ON PARALLELISM IN TURING MACHINES*

Dexter Kozen

TR 76-282

June 1976

Department of Computer Science
Cornell University
Ithaca, New York 14853

* This research has been supported in part by National Science Foundation Grant DCR75-09433.

ON PARALLELISM IN TURING MACHINES*

Dexter Kozen
Department of Computer Science
Cornell University
Ithaca, New York 14853

Abstract

A model of parallel computation based on a generalization of nondeterminism in Turing machines is introduced. Complexity classes //T(n)-TIME, //L(n)-SPACE, //LOGSPACE, //PTIME, etc. are defined for these machines in a way analogous to T(n)-TIME, L(n)-SPACE, LOGSPACE, PTIME, etc. for deterministic machines. It is shown that, given appropriate honesty conditions,

$$
\begin{array}{ll}
L(n)\text{-SPACE} & \subseteq //L(n)^2\text{-TIME} \\
T(n)\text{-TIME} & \subseteq //\log T(n)\text{-SPACE} \\
//L(n)\text{-SPACE} & \subseteq \exp L(n)\text{-TIME} \\
//T(n)\text{-TIME} & \subseteq T(n)^2\text{-SPACE}
\end{array}
$$

thus

$$
\begin{array}{ll}
\vdots & \vdots \\
//\text{EXPTIME} & = \text{EXPSPACE} \\
//\text{PSPACE} & = \text{EXPTIME} \\
//\text{PTIME} & = \text{PSPACE} \\
//\text{LOGSPACE} & = \text{PTIME} \\
? & = \text{LOGSPACE}
\end{array}
$$

That is, the deterministic hierarchy LOGSPACE $\subseteq$ PTIME $\subseteq$ PSPACE $\subseteq$ EXPTIME $\subseteq$ ... shifts by exactly one level when parallelism is introduced.

We give a natural characterization of the polynomial time hierarchy of Stockmeyer and Meyer in terms of parallel machines. Analogous space hierarchies are defined and explored, and a generalization of Savitch's result NONDET-L(n)-SPACE $\subseteq$ L(n)^2-SPACE is given.

Parallel finite automata are defined, and it is shown that, although they accept only regular sets, in general 2^{2^k} states are necessary and sufficient to simulate a k-state parallel finite automaton deterministically.

Introduction

In this paper we introduce a conceptually simple yet powerful model of parallel computation based on the one-tape Turing machine. Our model, called a parallel Turing machine, is obtained by generalizing the notion of nondeterminism.

One usually thinks of a nondeterministic Tm as a machine with a single process which must make choices during a computation, and accepts provided some sequence of choices leads to an accepting state. Alternatively, one may think of a nondeterministic Tm as a machine with an unlimited number of processes and a Boolean value B_α associated with each machine configuration α. The machine starts with a single process in the starting configuration α_0; whenever a nondeterministic choice must be made, the process spawns several independent parallel processes, each of which follows one of the possible choices. When a process enters an accept configuration α_A, the value of B_{α_A} becomes 1; when it enters a reject configuration, 0. These values are then implicitly passed back up the computation tree, a Boolean "or" being computed at each choice configuration. The machine accepts provided B_{α_0}=1. Note that the computation tree and the B_α are not explicitly represented.

The preceding may be generalized by allowing Boolean and's as well as or's to be computed at choice configurations. This is done by associating with each state q of the finite control a Boolean connective $g_q \in \{\wedge,\vee\}$. If configurations $\beta_1 \ldots \beta_n$ follow from configuration α in one step, and q is the state of the finite control associated with configuration α, then when the computation is finished and the values of $B_{\beta_1} \ldots B_{\beta_n}$ have been determined, we take

$$
B_\alpha = \begin{cases} B_{\beta_1} \wedge \ldots \wedge B_{\beta_n} & \text{if } g_q = \wedge, \\ B_{\beta_1} \vee \ldots \vee B_{\beta_n} & \text{if } g_q = \vee. \end{cases}
$$

As before, the machine is said to accept provided B_{α_0}=1. Again, observe that the B_α and the computation tree are not explicitly represented; although one might argue that this gives an oversimplified and impractical model, it in fact allows us to extract the essential nature of parallelism without introducing messy structures for process communication.

Formal Definitions

A parallel Turing machine is defined exactly as a nondeterministic Tm with a two-way read-only input tape and two-way read-write work tape as in [1], except that in addition we include a function $g: K \to \{\wedge,\vee\}$ associating with each state q of the finite control K a Boolean connective $g_q \in \{\wedge,\vee\}$.

A machine configuration is defined to be the contents of the work tape, position of the input head, position of the work head, and current state. The start configuration is $\alpha_0 = \langle \epsilon,1,1,q_0 \rangle$, where q_0 is the start state. An accept (reject) configuration is any configuration of the form $\langle x,i,j,q_A(q_R)\rangle$, where q_A and q_R are accept and reject states. We assume WLOG that once a machine enters an accept or reject configuration, it stays there.

The binary relation $\vdash$ on configurations is defined from the transition function δ of the machine with respect to some input string $w \in \Sigma^*$, in the obvious way. E.g. if $w = w_1 \ldots w_n$ and

(a ,	right ,	left ,	p)
↑	↑	↑	↑
symbol to print on work tape	direction to move input head	direction to move work head	state to enter

* This research has been supported in part by National Science Foundation Grant DCR75-09433.

$$\in \delta(\underset{\substack{\uparrow\\ \text{input}\\ \text{symbol}\\ \text{read}}}{w_i},\ \underset{\substack{\uparrow\\ \text{work}\\ \text{symbol}\\ \text{read}}}{x_j},\ \underset{\substack{\uparrow\\ \text{current}\\ \text{state}}}{q})$$

then $<x_1 \ldots x_m,i,j,q> \vdash <x_1 \ldots x_{j-1}ax_{j+1} \ldots x_m, i+1,j-1,p>$. The reflexive transitive closure of $\vdash$ is denoted $\overset{*}{\vdash}$.

With each configuration α is associated a Boolean value $B_\alpha \in \{0,1\}$, defined with respect to an input w recursively as follows:

i) $B_\alpha=1$ if α is an accept configuration;

ii) $B_\alpha=0$ if α is a reject configuration;

iii) if q is not a final state and $\alpha = <x,i,j,q>$ then

$$B_\alpha = \bigwedge_{\alpha \vdash \beta} B_\beta \quad \text{if } g_q = \wedge,$$

$$B_\alpha = \bigvee_{\alpha \vdash \beta} B_\beta \quad \text{if } g_q = \vee.$$

iv) otherwise, B_α is undefined.

The machine is said to <u>accept w</u> provided $B_{\alpha_0}=1$.

If $\alpha = <x,i,j,q>$ and there are at least two $\vdash$-successors of α, then α is an <u>$\wedge$-branch</u> if $g_q = \wedge$, an <u>v-branch</u> if $g_q = \vee$.

It should be remarked that allowing $g_q = \neg$ does not gain us anything, since a process could remember in its finite control when it has seen a $\neg$, and thenceforth compute $\wedge$'s instead of v's and vice versa, and accept instead of rejecting and vice versa.

Relationships between time- and tape-bounded deterministic and parallel computations

Definition

A parallel Tm is <u>T(n)-time bounded</u> if every process enters an accept or reject state after at most T(n) steps on inputs of length n; it is <u>L(n)-tape bounded</u> if every process uses no more than L(n) tape on inputs of length n.

The following sets are parallel counterparts to the deterministic complexity classes T(n)-TIME, L(n)-TAPE, PTIME, etc.

//T(n)-TIME = {A|A is a set accepted by a T(n)-time bounded parallel Tm}

//L(n)-SPACE = {A|A is a set accepted by an L(n)-tape bounded parallel Tm}

$//\text{LOGSPACE} = \bigcup_{k=0}^{\infty} //k\log n\text{-SPACE}$

$//\text{PTIME} = \bigcup_{k=0}^{\infty} //n^k\text{-TIME}$

$//\text{PSPACE} = \bigcup_{k=0}^{\infty} //n^k\text{-SPACE}$

$//\text{EXPTIME} = \bigcup_{k=0}^{\infty} //2^{n^k}\text{-TIME}$

$//\text{EXPSPACE} = \bigcup_{k=0}^{\infty} //2^{n^k}\text{-SPACE}$

Theorem 1

If L is a function which can be constructed by a deterministic Tm running in time $O(L(n)^2)$, $L(n) \geq O(n)$, and M is an L(n)-tape bounded deterministic Tm, then M can be simulated by a parallel Tm M* running in time $O(L(n)^2)$.

Proof

Let Δ be an alphabet in which M's configurations can be encoded succinctly, $|\Delta| = k$. By judicious selection of Δ we can insure that each configuration of M on input w, $|w| = n$, can be written down on L(n) tape. Thus M has at most $k^{L(n)}$ configurations, and accepts w iff there is a sequence of configurations $\alpha_0 \vdash \ldots \vdash \alpha_A$ where α_A is an accept configuration.

Let M* have 3 tracks on its work tape and operate as follows:

(1) construct L(n) on the work tape.

(2) write down M's starting configuration α_0 on track 1.

(3) write $k^{L(n)}$ in k-ary on track 3.

(4) "guess" an accept configuration α_A and write it on track 3. This is done by starting at the left of tape and moving cell by cell across the tape; at each step, each process enters an v-branch, spawning k new processes, each of which writes a different symbol of Δ on track 3.

At this point there are $k^{L(n)}$ concurrent processes, the $i^{\underline{th}}$ of which contains

α_0
$k^{L(n)}$
β_i

on its work tape, where β_i is the $i^{\underline{th}}$ configuration of M.

(5) Process i determines whether β_i is an accept configuration, rejecting immediately if not.

(6) Process i now enters a procedure **A** which, if the tape contains

α
m
β

will determine whether $\alpha \overset{*}{\vdash} \beta$ in $\leq$ **m** steps, as follows:

(6.1) if m=0 then if $\alpha=\beta$ then accept else reject;
if m=1 then if $\alpha=\beta$ or $\alpha \vdash \beta$ then accept else reject;

(6.2) "guess" (using v-branching as in (4)) the middle configuration γ on the path $\alpha \overset{*}{\vdash} \beta$, and verify in parallel (using $\wedge$-branching) that $\alpha \overset{*}{\vdash} \gamma$ in $\leq m/2$ steps and $\gamma \overset{*}{\vdash} \beta$ in $\leq m/2$ steps, by calling A in parallel with arguments

α
$m/2$
γ

and

γ
$m/2$
β

The running time of A is given by the recurrence

$$F(0) = bL(n)$$
$$F(m) = cL(n) + F(m/2),\ b,\ c \text{ constants}$$

which has solution

$$F(m) = bL(n) + cL(n)\log m,$$

thus M*'s running time is given by

$$\begin{aligned} T(n) &= aL(n)^2 + F(k^{L(n)}) \\ &= O(L(n)^2) + cL(n)\log(k^{L(n)}) \\ &= O(L(n)^2). \end{aligned}$$

It is easy to prove that $T(M^*) = T(M)$. One shows that procedure A works as expected by showing that, if M* starts executing A in configuration α^* with tape

α
m
β

then $\alpha \overset{*}{\vdash} \beta$ in $\le m$ steps iff $B_{\alpha^*}=1$, by induction on m. Then by (1)-(5), for each guessed accept configuration α_i of M there is a process in configuration α^*_i of M* which will determine whether $\alpha_0 \overset{*}{\vdash} \alpha_i$ in $\le k^{L(n)}$ steps using A, thus M* accepts iff $\bigvee_i B_{\alpha^*_i}=1$ iff $\exists i\ \alpha_0 \overset{*}{\vdash} \alpha_i$ in $\le k^{L(n)}$ steps iff M accepts. ||

Simulating in the other direction, we get

Theorem 2

If T is a tape-constructible function, $T(n) \ge O(n)$, and M* is a parallel T(n)-time bounded machine, then M* can be simulated by a deterministic Tm M in $O(T(n)^2)$ space.

Proof

On input w, $|w|=n$, M* will compute for at most T(n) steps, hence its configurations, given a suitable encoding, will be at most T(n) in length. Thus M*'s computation may be represented by a tree of height $\le T(n)$ whose nodes are configurations of M*, β a son of α iff $\alpha \vdash \beta$, the start configuration α_0 at the root, and accept or reject configurations at the leaves.

M on input w first constructs T(n) and writes down the start configuration α_0, then builds and traverses the tree, calculating B_α at each node α in postorder, and erasing configurations it has already visited so that tape may be reused. M accepts iff the calculated value of B_{α_0} is 1.

At any point in the computation, if M is visiting node α, only α and its ancestors need appear on the tape. Since the tree is of height $\le T(n)$, at most $T(n)^2$ tape is needed. ||

Corollary 1

//PTIME=PTAPE and //EXPTIME=EXPTAPE.

The above results relate parallel time bounds to deterministic tape bounds. Similar results relate parallel tape bounds to deterministic time bounds.

Theorem 3

If L is a function which can be constructed deterministically in time $d^{L(n)}$ for some constant d, $L(n) \ge O(\log n)$, and M* is an L(n)-tape bounded parallel machine, then M* can be simulated by a deterministic Tm M in time $c^{L(n)}$ for some constant c.

Proof

Assume that on any input all processes of M* halt. This may be effected by attaching a counter to M* without loss of tape efficiency. Let $|\Delta|=k$ where Δ is M*'s configuration alphabet, as in Theorem 1. On input w, $|w|=n$, M operates as follows:

(1) construct L(n) on track 1.
(2) write down all configurations of M* side by side on track 1.
(3) on track 2 below the first symbol of each configuration, place a 1 if it is an accept configuration, 0 if a reject configuration, N if neither.

The above 3 steps take time $O(L(n)d^{L(n)})$ for some constant d.

Hereafter the contents of the cells beneath the first and second symbols of configuration α will be denoted $1^{\underline{st}}(\alpha)$ and $2^{\underline{nd}}(\alpha)$, respectively.

Repeat steps (4) and (5) $k^{L(n)}$ times:

(4) Repeat steps (4.1) through (4.3) for each α:
  (4.1) Find and mark all configurations $\beta_1 \ldots \beta_n$ such that $\alpha \vdash \beta_i$.
  (4.2) If there is only one such β, copy $1^{\underline{st}}(\beta)$ to $2^{\underline{nd}}(\alpha)$. Otherwise α is either an $\wedge$-branch or an $\vee$-branch. If α is an $\wedge$-branch then execute (4.2.1), else execute (4.2.2).
  (4.2.1) (α is an $\wedge$-branch)
    if $1^{\underline{st}}(\beta_i)=1$ for all β_i such that $\alpha \vdash \beta_i$ then enter 1 in $2^{\underline{nd}}(\alpha)$;
    if $1^{\underline{st}}(\beta_i)=1$ or 0 for all β_i such that $\alpha \vdash \beta_i$ and some $1^{\underline{st}}(\beta_i)=0$, then enter 0 in $2^{\underline{nd}}(\alpha)$;
    otherwise, enter N in $2^{\underline{nd}}(\alpha)$.
  (4.2.2) (α is an $\vee$-branch)
    if $1^{\underline{st}}(\beta_i)=0$ for all β_i such that $\alpha \vdash \beta_i$ then enter 0 in $2^{\underline{nd}}(\alpha)$;
    if $1^{\underline{st}}(\beta_i)=1$ or 0 for all β_i such that $\alpha \vdash \beta_i$ and some $1^{\underline{st}}(\beta_i)=1$, then enter 1 in $2^{\underline{nd}}(\alpha)$;

otherwise, enter N in $2^{\underline{nd}}(\alpha)$.

(4.3) erase all extraneous marks.

(5) move $2^{\underline{nd}}(\alpha)$ to $1^{\underline{st}}(\alpha)$ for all configurations α.

(6) if $1^{\underline{st}}(\alpha_0)=1$ then accept, else reject.

Step (4) takes $O(L(n)k^{L(n)})$ time for each configuration hence $O(L(n)^2k^{2L(n)})$ in all; step (5) takes $2L(n)k^{L(n)}$ time, hence steps (4) and (5) repeated $k^{L(n)}$ times takes $O(L(n)^2k^{3L(n)})$, hence the total running time of M on w is $\leq dk^{4L(n)}$ a.e., so take $c = dk^4$.

To see that M and M* accept the same set, observe that B_α is computed for each α in the computation tree starting at the leaves and working upward toward the root. If the subtree rooted at α has height h then the value of B_α may be found in $1^{\underline{st}}(\alpha)$ after h executions of steps (4) and (5). Since the entire tree is of height at most $k^{L(n)}$, $1^{\underline{st}}(\alpha_0)$ contains B_{α_0} when step (6) is executed. ||

We wish to prove a result simulating in the other direction. This will be done via two lemmas. Lemma 1 gives a parallel log-space bounded algorithm for a PTIME-complete set. Lemma 2 takes advantage of the logspace reduction from any set A ∈ PTIME to the complete set in order to construct a parallel log-space algorithm for A. The general result follows from a padding argument.

Definition

The circuit value problem (CVP) consists of a rooted dag $\mathscr{B}$ of out-degree ≤ 2, with 0 or 1 associated with each leaf, and ∧ or ∨ associated with each interior node. The value of a node is the Boolean value computed in the obvious way. The value of $\mathscr{B}$ is taken to be the value of the root. The set CVP is defined as

$$CVP = \{\mathscr{B} \mid value(\mathscr{B})=1\}.$$

As demonstrated in [2], CVP is complete for PTIME; i.e., CVP is recognized by a deterministic n^k-time bounded Tm, and every set in PTIME can be reduced to CVP via a deterministic log-space bounded computation.

Lemma 1

CVP ∈ //log(n)-SPACE.

Proof

Let parallel Tm M* operate as follows:

(1) verify that the input is of the proper format. Given a suitable encoding of directed graphs, this requires only log(n) tape and can be done deterministically.

(2) locate the root. Again, this can be done deterministically on log(n) tape.

(3) enter procedure A. This procedure, when entered with the input head pointing to vertex c, determines whether value(c)=1. A operates as follows:

(3.1) if c is a leaf with value 1 then accept.

(3.2) if c is a leaf with value 0 then reject.

(3.3) otherwise, c is an ∧-node or an ∨-node. Take an ∧-branch or ∨-branch accordingly, spawning two processes, one of which will locate the left son of c, the other the right son (requiring log(n) tape each), then both call A.

It is evident that no process uses more than log(n) tape, and an inductive argument shows that if A is entered with the input head pointing to c in configuration α then value(c) $=B_\alpha$. Thus value($\mathscr{B}$)$=B_{\alpha_0}$. ||

Lemma 2

If A ∈ PTIME then A is accepted by a parallel Tm M_A using O(log(n)) tape.

Proof

Let M_σ be a deterministic log-space bounded transducer computing σ, where σ is a reduction from A to CVP; i.e., x ∈ A iff $\sigma(x)$ ∈ CVP. Note that $|\sigma(x)| \leq |x|^k$ for some k, since M_σ is polynomially time bounded. We modify M_σ to be a subroutine of the M* of the previous lemma. The modified M_σ, when started with m in binary on its work tape, does not output any symbols, but instead writes the $m^{\underline{th}}$ symbol it would have output on track one of the work tape, and then returns. M_A is essentially M* modified to maintain a virtual input head position on a special track of the work tape. Initially this track contains 1. When M* would read a symbol from its input tape, M_A calls M_σ and reads the appropriate symbol from track one of its work tape. When M* would move its input head left or right, M_A subtracts or adds one, respectively, from the virtual input head position. Thus on input x, M_A simultaneously computes $\sigma(x)$ and runs M* on $\sigma(x)$. Moreover, M_A runs in O(log(n)) space, since M_σ and M* do, and the virtual input head track never gets longer than $\log(|\sigma(x)|) \leq k\log(n)$. ||

Theorem 4

If log(T(n)) is tape-constructible, T(n) ≥ O(n), and M is a deterministic T(n)-time bounded Tm, then M may be simulated by a parallel machine M* using tape O(log T(n)).

Proof

We have the result for $T(n) \leq O(n^2)$ by Lemma 2, so assume $T(n) > O(n^2)$. Let M' on input w, |w|=n,

(1) check that w ∈ Σ*#*, where # ∉ Σ.

(2) copy w to the work tape.

(3) if $w = x\#^k$, simulate M on x. For each step of M on x, erase one #.

If no more #'s are left, reject.

(4) accept iff M accepted x and at least one # is left.

Steps (1), (2), and (4) require $O(n)$ time. Step (3) requires at most $2n$ steps for every simulated step of M, and at most n steps are simulated, thus M' runs in time $O(n^2)$, and accepts the set $\{x\#^k | M \text{ accepts } x \text{ and } k > T(|x|)\}$. By lemma 2, there is a parallel log-space bounded machine $M^{*'}$ accepting this set. Now construct parallel machine M* which on input x simulates $M^{*'}$ on $x\#^{T(|x|)+1}$ as follows:

(1) construct $\log(T(n))$ and write $T(n)+1$ in binary on track 1 of the work tape. Track 1 will contain the position of the simulated input head of $M^{*'}$, when $M^{*'}$ would have tried to read #'s to the right of x.

(2) Simulate $M^{*'}$ on $x\#^{T(|x|)+1}$, using the remaining work tape tracks for $M^{*'}$s computation. When $M^{*'}$ would be scanning x, M* scans the same cell on its own input tape; when $M^{*'}$ would go beyond x, M* maintains the distance of the head from the right end of $M^{*'}$s simulated input tape on track 1. Whenever $M^{*'}$ would read the current input symbol in this situation, M* automatically supplies a #. Thus M* accepts x iff $M^{*'}$ accepts $x\#^{T(|x|)+1}$ iff M accepts x. Moreover M* uses tape $\log(T(n))$ for the simulated input head and $O(\log(|x\#^{T(|x|)+1}|)) = O(\log(T(n)))$ for the work tape of M^*. ||

<u>Corollary 2</u>

//LOGSPACE = PTIME and //PSPACE = EXPTIME.

The above results not only characterize the power of parallelism, but also reveal the striking relationship between Turing machine space and time, namely

$$
\begin{array}{lcl}
\vdots & & \vdots \\
\text{//EXPTIME} & = & \text{EXPSPACE} \\
\text{//PSPACE} & = & \text{EXPTIME} \\
\text{//PTIME} & = & \text{PSPACE} \\
\text{//LOGSPACE} & = & \text{PTIME} \\
? & = & \text{LOGSPACE}
\end{array}
$$

That is, the deterministic hierarchy LOGSPACE $\subseteq$ PTIME $\subseteq$ PSPACE $\subseteq$ EXPTIME $\subseteq \ldots$ shifts by exactly one level when parallelism is introduced. Observe that if one could show any implication of the form

PTIME = PSPACE $\rightarrow$ //PTIME = //PSPACE

for example, then a major open problem in computer science would be solved.

<u>A characterization of the polynomial time hierarchy</u>

In this section we give a useful characterization of the polynomial time hierarchy in terms of parallel machines.

<u>Definition</u>

Let M be a parallel Tm. We say a process (path in the computation tree) p on input w <u>alternates k times</u> if k is the largest number such that $p = \alpha_0 \vdash^* \alpha_1 \vdash^* \ldots \vdash^* \alpha_k \vdash^* \alpha_F$ where α_F is a final configuration, $\alpha_1 \ldots \alpha_k$ are branches, and α_i is an $\wedge$-branch iff α_{i+1} is an $\vee$-branch, for $1 \le i < k$; i.e., k is the number of alternations of $\wedge$- and $\vee$-branches in p.

<u>Definition</u>

A Σ^k-machine (Π^k-machine) is a parallel Tm in which each process alternates at most k times, starting with an $\vee(\wedge)$.

<u>Examples</u>

Σ^0- and Π^0-machines are deterministic Turing machines; a Σ^1-machine is a nondeterministic Turing machine.

<u>Definition</u>

$//\Sigma_p^k = \{T(M) | M$ is a polynomially time bounded Σ^k-machine$\}$

$//\Pi_p^k = \{T(M) | M$ is a polynomially time bounded Π^k-machine$\}$

<u>Examples</u>

$//\Sigma_p^0 = //\Pi_p^0 =$ PTIME

$//\Sigma_p^1 =$ NP

$//\Pi_p^1 =$ co-NP.

Let Σ_p^k and Π_p^k represent the $k^{\underline{th}}$ Σ and Π levels of the polynomial time hierarchy as defined by Stockmeyer.[3] The following theorem will perhaps aid in the placement of natural problems in this hierarchy.

<u>Theorem 5</u>

(i) $//\Sigma_p^k = \Sigma_p^k$

(ii) $//\Pi_p^k = \Pi_p^k$.

<u>Proof</u>

We will prove (i); a proof of (ii) is completely analogous.

As demonstrated in [3], $A \in \Sigma_p^k$ iff there is a polynomial p and a deterministic polynomially time bounded Tm M such that

$$A = \{x | \exists_p y_1 \forall_p y_2 \exists_p y_3 \cdots Q_p y_k \text{ [M accepts } x\#y_1\#\ldots\#y_k]\}$$

where $x \in \Sigma^*$, $y_i \in \Gamma^*$, and $\# \notin \Sigma \cup \Gamma$. In the above, $Q_p y_i$ is shorthand for $Qy_i |y_i| \le p(|x|)$.

To show $\Sigma_p^k \subseteq //\Sigma_p^k$, let A be as above. Let M* on input x use $\vee$-branching to guess y_1, $|y_1| \le p(|x|)$, and write down $x\#y_1$, then use $\wedge$-branching to write down $x\#y_1\#y_2$ for all possible y_2 with $|y_2| \le p(|x|)$, etc., until $x\#y_1\#y_2\#\ldots\#y_k$ is written down on the tape, then run M deterministically on $x\#y_1\#\ldots\#y_k$.

Then M^* is a polynomially time bounded Σ^k-machine accepting A, thus $A \in //\Sigma_p^k$.

To show $//\Sigma_p^k \subseteq \Sigma_p^k$, let M^* be an arbitrary Σ^k-machine with input alphabet Σ time bound p where p is a polynomial, and maximal degree of out-branching m (i.e. if $\alpha \vdash \beta_1, \ldots, \alpha \vdash \beta_n$ then $n \le m$). Construct deterministic Tm M with input alphabet $\Sigma \cup \Delta \cup \{\#\}$, where Σ, Δ, and $\{\#\}$ are pairwise disjoint and $|\Delta| = m$. On input w, M does the following:

(1) reject immediately if w is not of the form $x\#y_1 \ldots \#y_k$, with $x \in \Sigma^*$, $y_i \in \Delta^*$, and $|y_i| \le p(|x|)$.

(2) copy $y_1\#\ldots\#y_k$ onto track 1 of the work tape, padding any y_i of length $< p(|x|)$ out to length $p(|x|)$ with an arbitrary but fixed element of Δ.

(3) Simulate some process of M^* on x. The process to be simulated is determined by $y_1\#\ldots\#y_k$. When the first v-branch α of M^* is encountered, M takes the path determined by the first symbol a of y_1; i.e., if $\alpha \vdash \beta_1, \ldots, \alpha \vdash \beta_n$, $n \le m$, and a is the $i^{\underline{th}}$ symbol of Δ, then M simulates moving to configuration $\beta_{(i \bmod n)+1}$. The first symbol of y_1 is then erased. The next v-branch is determined by the second symbol of y_1, etc., until an $\wedge$-branch is encountered. Then the rest of y_1 is erased and subsequent $\wedge$-branches are determined by y_2, etc.

(4) accept (reject) if the simulated process ever enters an accept (reject) state.

Since M^* is p(n) time bounded, each y_i is long enough; since M^* alternates at most k times, there are enough y_i's.

Let $y \in \Delta^0 \cup \Delta^1 \cup \ldots \cup \Delta^{p(|x|)}$, and let αy denote the configuration of M^* obtained by starting in configuration α and taking the path specified by y, as outlined above. Note that $\forall_p y\ \alpha_0 y$ is an $\wedge$-branch or final configuration, and if α is an $\wedge$(v)-branch then αy is an v($\wedge$)-branch or final configuration, and no other configuration on the path from α to αy is an v($\wedge$)-branch.

Then

M^* accepts x iff $B_{\alpha_0} = 1$

iff $\exists_p y_1\ B_{\alpha_0 y_1} = 1$

iff $\exists_p y_1 [\alpha_0 y_1$ is an accept configuration or $\forall_p y_2\ B_{(\alpha_0 y_1) y_2} = 1]$

iff ...

iff $\exists_p y_1 [\alpha_0 y_1$ is an accept configuration or $\forall_p y_2 [(\alpha_0 y_1) y_2$ is an accept configuration or $\exists_p y_3 [\ \ldots$

$\vdots$

$Q_p y_k [(\ldots(\alpha_0 y_1) y_2)\ldots) y_k$ is an accept configuration$]\ldots]$

iff $\exists_p y_1\ \forall_p y_2 \ldots Q_p y_k [y_1\#y_2\#\ldots\#y_k$ determines a path from α_0 to an accept configuration of $M^*]$

iff $\exists_p y_1\ \forall_p y_2 \ldots Q_p y_k [M$ accepts $x\#y_1\#\ldots\#y_k]$. ||

The above theorem also gives us new complete problems for Σ_p^k and Π_p^k, namely $S_p^k(P_p^k) = \{\#\underline{M}\#\ \mathrm{code}(x)\#^{3|\underline{M}|m} | M$ is a $\Sigma^k(\Pi^k)$-machine accepting x in time m$\}$ where $\underline{M}$ is a suitable encoding of M. The construction is a straightforward generalization of the case for Σ_p^1 which appears in [4] (q.v. for definition of notation), once we observe that there is a universal parallel Tm which makes the same sequence of alternations as the machine it is simulating.

Other Hierarchies

By changing the resource bounds on Σ^k- and Π^k-machines, new hierarchies are obtained.

Definition

A $\underline{\Sigma^k_{T(n)\text{-TIME}}(\Pi^k_{T(n)\text{-TIME}})\text{-machine}}$ is a T(n)-time bounded $\Sigma^k(\Pi^k)$-machine, and $//\Sigma^k_{T(n)\text{-TIME}}(//\Pi^k_{T(n)\text{-TIME}}) = \{T(M) | M$ is a $\Sigma^k_{T(n)\text{-TIME}}(\Pi^k_{T(n)\text{-TIME}})$-machine$\}$. $\Sigma^k_{L(n)\text{-SPACE}}(\Pi^k_{L(n)\text{-SPACE}})$-machines and $//\Sigma^k_{L(n)\text{-SPACE}}$, $//\Pi^k_{L(n)\text{-SPACE}}$ are defined analogously.

Lemma 3

$$//\Sigma^k_{T(n)\text{-TIME}} = \text{co-}//\Pi^k_{T(n)\text{-TIME}} \text{ and}$$
$$//\Sigma^k_{L(n)\text{-SPACE}} = \text{co-}//\Pi^k_{L(n)\text{-SPACE}}.$$

Proof

Given a Σ^k-machine M, change accept (reject) states to reject (accept) states, and change v($\wedge$) states to $\wedge$(v) states. The resulting machine M' is a Π^k-machine with the same time and space bounds as M, accepting the complement of T(M). ||

Lemma 4

$$//\Sigma^k_{L(n)\text{-SPACE}} \cup //\Pi^k_{L(n)\text{-SPACE}} \subseteq //\Sigma^{k+1}_{L(n)\text{-SPACE}} \cap //\Pi^{k+1}_{L(n)\text{-SPACE}}$$

$$//\Sigma^k_{T(n)\text{-TIME}} \cup //\Pi^k_{T(n)\text{-TIME}} \subseteq //\Sigma^{k+1}_{T(n)\text{-TIME}} \cap //\Pi^{k+1}_{T(n)\text{-TIME}}.$$

Proof

All Σ^k-machines and Π^k-machines are both Σ^{k+1}-machines and Π^{k+1}-machines. ||

The following theorem on space hierarchies generalizes Savitch's result NONDET.-$L(n)$-SPACE $\subseteq L(n)^2$-SPACE.[5] It states that squaring the space bound allows you to move one step up in the hierarchy.

Theorem 6

Let L be a tape-constructible function, $L(n) \geq O(\log n)$. Then

(i) $//\Sigma^{k+1}_{L(n)\text{-SPACE}} \subseteq //\Sigma^k_{L(n)^2\text{-SPACE}}$, and

(ii) $//\Pi^{k+1}_{L(n)\text{-SPACE}} \subseteq //\Pi^k_{L(n)^2\text{-SPACE}}$.

Proof

The proof is by induction on k. The basis is provided by Savitch: a $\Sigma^1_{L(n)\text{-SPACE}}$-machine is a nondeterministic $L(n)$-space bounded Tm, thus may be simulated in $L(n)^2$ space deterministically, hence $//\Sigma^1_{L(n)\text{-SPACE}} \subseteq //\Sigma^0_{L(n)^2\text{-SPACE}}$. Also, using Lemma 3, $//\Pi^1_{L(n)\text{-SPACE}} = \text{co-}//\Sigma^1_{L(n)\text{-SPACE}} \subseteq \text{co-}L(n)^2\text{-SPACE} = L(n)^2\text{-SPACE} = //\Pi^0_{L(n)^2\text{-SPACE}}$.

Now suppose $//\Sigma^k_{L(n)\text{-SPACE}} \subseteq //\Sigma^{k-1}_{L(n)^2\text{-SPACE}}$ and $//\Pi^k_{L(n)\text{-SPACE}} \subseteq //\Pi^{k-1}_{L(n)^2\text{-SPACE}}$. Observe that every computation of a $\Sigma^{k+1}_{L(n)\text{-SPACE}}$-machine M is essentially a sequence of $\vee$-branches followed by several concurrent computations of a $\Pi^k_{L(n)\text{-SPACE}}$ machine M' started in different configurations. By the induction hypothesis, M' is simulated by a $\Pi^{k-1}_{L(n)^2\text{-SPACE}}$-machine M", thus M may be replaced by a $\Sigma^k_{L(n)^2\text{-SPACE}}$ machine which initially makes the same $\vee$-branches as M, but runs M" when the first $\wedge$-branch is encountered. Also, $//\Pi^{k+1}_{L(n)\text{-SPACE}} = \text{co-}//\Sigma^{k+1}_{L(n)\text{-SPACE}} \subseteq \text{co-}//\Sigma^k_{L(n)^2\text{-SPACE}} = //\Pi^k_{L(n)^2\text{-SPACE}}$. ||

Corollary 3

$$//\Sigma^k_{L(n)\text{-SPACE}} \cup //\Pi^k_{L(n)\text{-SPACE}} \subseteq L(n)^{2^k}\text{-SPACE}.$$

Proof

Induction on k, using the above theorem. ||

Corollary 4

$$//\Sigma^k_{L(n)\text{-SPACE}} \cup //\Pi^k_{L(n)\text{-SPACE}} \subseteq //L(n)^{2^{k+1}}\text{-TIME}.$$

Proof

Corollary 3 and Theorem 1. ||

It is evident that the logspace hierarchy defined by

$$//\Sigma^k_{\text{LOGSPACE}} = \bigcup_{c=0}^{\infty} //\Sigma^k_{c\log(n)\text{-SPACE}} = //\Sigma^k_{\log(n)\text{-SPACE}}$$

$$//\Pi^k_{\text{LOGSPACE}} = \bigcup_{c=0}^{\infty} //\Pi^k_{c\log(n)\text{-SPACE}} = //\Pi^k_{\log(n)\text{-SPACE}}$$

is analogous to the polynomial time hierarchy in many ways. Some of its properties are listed below:

(1) $//\Sigma^k_{\text{LOGSPACE}} \cup //\Pi^k_{\text{LOGSPACE}} \subseteq //\Sigma^{k+1}_{\text{LOGSPACE}} \cap //\Pi^{k+1}_{\text{LOGSPACE}}$ follows from Lemma 4;

(2) $//\Sigma^k_{\text{LOGSPACE}} \cup //\Pi^k_{\text{LOGSPACE}} \subseteq$ //PTIME follows from Corollary 2;

(3) $//\Sigma^k_{\text{LOGSPACE}} \cup //\Pi^k_{\text{LOGSPACE}} \subseteq (\log n)^{2^k}$-SPACE follows from Corollary 3;

(4) $S^k_{\text{LOG}}(P^k_{\text{LOG}}) = \{\#\underline{M}\#\text{code}(x)\#^m \mid M \text{ is a } \Sigma^k(\Pi^k)\text{-machine accepting } x \text{ on } \log(m) \text{ tape}\}$ is complete for $//\Sigma^k_{\text{LOGSPACE}}(//\Pi^k_{\text{LOGSPACE}})$, and $\bigcup_{k=0}^{\infty} S^k_{\text{LOG}} \cup P^k_{\text{LOG}}$ is complete for PTIME, in the same way that $\bigcup_{k=0}^{\infty} S^k_P \cup P^k_P$ is complete for PSPACE (proof is straightforward). This says that PTIME is the ω-jump of the logspace hierarchy, in the same way that PSPACE is the ω-jump of the ptime hierarchy (see [3]).

Parallel Finite Automata

In this section we further characterize the power of parallelism. A standard construction shows that a k-state nondeterministic finite automaton (f.a.) is simulated deterministically with a 2^k-state f.a. We define parallel f.a. in a natural way and show that, despite the fact that all parallel f.a. accept only regular sets, 2^{2^k} states are necessary and sufficient to simulate a k-state parallel f.a. deterministically.

Definition

A parallel f.a. is a 5-tuple

$P = \langle K, \Sigma, q_1, F, g \rangle$

where

K is a finite set of states $q_1 \dots q_k$,

Σ is a finite alphabet,

$q_1 \in K$ is the start state,

$F \subseteq K$ are the final states, and

$g: K \to (\Sigma \times 2^k \to 2)$ is a function associating with each state $q_i \in K$ a Boolean valued function $g(q_i) = g_i: \Sigma \times 2^k \to 2$. One can think of g_i as a function which, given some input symbol and a Boolean value associated with each of the k states, computes a new Boolean value to be associated with state q_i.

Let $\bar{x}$ denote a k-tuple of Boolean values $x_1 \dots x_k$, and let π_i denote the ith projection function $\lambda\bar{x}.x_i$.

Let χ = the characteristic vector of F, i.e.

$$\pi_i(\chi) = \begin{cases} 1 \text{ if } q_i \in F \\ 0 \text{ if } q_i \notin F. \end{cases}$$

Define $F_i: \Sigma^* \to (2^k \to 2)$, $1 \le i \le k$, inductively as follows:

$F_i(e) = \pi_i = \lambda\bar{x}.x_i$

$F_i(aw) = \lambda\bar{x}.g_i(a, F_1(w)(\bar{x}), \dots, F_k(w)(\bar{x}))$

where $a \in \Sigma$, $w \in \Sigma^*$.

$F_i(w)(\chi)$ is meant to correspond to the B_α of the previous sections. I.e., $F_i(e)(\chi) = 1$ iff q_i is an accept state, and if aw is the input remaining, a process in state q_i scans a and splits into k processes which run to completion, determining the values of $F_j(w)(\chi)$, $1 \le j \le k$, then $\lambda\bar{x}.g_i(a,\bar{x})$ is applied to these values to get $F_i(aw)(\chi)$. Again, there is no explicit machinery for computing the g_i or F_i. The following definition is the natural analog of $B_{\alpha_0} = 1$.

Definition

P accepts w provided $F_1(w)(\chi) = 1$.

The F_i are defined recursively "inside out". For technical reasons, we wish to define a similar function "outside in". Let

$G_i(e) = \pi_i = \lambda\bar{x}.x_i$

$G_i(wa) = \lambda\bar{x}.G_i(w)(g_1(a,\bar{x}), \dots, g_k(a,\bar{x}))$.

Lemma 5

$F_i = G_i, \quad 1 \le i \le k.$

Proof

By definition $F_i(e) = G_i(e) = \pi_i$. We have

$$(*)\ \lambda\bar{x}.G_i(wa)(F_1(y)(\bar{x}), \dots, F_k(y)(\bar{x}))$$
$$= \lambda\bar{x}.G_i(w)(g_1(a, F_1(y)(\bar{x}), \dots, F_k(y)(\bar{x})),$$
$$\vdots$$
$$g_k(a, F_1(y)(\bar{x}), \dots, F_k(y)(\bar{x}))$$
$$= \lambda\bar{x}.G_i(w)(F_1(ay)(\bar{x}), \dots, F_k(ay)(\bar{x})).$$

But

$$G_i(w) = \lambda\bar{x}.G_i(w)(\bar{x})$$
$$= \lambda\bar{x}.G_i(w)(F_1(e)(\bar{x}), \dots, F_k(e)(\bar{x})),$$

and applying (*) $|w|$ times we get

$$G_i(w) = \lambda\bar{x}.G_i(e)(F_1(w)(\bar{x}), \dots, F_k(w)(\bar{x}))$$
$$= \lambda\bar{x}.\pi_i(F_1(w)(\bar{x}), \dots, F_k(w)(\bar{x}))$$
$$= F_i(w). \quad ||$$

Theorem 7

Parallel f.a. accept regular sets.

Proof

Define $x \approx y$ iff $G_1(x) = G_1(y)$. Then $\approx$ is an equivalence relation which is

(i) of finite index, since there are only 2^{2^k} functions $2^k \to 2$;

(ii) right invariant, since if $x \approx y$ then

$G_1(x) = G_1(y)$, and $\forall a \in \Sigma$

$$G_1(xa) = \lambda\bar{x}.G_1(x)(g_1(a,\bar{x}), \dots, g_k(a,\bar{x}))$$
$$= \lambda\bar{x}.G_1(y)(g_1(a,\bar{x}), \dots, g_k(a,\bar{x}))$$
$$= G_1(ya), \text{ hence } xa \approx ya.$$

It remains to show that T(P) is a union of $\approx$-equivalence classes. Suppose x is accepted and $x \approx y$. Then by Lemma 5

$$F_1(x)(\chi) = 1 \Rightarrow G_1(x)(\chi) = 1$$
$$\Rightarrow G_1(y)(\chi) = 1$$
$$\Rightarrow F_1(y)(\chi) = 1,$$

thus y is accepted also. ||

By the above construction, we see that a Parallel f.a. can be simulated by a deterministic f.a. with 2^{2^k} states.

Theorem 8

2^{2^k} states are necessary, in general, to simulate a k-state parallel f.a. deterministically.

Proof

Let $\Sigma = \{a,b,c\}$. Consider a k-tuple in 2^k as a binary numeral between 0 and 2^k-1, inclusive. Define

$$g_i(a,\bar{x}) = \underline{ith} \text{ digit of } (\bar{x}-1) \bmod 2^k.$$

Let $\sim$ be the equivalence relation defined by $x \sim y$ iff $\forall w$(xw is accepted by $P \leftrightarrow yw$ is accepted by P). Then the $\sim$-equivalence classes give the minimal f.a. accepting $T(P)$.

Claim

$$x \approx y \text{ iff } x \sim y.$$

Proof of claim

Clearly $x \approx y \rightarrow x \sim y$. Now suppose $x \not\approx y$. Then $G_1(x) \neq G_1(y)$, hence $\exists p \in 2^k$ $G_1(x)(p) \neq G_1(y)(p)$. Let $n = (\chi - p) \bmod 2^k$. Then

$$\begin{aligned} G_1(xa^n)(\chi) &= G_1(xa^{n-1})(g_1(a,\chi),\ldots,g_k(a,\chi)) \\ &= G_1(xa^{n-1})((\chi-1) \bmod 2^k) \\ &= \ldots = G_1(x)((\chi-n) \bmod 2^k) \\ &= G_1(x)(p) \end{aligned}$$

and similarly $G_1(ya^n)(\chi) = G_1(y)(p)$ hence xa^n accepted $\leftrightarrow$ ya^n not accepted, thus $x \not\sim y$.

It remains to construct $g_1(b,\bar{x}),\ldots, g_k(b,\bar{x})$, $g_1(c,\bar{x}),\ldots, g_k(c,\bar{x})$ so that all $\approx$-equivalence classes are nonempty.

Take $g_i(b,\bar{x}) = \begin{cases} \underline{ith} \text{ digit of } \bar{x} & \text{if } \bar{x} \neq \bar{0} \text{ or } i \neq 1 \\ 1 & \text{otherwise} \end{cases}$

$g_i(c,\bar{x}) = \begin{cases} \underline{ith} \text{ digit of } \bar{x} & \text{if } (\bar{x} \neq \bar{0} \ \& \ \bar{x} \neq 2^{k-1}) \text{ or } i \neq 1 \\ \neg x_1 & \text{otherwise.} \end{cases}$

Then
$$\begin{aligned} (g_1(b,\bar{x}),\ldots,g_k(b,\bar{x})) &= \bar{x} \text{ if } \bar{x} \neq \bar{0}, \\ (g_1(b,\bar{0}),\ldots,g_k(b,\bar{0})) &= 2^{k-1} \\ (g_1(c,\bar{x}),\ldots,g_k(c,\bar{x})) &= \bar{x} \text{ if } \bar{x} \neq \bar{0} \text{ and } \bar{x} \neq 2^{k-1} \\ (g_1(c,\bar{0}),\ldots,g_k(c,\bar{0})) &= 2^{k-1} \\ (g_1(c,2^{k-1}),\ldots,g_k(c,2^{k-1})) &= \bar{0}. \end{aligned}$$

Now given an arbitrary $f: 2^k \rightarrow 2$, we construct $w_0 \ldots w_{2^{k-1}-1}$ so that $G_1(w) = f$, as follows:

for each i, $0 \le i < 2^{k-1}$, let w_i be given by

$w_i = e$
if $f(i) = 0$ and $f(2^{k-1}+i) = 1$

$w_i = a^{2^{k-1}-i} b a^{2^{k-1}} b a^i$
if $f(i) = 0$ and $f(2^{k-1}+i) = 0$

$w_i = a^{2^{k-1}-i} b a^{2^{k-1}+i}$
if $f(i) = 1$ and $f(2^{k-1}+i) = 1$

$w_i = a^{2^k-i} c a^i$
if $f(i) = 1$ and $f(2^{k-1}+i) = 0$.

Then $G_1(w) = f$. ||

References

1. J.E. Hopcroft and J.D. Ullman, Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, MA, 1969.
2. R.E. Ladner, "The Circuit Value Problem is Log Space Complete for P", SIGACT NEWS 7:1, January 1975.
3. L.J. Stockmeyer, "The Polynomial-time Hierarchy", Report RC5379, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1975.
4. J. Hartmanis and H.B. Hunt III, "The LBA Problem and its Importance in the Theory of Computing", SIAM-AMS Proc., vol. 7, Amer. Math. Soc., Providence, RI, 1974.
5. W.J. Savitch, "Relationships Between Nondeterministic and Deterministic Tape Complexities", J. Comput. System Sci. 4, 1970.