell, r, 0\}^k$ where

$$e_i=(z(i), a_1(i),\ldots,a_k(i), z'(i), b_1(i),\ldots,b_k(i), s_1(i),\ldots,s_k(i))$$

iff in configuration $C_i M$ is in state $z(i)$, reads $a_1(i),\ldots,a_k(i)$, goes into state $z'(i)$, prints $b_1(i),\ldots,b_k(i)$ and makes the headmovements specified by $s_1(i),\ldots,s_k(i)$, where ℓ = left, r = right, 0 = no movement $(1 \leqq i \leqq t)$.

A sequence of entries e_i is a protocol of some sequence of configurations in the computation tree of M, iff (3.6) and (3.7) hold.

(3.6) the e_i are consistent with the transition relation of M.

(3.7) Let $p(i,v)=|\{u|s_v(u)=r,\ u<i\}|-|\{u|s_v(u)=\ell,\ u<i\}|$ = headposition on tape v before step i.

For all $i \in \{1,\ldots,t\}$ and all $v \in \{1,\ldots,k\}$ holds:
a) if i' is the largest number $<i$ such that $p(i,v)=p(i',v)$, then $a_v(i)=b_v(i')$.
b) if no such i' exists then

$$a_v(i)=\begin{cases}\text{the } p(i,v)\text{th symbol of the input if } v=1 \text{ and } 1 \leqq p(i,v) \leqq |\text{input}| \\ B \text{ otherwise}\end{cases}$$

In order to simulate M, Q first ignores the input and guesses successively a sequence of entries e_i with $z(i)=z'(i-1)$ for all i. For each i $a_1(i),\ldots,a_k(i)$ are guessed existentially. If $z(i)$ is an existential state, then M guesses the rest of e_i consistent with the transition relation of M existentially, otherwise M guesses the rest of e_i consistent with the transition relation of M universally.

Every time Q has guessed an entry e_i, it checks if $z'(i)$ is a final state. If $z'(i)$ is not a final state, Q generates one more entry. If $z'(i)$ is rejecting, Q rejects. If $z'(i)$ is accepting Q tries to verify (3.7) for the generated sequence of entries. If it fails it rejects, if it succeeds it accepts.

Assuming the test for (3.7) is correct and fast we show the correctness and derive the time bound for this part of the simulation. For an input w let T be the computation tree of Q.

Let T_1 be the part of T which consists of all paths from the root of T to nodes, where Q has just guessed an entry e_i such that $z'(i)$ is a final state and (3.7) holds for the generated sequence of entries.

The paths from the root of T to the leaves ℓ of T_1 correspond in a natural way to the finite paths from the root to leaves in the computation tree of M. Let T_2 be the subgraph of T consisting of T_1 and the (finite) subtrees of T whose roots are leaves of T_1. If T would be equal to T_2 then Q would accept the input iff M accepts the input. But all paths in T which do not belong to T_2 meet T_2 in existential nodes. Moreover all these paths are infinite or lead to rejecting leaves of T. Thus the paths in T outside T_2 have no influence on whether Q accepts the input or not.

If the input has length n and the input is accepted, then the computation tree

of M cut at depth $t(n)$ and evaluated by rule (2.2) is accepting. The protocols for this part of the computation tree of M have length $O(t(n))$ and are generated in $O(t(n))$ steps. The time bound follows if we show how to test (3.7) for a sequence of entries of length $O(t)$ in time $O(t)$.

This is done in the following way:

(3.8) On an extra track Q chooses existentially some entries $e_{i_1},\dots,e_{i_s}$ and marks them. Q also marks $e_{i_0}=e_1$.

(3.9) For each marked entry e_{i_j} Q guesses existentially the binary representation of numbers $q(i_j,1),\dots,q(i_j,k)$ and writes these down on k further extra tracks, beginning at e_{i_j}. These are candidates for head positions.

We say a subtree of the computation tree of Q is *good* if the marked entries e_{i_j} are at a distance of at most $O(t/\log t)$ and the numbers $q(i_j,v)$ have length $O(\log t)$. In good subtrees (3.9) contributes depth $O(t)$. In the sequel our estimates for the runtime Q will only hold for parts of the computation belonging to good subtrees, but the construction will be such that if Q accepts input w, then there is a good accepting subtree.

Next Q chooses universally one of the marked entries e_{i_j} as well as a number $v\in\{1,\dots,k\}$ and tests

(3.10)
$$\begin{aligned} &q(i_0,v)=0 \quad \text{if } j=0\\ &q(i_j,v)=q(i_{j-1},v)\\ &\qquad +|\{u\mid s_v(u)=r,\ i_{j-1}\leqq u<i_j\}\\ &\qquad -|\{u\mid s_v(u)=\ell,\ i_{j-1}\leqq u<i_j\}. \end{aligned}$$

This is done deterministically. In order to test (3.10) it is only necessary to move a counter of length $O(\log t)$ over a distance of length $O(t/\log t)$. If (3.10) does not hold, Q rejects. Thus in accepting subtrees (3.10) must hold for all j and v, i.e. the numbers $q(i_j,v)$ are the headpositions $p(i_j,v)$ as defined in (3.7).

Q chooses universally an entry e_i $(1\leqq i\leqq t)$ as well as a number $v\in\{1,\dots,k\}$. Then Q chooses existentially an entry $e_{i'}$ with $i'<i$ or a fictitious entry e_0. In the first case Q tests (3.7a), in the second (3.7b) and rejects if the outcome of the test is negative.

Obviously $a_v(i)=b_v(i')$ from (3.7a) can be tested in $O(t)$ steps. The correctness of (3.7b) is tested in the following way: by moving a counter Q determines $p(i,v)$ from a $q(i_j,v)$ nearby. As indicated in the remark after Lemma 2 Q determines the representation of the length n of the input and copies it on an extra track below the representation of $p(i,v)$. If $n<p(i,v)$ only $a_v(i)=B$ has to be checked. Otherwise Q chooses existentially a position ℓ of the input and erases symbols $\ell+1,\dots,n$ of the input. The binary representation of ℓ (=length of the rest of the input) is determined and copied below the representation of $p(i,v)$. If $\ell\neq p(i,v)$ Q rejects, otherwise Q checks if $a_v(i)$ equals the last symbol of the rest of the input. It remains to show, how to verify, that i' is the largest number $<i$ such that $p(i',v)=p(i,v)$ or (in the second case) that no such number exists.

Q chooses universally an entry $e_{i''}, i'<i''<i$ ($i''<i$ in the second case),

computes $p(i',v)$, $p(i'',v)$ and $p(i,v)$ - these can easily be computed from the numbers $q(i_j,v)$ - and tests $p(i',v)=p(i,v)+p(i'',v)$ using Lemma 1. If the outcome of this test is negative, Q rejects, otherwise (3.7) holds. □

We remark that with the same techniques a proof in [12] can be adapted to show, that every alternating $t(n)$-time bounded random access machine with logarithmic cost measure can be simulated by an alternating $t(n)$-time bounded TM.

4. Simulation of 1-Tape Machines

Let M be a nondeterministic 1-tape Turing machine with set of states Z, alphabet Σ, starting state q_0, accepting state q_1 and rejecting state q_2 (if there is one). B denotes the blank symbol. Let w be an input and $|w|=n$.
w.l.o.g. we may assume

(4.1) w stands originally on the tape squares $1,\ldots,n$, the head of M stands originally on square 0. M starts by going in state q_0 to square 1 and visits square 0 again only if it is in a final state and after erasing the inscription of the tape.

For a computation of M with input w the *(oriented) crossing sequence* (CS) between tape square i and $i+1$ is the string $c_1,\ldots,c_r$ over $\{\vec{S},\vec{S}\,|S\in Z\}$ where $c_j=\vec{S}(\vec{S})$ if the j'th crossing between the tape cells is from left to right (right to left) and immediately after the crossing the machine is in state S. As the first crossing is always from left to right, the second from right to left and so on, orienting a crossing sequence is unnecessary if only entire sequences are considered, but later we will chop up the sequences.

Lemma 3. *If M started with input w accepts w, but the computation produces two identical crossing sequences on tape squares to the right of the input, then there is an accepting computation of M started with input w, which uses less time and space.*

The *proof* is almost immediate from Fig. 2.

If the crossing sequences C and D are identical, then the computation which is obtained by deleting A' and pasting the remaining parts of the computation together at the common crossing sequence gives the desired computation.

Lemma 4. *For $t(n)\geqq n\log n$ every nondeterministic $t(n)$-time bounded 1-tape Turing machine M is $O(t(n)/\log t(n))$ tape bounded; moreover the time- and space-bounds are achieved on the same computations.*

Proof. Similar to [6], where this result is proven for deterministic machines:

Consider any minimum space accepting computation of a $t(n)$-time bounded 1-tape machine M started with some input w of length n. Let s be the space used by the computation, let z equal the number of states of M and for each i let a_i be the number of crossing sequences of length i generated on the originally blank part of the tape. Then $\Sigma\, i\, a_i\leqq t(n)$, $s-n=\Sigma\, a_i$ and for each i $a_i\leqq z^i$ by Lemma 3. Hence

$$t(n)/(s-n)\geqq(\Sigma\, i\, a_i/\Sigma\, a_i).$$

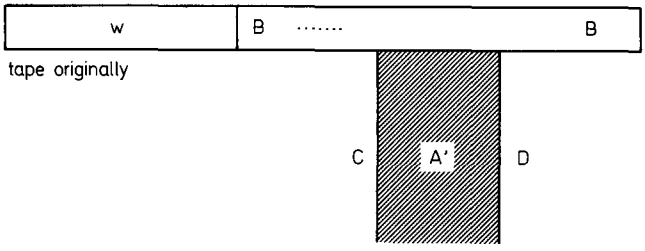

Fig. 2

The right hand side is minimal if there are as many short crossing sequences as possible. In this case it can be estimated from below by $\Omega(\log t(n))$ by an easy calculation. □

A *CS-stenogram* of a computation of M started with w is a string of the form $(\text{bin } r_1, C_1, \ldots, \text{bin } r_p, C_p)$[1], where for $1 \leqq i \leqq p$ C_i is the crossing sequence between the r_ith and the (r_i+1)th tape cell and $r_1 \leqq \ldots \leqq r_p$.

Lemma 5. *Let $t(n) \geqq n \log n$, $|w| = n$ and M accept w. Then there is a CS-stenogram of length $O(t^{1/2}(n))$ of an accepting computation of M started with input w, such that $C_p = \Lambda$ = the empty word (i.e. M never visits the tape square $r_p + 1$) and $r_i - r_{i-1} \leqq 2t^{1/2}(n)$ for $1 \leqq i \leqq p$ (take $r_0 = 0$).*

Proof. By Lemma 4 for input w there is an accepting computation of M of length $O(t(n))$ which uses only $O(t(n)/\log t(n))$ tape squares. Partition the tape squares used into $p = O(t^{1/2}(n)/\log t(n))$ blocks B_i each of length $t^{1/2}(n)$ and choose in each block B_i C_i as the shortest crossing sequence in this block. Then

$$t(n) \geqq \sum_{i=1}^{p} \sum_{\substack{S \text{ is a } CS \\ \text{in } B_i}} |S| \geqq \sum_{i=1}^{p} |C_i| t^{1/2}(n).$$

Also

$$\sum_{i=1}^{p} |\text{bin } r_i| \leqq \sum_{i=1}^{p} (\log t(n) + 1) = O(t^{1/2}(n)). \quad \square$$

Theorem 2.

$$NTIME_1(t(n)) \subseteq ATIME(n + t^{1/2}(n))$$

for all $t(n)$.

The *proof* is - except for the obvious modifications and some simplifications by Lemma 5 - almost copied from [13]. Let M be a nondeterministic 1-tape machine as described above.

A finite computation of M started with input w can be described as a *matrix* A with entries $\varepsilon \Sigma \cup (Z \times \Sigma)$. Here $a_{ij} = b$ means: before the i'th step of the computation square j of the tape contains symbol b.

[1] bin r denotes the binary representation of r

$a_{ij}=(z,b)$ means: before the i'th step of the computation M is in state z, the head visits square j and reads b. Consecutive rows correspond to subsequent ID's of M. Time and tape complexity correspond to the number of rows and columns of A. If A is a $t\times s$ - computation matrix and $0\leqq t_1\leqq t_2\leqq t$, $0\leqq s_1\leqq s_2\leqq s$, then the $(t_2-t_1)\times(s_2-s_1)$ - matrix A' with $a'_{ij}=a_{t_1+i,s_1+j}$ describes what happens in the computation from step t_1 to step t_2 between the s_1th and s_2th tape square. A' is called a *submatrix*. The *perimeter* of A' is the tupel (R_1,R_2,S_1,S_2), where R_1 and R_2 are the first and last rows of A' and $S_1(S_2)$ is the part of the crossing sequence between cell s_1-1 and s_1 (s_2 and s_2+1) which corresponds to crossings between step t_1 and step t_2. We say, that the predicate $\mathrm{comp}(R_1,R_2,S_1,S_2)$ holds, iff (R_1,R_2,S_1,S_2) is the perimeter of any submatrix of any computation matrix of M.

Proposition 6 (*formula C in* [11]).
$\mathrm{comp}(R_1,R_2,S_1,S_2)$ iff (4.2) or (4.3) hold:

(4.2) $\exists(R,S'_1,S''_1,S'_2,S''_2)$ *s.t.* (4.2a) *to* (4.2d) *hold.*

(4.2a) $$S'_1\,S''_1=S_1$$

(4.2b) $$S'_2\,S''_2=S_2$$

(4.2c) $$\mathrm{comp}(R_1,R,S'_1,S'_2)$$

(4.2d) $$\mathrm{comp}(R,R_2,S''_1,S''_2)$$

(4.3) $\exists(S,R'_1,R''_1,R'_2,R''_2)$ *s.t.* (4.3a) *to* (4.3d) *hold.*

(4.3a) $$R'_1\,R''_1=R_1$$

(4.3b) $$R'_2\,R''_2=R_2$$

(4.3c) $$\mathrm{comp}(R'_1,R'_2,S_1,S)$$

(4.3d) $$\mathrm{comp}(R''_1,R''_2,S,S_2).$$

Figure 3 illustrates this proposition. Clearly there are in general many possibilities for cutting a matrix.

If M is $t(n)$-time bounded, $|w|=n$ and (4.1) holds then M accepts w iff $\mathrm{comp}(R_1,R_2,S_1,S_2)$ holds for

$$R_1=w\,B^{c\cdot t(n)/\log t(n)-n}$$
$$R_2=B^{c\cdot t(n)/\log t(n)}$$
$$S_1=\tilde{q}_0\,\tilde{q}_1$$
$$S_2=\Lambda$$

where the constant c is chosen according to Lemma 4.

Now in order to prove Theorem 2 we may assume w.l.o.g. $t(n)\geqq n^2$, for if Theorem 2 holds for such $t(n)$, then it follows for all $t(n)$ by

$$NTIME_1(t(n))\subseteq NTIME_1(n^2+t(n))$$
$$\subseteq ATIME(n+(n^2+t(n))^{1/2})\subseteq ATIME(n+t^{1/2}(n)).$$

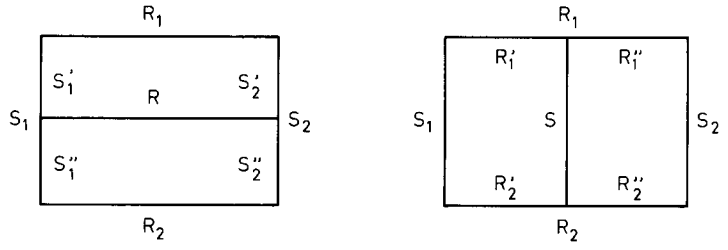

Fig. 3

We describe how an alternating machine AM simulates M such that AM is $O(t^{1/2}(n))$-time bounded if M is $t(n)$-time bounded. Given input w AM works in the following way:

(4.4) AM guesses existentially a CS-stenogram $(\text{bin}\, r_1, C_1, \ldots, \text{bin}\, r_p, C_p)$

(4.5) AM chooses universally a number $q \in \{1, \ldots, p\}$.

(4.6) Let $r_0 = 0$, $c_0 = \tilde{q}_0 \tilde{q}_1$ and $\tilde{w} = w B^{ct(n)/\log t(n) - n}$.

AM tests

$$\text{comp}(\tilde{w}_{r_{q-1}+1} \ldots \tilde{w}_{r_q}, B^{r_q - r_{q-1}}, C_{q-1}, C_q).$$

If the outcome of the test is positive, AM accepts, otherwise AM rejects. Thus AM accepts $w \Leftrightarrow$ there is a CS-steonogram such that comp (...) holds for all q $\Leftrightarrow M$ accepts w.

If M accepts w, then there is a CS-stenogram of length $O(t^{1/2}(n))$ of an accepting computation for w as in Lemma 5. AM can guess this CS-stenogram in $O(t^{1/2}(n))$ steps. Also observe that for this CS-stenogram the length of each perimeter (R_1, R_2, S_1, S_2) of the comp-clauses tested in (4.6) is bounded by $O(t^{1/2}(n))$.

Let $T(r, s, t)$ be the time in which AM can verify

$$\text{comp}(R_1, R_2, S_1, S_2) \quad \text{for } |R_1| = |R_2| = r, \quad |S| + |S_2| = s,$$

where (R_1, R_2, S_1, S_2) is the perimeter of a submatrix of at most t rows (i.e. at most t computation steps of M are performed "inside" the perimeter). In order to prove Theorem 2 it now suffices to prove

Lemma 7. *There is a constant k_1 such that for all r, s and t*

$$T(r, s, t) \leqq k_1(r + s + t^{1/2}).$$

Proof. In order to test $\text{comp}(R_1, R_2, S_1, S_2)$ AM works recursively. AM decides existentially to perform an R-cut or an S-cut or to verify by direct simulation of M.

R-cut. AM guesses existentially $R, S_1', S_1'', S_2', S_2''$ and verifies $|R| = |R_1|$, (4.2a) and (4.2b). This can be done in $k_2(r+s)$ steps. AM then chooses universally to test (4.2c) or (4.2d)

S-cut. AM guesses existentially $S, R_1', R_1'', R_2', R_2''$ and verifies (4.3a) and (4.3b).

This can be done in $k_2(r+s+|S|)$ steps. AM then chooses universally to test (4.3c) or (4.3d). In any accepting computation tree of M there will be an accepting subtree, where AM works in the following way.

(4.7) For small t AM verifies directly.

(4.8) If $r \leqq s$ AM makes an R-cut such that s is halved.

(4.9) If $r > s$ and $r+s < 7t^{1/2}$ AM makes an R-cut such that t is halved (if s or t are odd they can only be approximately halved; the resulting modifications in the analysis which follows are routine).

(4.10) Otherwise AM guesses in the middle third of R_1 a crossing sequence S(S-cut) such that $|S| \leqq 3t/r$ (such a crossing sequence exists, otherwise more than t computation steps belong to the perimeter (R_1, R_2, S_1, S_2)).

Now Lemma 7 is proven by induction: by choosing k_1 large enough it is made true for t small. Now we get

for (4.8):

$$\begin{aligned} T(r,s,t) &= k_2(r+s) + T(r, s/2, t) \\ &= k_2(r+s) + k_1(r+s/2+t^{1/2}) \\ &\leqq 2k_2 s - k_1 s/2 + k_1(r+s+t^{1/2}) \\ &\leqq k_1(r+s+t^{1/2}) \quad \text{for } k_1 \geqq 4k_2. \end{aligned}$$

for (4.9):

$$\begin{aligned} T(r,s,t) &= k_2(r+s) + T(r,s,(t/2)^{1/2}) \\ &\leqq k_2(r+s) + k_1(r+s+t^{1/2}/2^{1/2}) \\ &\leqq 7k_2 t^{1/2} - k_1 t^{1/2}(1-(1/2)^{1/2}) + k_1(r+s+t^{1/2}) \\ &\leqq k_1(r+s+t^{1/2}) \quad \text{for } k_1 \geqq 7k_2/(1-(1/2)^{1/2}). \end{aligned}$$

for (4.10):

$$\begin{aligned} T(r,s,t) &= k_2(r+s+3t/r) + T(2r/3, s+3t/r, t) \\ &\leqq k_2(2r+3t/r) + k_1(2r/3+s+3t/r+t^{1/2}). \end{aligned}$$

Now $s < r$ and $7t^{1/2} < r+s$ implies $49t/r < 4r$.

Thus

$$\begin{aligned} T(r,s,t) &\leqq k_2(2r+12r/49) + k_1 12r/49 + k_1(2r/3+s+t^{1/2}) \\ &\leqq r(110k_2/49 + 12k_1/49) + k_1(2r/3+s+t^{1/2}) \\ &\leqq k_1 r/3 + k_1(2r/3+s+t^{1/2}) \quad \text{for } k_1 \geqq 125k_2 \\ &= k_1(r+s+t^{1/2}). \quad \square \end{aligned}$$

Call $t(n)$ *well behaved* if the binary representation of $t(n)$ can be computed from the unary representation of n by a deterministic off-line TM using $t^{1/2}(n)$ tape cells

Theorem 3. *If $t(n)$ is well behaved, then*

$$NTIME_1(t(n)) \subseteq DTAPE((n \log n)^{1/2} + t^{1/2}(n))$$

for all well behaved $t(n)$.

Proof. For $t(n) \geqq n^2$ this follows of course from Theorem 2 and Theorem 3.1 in [3]. W.l.o.g. assume $t(n) \geqq n \log n$ and let M be a $t(n)$-time bounded nondeterministic 1-tape Turing machine as above. We describe a $t^{1/2}(n)$-tape bounded off-line Turing machine OM which simulates M.

Given an input w of length n OM first computes the binary representation of $t(n)$ and from this the binary representation of $t^{1/2}(n)$, OM lays off $c \cdot t^{1/2}(n)$ tape squares, where c is chosen such that for every accepting computation of M which has $t(n)$ steps there is a CS-protocol of length $c \cdot t^{1/2}(n)$ with the properties stated in Lemma 5. OM enumerates in lexicographical order all CS-protocols of this length. For every CS-protocol OM tests all the comp-clauses (4.6). OM is $O(t^{1/2}(n))$-tape bounded, if every comp-clause can be verified on $O(t^{1/2}(n))$ tape squares.

OM verifies $\text{comp}(R_1, R_2, S_1, S_2)$ by systematically enumerating R-cuts or S-cuts which are consistent with the rules (4.7) to (4.10). For each enumerated cut, the subproblems (4.2c), (4.2d) or (4.3c), (4.3d) are recursively treated one after another on the same portion of the tape. The recurrence equations for the tape used in this simulation are the same as the equations for the time used in the proof of Theorem 2. □

The above proof is of course nothing but the original proof from [13] simplified and extended to $t(n) < n^2$ by Lemma 5.

Corollary 1. *Every deterministic $t(n)$-time bounded 1-tape Turing machine M can be simulated by a deterministic $O((n \log n)^{1/2} + t^{1/2}(n))$-tape bounded off-line Turing machine M'.*

Proof. Let q_2 be the rejecting state of M. M accepts or rejects w after t steps iff for $i=1$ or $i=2$ (4.11) holds

$$\text{comp}(w B^{t-n}, B^t, \hat{q}_0 \hat{q}_i, \Lambda). \tag{4.11}$$

There is not by hypothesis given an easy way to compute $t(n)$ given n. Thus M' tries succesively for $t = |w|, |w|+1, \ldots$ to verify (4.11) for $i=1$ or $i=2$. This is done using the method described above. As M is $t(n)$-time bounded, the simulation succeeds for inputs of length n for $t = O(n + t(n)/\log t(n))$ or earlier. □

References

1. Bermann L (1977) Precise bound for Presburger arithmetic and the reals with addition. 18th IEEE-FOCS, 95-99
2. Book RV, Greibach SA, Wegbreit B (1970) Time and tape bounded Turing acceptors and AFL's. J Comput System Sci 4:606-621
3. Chandra A, Stockmeyer L (1976) Alternation, 17th IEEE-FOCS, 98-108

4. Erdös P, Graham R, Szemeredi E (1975) Sparse graphs with dense long paths. STAN-CS-75-504, Computer Science Dept, Stanford MA
5. Fischer MJ, Ladner RE (1977) Propositional modal logic of programs. 9th ACM-STOC, 286–294
6. Hartmanis J (1965) Size arguments in the study of computation speeds. Symp on Computers and Automata, Polytechnic Inst of Brooklyn, 1–18
7. Hartmanis J, Lewis PM, Stearns RE (1965) Hierarchies of memory limited computations. 6th IEEE Symposium on Switching Circuit Theory and Logical Design, 179–190
8. Hennie FC (1965) One-tape off-line Turing machine computations. Information and Control 8:553–578
9. Hopcroft JE, Paul WJ, Vailant LG (1977) On time versus space. J ACM 24:332–337
10. Kozen D (1976) On parallelism in Turing machines. 17th IEEE-FOCS, 89–97
11. Lipton R, Tarjan R (1977) Applications of a planar separator theorem. 18th IEEE-FOCS, 162–170
12. Monien B (1977) About the derivation languages of grammars and machines. Proc. 3rd ICALP, Lecture notes in Computer Science
13. Paterson M (1972) Tape bounds for time-bounded Turing machines. J Comput System Sci 6:116–124
14. Pippenger N, Fischer M (1979) Relations among complexity measures, J. ACM 26:361–381

Received February 14, 1979; Revised March 18, 1980