ON THE POWER OF RANDOM ACCESS MACHINES

Arnold Schönhage
Mathematisches Institut der Universität Tübingen, Germany

Abstract. We study the power of deterministic successor RAM's with extra instructions like $+,*,\div$ and the associated classes of problems decidable in polynomial time. Our main results are $NP \subseteq PTIME(+,*,\div)$ and $PTIME(+,*) \subseteq RP$, where RP denotes the class of problems randomly decidable (by probabilistic TM's) in polynomial time.

1. Random Access Machines

All RAM models considered here have countably many storage locations with addresses $0,1,2,\ldots,n,\ldots$ each of which can store a natural number denoted by $\langle n\rangle$, and an extra accumulator register with current contents z. Initially, $z = 0$ and $\langle n\rangle = 0$ for all n. For input and output the alphabet $\{0,1\}$ is used. The finite control is always given by a deterministic program written as a sequence of labels and instructions.

All RAM's will have the following common instructions:

goto λ ;	control is transferred to label λ ;
input λ_0,λ_1 ;	the next input symbol β is read and causes branching to λ_β ; if the input string is exhausted, the next instruction is performed;
output α ;	$\alpha \in \{0,1\}$ is output;
halt;	the RAM halts;
a-load n ;	$z := n$;
load n ;	$z := \langle n\rangle$;
i-load n ;	$z := \langle\langle n\rangle\rangle$;
store n ;	$\langle n\rangle := z$;
i-store n ;	$\langle\langle n\rangle\rangle := z$;
equal n ;	if $z = \langle n\rangle$ then $z := 0$ else $z := 1$;
skip;	the next instruction is skipped iff $z = 0$;
suc;	$z := z + 1$;

This list describes the class of the so-called successor RAM's, denoted by $\mathcal{M}_0$. Their relationship to Turing machines and storage modification machines is thoroughly investigated in [4]. Now we are interested in the additional power of RAM's having some extra instructions from the following list:

less	n ;	$<$	if $z < \langle n \rangle$ then $z := 0$ else $z := 1$;
add	n ;	$+$	$z := z + \langle n \rangle$;
sub	n ;	$\dot{-}$	$z := \max \{0, z - \langle n \rangle\}$;
mult	n ;	$*$	$z := z * \langle n \rangle$;
div	n ;	$\div$	$z := \lfloor z/\langle n \rangle \rfloor$; halt, if $\langle n \rangle = 0$;
shift	n ;	$\leftarrow$	$z := \lfloor z/2^{\langle n \rangle} \rfloor$;
and	n ;	$\wedge$	$z := z \wedge \langle n \rangle$, where both operands are considered as binary strings.

Observe that (in contrast to other authors in this field) we prefer to consider general comparisons like less as extra instructions, whereas the instruction equal belongs to the basic features of our RAM's, since it can be simulated in real-time by means of indirect addressing (see [4]).

2. Simulations

Let $\mathcal{M}(Q)$ denote the class of all RAM's with the common instructions (as listed in 1.) and the extra instructions belonging to Q; thus, for instance, $\mathcal{M}(+,\dot{-},*)$ contains all RAM's with the extra instructions add, sub, mult; and $\mathcal{M}(\emptyset) = \mathcal{M}_0$.

A RAM M_2 is said to simulate M_1 in polynomial (linear, real) time, denoted by $M_1 \xrightarrow{p} M_2$ ($M_1 \xrightarrow{l} M_2$, $M_1 \xrightarrow{r} M_2$), if there exists a constant c such that for every input $x = x_1 x_2 \ldots x_n$ the following holds: if x causes M_1 to read an input symbol, or to print an output symbol, or to halt at time steps $0 = s_0 < s_1 < \ldots < s_\ell$, respectively, then x will cause M_2 to act in the very same way with regard to those external instructions at time steps $0 = t_0 < t_1 < \ldots < t_\ell$, where $t_\ell \leq cs_\ell^c$ (rsp. $t_\ell \leq c \cdot s_\ell$ for linear time, $t_j - t_{j-1} \leq c(s_j - s_{j-1})$ for $1 \leq j \leq \ell$ in the real time case).

For machine classes $\mathcal{M}_1$, $\mathcal{M}_2$ we define polynomial reducibility $\mathcal{M}_1 \xrightarrow{p} \mathcal{M}_2$ by the condition that for each $M_1 \in \mathcal{M}_1$ there exists a machine $M_2 \in \mathcal{M}_2$ such that $M_1 \xrightarrow{p} M_2$. $\mathcal{M}_1 \xrightarrow{l} \mathcal{M}_2$ and $\mathcal{M}_1 \xrightarrow{r} \mathcal{M}_2$ are defined similarly. For the class $\mathcal{T}_1$ of all multitape Turing machines we know

$$\mathcal{T}_1 \xrightarrow{r} \mathcal{M}_0 \xrightarrow{r} \mathcal{M}(+,\dot{-},<,\wedge) \xrightarrow{p} \mathcal{T}_1 \ , \tag{2.1}$$

which yields the equivalences

$$\mathcal{T}_1 \xleftrightarrow{p} \mathcal{M}_0 \xleftrightarrow{p} \mathcal{M}(+,\dot{-},<,\wedge) \ . \tag{2.2}$$

On the other hand the speed-up theorem in [2, p. 63] shows that $\mathcal{M}(+) \xrightarrow{\ell} \mathcal{T}_1$ is not true.

Sometimes it will be possible to simulate an instruction directly without changing the internal representation of numbers; so we have $\mathcal{M}(<,F) \xrightarrow{r} \mathcal{M}(\dot{-},F)$ for arbitrary instruction sets F by observing that less n can be replaced by the instruction sequence suc; sub n; skip; a-load 1; in general, however, it is by no means clear whether $\mathcal{M}(Q_1) \xrightarrow{p} \mathcal{M}(Q_2)$ implies $\mathcal{M}(Q_1 \cup F) \xrightarrow{p} \mathcal{M}(Q_2 \cup F)$.

3. Polynomial classes

From the work on vector machines [1,3] it is known that

$$\mathrm{PSPACE}(\mathcal{T}_1) = \mathrm{PTIME}(\mathcal{M}(+,\dot{-},*,\wedge)) \ , \tag{3.1}$$

and that on this level non-determinism does not make any difference. Here we are mainly interested in classes below this level.

Theorem 1. $\mathrm{NPTIME}(\mathcal{T}_1) \subseteq \mathrm{PTIME}(\mathcal{M}(+,*,\leftarrow))$.

The proof is by constructing a machine $M \in \mathcal{M}(+,*,\leftarrow)$ which decides the satisfiability of propositional formulae in conjunctive normal form in polynomial time. Consider a conjunction $C \equiv D_1 \wedge D_2 \wedge \ldots \wedge D_m$ of disjunctions D_k in boolean variables $x_0, x_1, \ldots, x_{n-1}$ and their complements $\overline{x}_j$. We may assume that it is presented as a binary input of length L by some suitable encoding such that each atom x or $\overline{x}$ ocurring in one of the D's requires at least one bit. By thoroughly applying the or-and distributive law to $D_1 \wedge \ldots \wedge D_m$ a disjunctive expansion $C \equiv C_1 \vee C_2 \vee \ldots \vee C_R$ is obtained, i.e. the C_r are conjunctions in the x's and their complements. Assuming that D_k has d_k atoms and binary length L_k we get

$$R = \prod_k d_k \leq \prod_k L_k < \prod_k 2^{L_k} \leq 2^L \ . \tag{3.2}$$

Now the main idea is to simulate this expansion by means of the add-mult distributive law arithmetically.

After having read the input the machine M chooses minimal numbers β and λ such that

$$b = 2^\beta > m \ , \quad \ell = 2^\lambda \geq L \ . \tag{3.3}$$

For each disjunction D_k it computes an arithmetic representation

$$A_k = \sum_{j=0}^{n-1} (\xi_{k,j} 2^{\ell b^{2j}} + \eta_{k,j} 2^{\ell b^{2j+1}}) \ , \tag{3.4}$$

where the ξ's and η's are chosen as zero or one according to

$$\begin{aligned} \xi_{k,j} &= 1 \quad \text{iff} \quad x_j \ \text{occurs in} \ D_k \ , \\ \eta_{k,j} &= 1 \quad \text{iff} \quad \bar{x}_j \ \text{occurs in} \ D_k \ . \end{aligned} \tag{3.5}$$

This requires (by means of successive squaring) not more than $\lambda + 2n\beta = O(L \log L)$ multiplications and $O(L)$ other steps. Then $m-1$ further multiplications yield the product

$$B = A_1 * A_2 * \ldots * A_m = \sum_{p=0}^{b^{2n}-1} w_p 2^{\ell p} \ , \tag{3.6}$$

which is closely related to the disjunctive expansion mentioned above. By (3.2) and (3.3) we have

$$\sum_p w_p = R < 2^{\ell} \ . \tag{3.7}$$

If p has the b-ary expansion

$$p = \sum_{j=0}^{n-1} (u_{p,j} b^{2j} + v_{p,j} b^{2j+1}) \qquad (0 \le u_{p,j}, v_{p,j} < b) \ ,$$

then there are precisely w_p many conjunctions C_r which contain exactly $u_{p,j}$ times the variable x_j and $v_{p,j}$ times $\bar{x}_j$, so for $0 \le j < n$. Since C is satisfiable iff in the disjunctive expansion there is at least one C_r not containing x_j and $\bar{x}_j$ simultaneously for any j, we introduce the set of indices

$$S = \{p < b^{2n} \mid u_{p,j} v_{p,j} = 0 \ \text{ for all } \ j \}$$

and arrive at the criterion:

$$C \ \text{satisfiable} \quad \text{iff} \quad W = \sum_{p \in S} w_p > 0 \ . \tag{3.8}$$

In order to extract this sum from the expansion (3.6) we exploit the convolution property of multiplication by forming the product

$$G = B * F = \sum_s g_s 2^{\ell s} \qquad (0 \le g_s < 2^{\ell}) \tag{3.9}$$

where the "filter" number F is defined and computed as

$$F = \sum_{p \in S} 2^{\ell(nh-p)} = F_0 * F_1 * \ldots * F_{n-1} \ ,$$

with $h = b + b^2 + \ldots + b^{2n}$ and

$$F_j = 2^{\ell h} + \sum_{k=1}^{b-1} \left(2^{\ell(h-kb^{2j})} + 2^{\ell(h-kb^{2j+1})} \right) . \qquad (3.10)$$

By means of

$$2^{\ell(h-kb^q)} = \left(\prod_{\substack{t=0 \\ t \neq q}}^{2n-1} 2^{\ell b^{t+1}} \right) \cdot \left(2^{\ell b^q} \right)^{b-k}$$

these computations require at most $O(L^2)$ steps.
The binary expansion of G contains the test quantity $W = g_{nh}$ as a substring (see (3.9); inequality (3.7) excludes improper overflow). Therefore the machine finally computes

$$G' = \lfloor G/2^{nh\ell} \rfloor$$

and performs the test $W > 0$ by checking whether

$$\lfloor G'/2^{\ell} \rfloor * 2^{\ell} \neq G' .$$

This completes our proof of Theorem 1. The reader will have observed that it is only this final extraction of W, where the instruction shift ($\leftarrow$) is used. In order to compare the result of Theorem 1 with (3.1) we like to state several polynomial reducibilities among machine classes with different sets of extra instructions.

Theorem 2.

$$\begin{array}{ccccc} \mathcal{M}(+,\dot{-},*,\wedge) & \overset{p}{\underset{(1)}{\longleftrightarrow}} & \mathcal{M}(+,*,\wedge) & & \\ \uparrow{\scriptstyle p}\ (2) & & \uparrow{\scriptstyle p} & & \\ \mathcal{M}(+,\dot{-},*,\div) & \overset{p}{\underset{(3)}{\longleftrightarrow}} & \mathcal{M}(+,\dot{-},*,\leftarrow) & \overset{p}{\underset{(4)}{\longleftrightarrow}} & \mathcal{M}(+,*,\leftarrow) \end{array} .$$

Similar reductions have been given by others. One has to be very careful, however, since in our setting comparisons like less are not always available and indirect addressing cannot be eliminated in general.

By means of $\wedge$ and $\dot{-}$ all bitwise boolean operations can be implemented (after time t the masks $m_\ell = 2^\ell - 1$ can be computed for any $\ell \leq 2^t$ in time $O(t)$, even without $\dot{-}$ by use of $+$ and $*$ and $m_{2k} = m_k * (m_k + 2)$, $m_{2k+1} = 2m_{2k} + 1$). The proof of (1) is based upon the observation that by [$m_\ell \wedge u = u$ iff $u < 2^\ell$] the binary length of any number u can be determined economically. Now, for $u, v < 2^\ell$, proper subtraction $u \dot{-} v$ is achieved by computing $w = m_\ell \wedge ((m_\ell * v) + u)$, provided $v \leq u$. Otherwise we have $v > u$ and $u \dot{-} v = 0$. These two cases can be distinguished simply by checking whether $v + w = u$ holds.

The proof of (3) is by Newton iteration essentially, where $\leftarrow$ is used to furnish some kind of floating point arithmetic. So far direct simulations could be used, i.e. without changing the internal representation of numbers. For the reductions (2) and (4) we need a different technique which will be described in the next section; therefore we postpone the corresponding proofs.

We do not know whether the vertical arrows in Theorem 2 can be reversed (if so, then Theorem 1 would become a simple corollary of (3.1)). This question is closely related to the problem $NP \stackrel{?}{=} PSPACE$.

4. Further polynomial reducibilities

In view of Theorem 1 we are interested in the complexity levels below the second line of Theorem 2, i.e. we would like to answer the question-marks in

$$\mathcal{M}(+,*,\leftarrow) \xrightarrow[p]{?} \mathcal{M}(+,*) \xrightarrow[p]{?} \mathcal{M}_0 \ . \tag{4.1}$$

At first we discuss the second one. We consider straight line programs which begin with the instruction $x_0 := 0$ followed by computational steps of the form $x_k := x_i + 1$ or $x_k := x_i \ \underline{op} \ x_j$ (or $x_k := \underline{op}\ x_i$), where $0 \le i,j < k$, and $\underline{op}$ is any binary (or unary) operation in some finite set Q of extra instructions, so for $1 \le k \le n$; finally some index $r \le n$ is stated. (In this particular case we have to deal with $Q = \{+,*\}$). Such data are considered as generic descriptions for natural numbers like x_r or x_n . Let $COMP_{=}(+,*)$ or, more generally $COMP_{=}(Q)$ ($COMP_{\neq}(Q)$ resp.) denote the language of all such straight line programs (with given r) which fulfil $x_r = x_n$ ($x_r \neq x_n$, resp.). Beyond its own appeal the corresponding decision problem is linked to our previous discussion by

Theorem 3. $\mathcal{M}(Q_1) \xrightarrow{p} \mathcal{M}(Q_2)$ iff $COMP_{=}(Q_1) \in PTIME(\mathcal{M}(Q_2))$, in particular, $\mathcal{M}(+,*) \xrightarrow{p} \mathcal{M}_0$ iff $COMP_{=}(+,*) \in PTIME(\mathcal{M}_0)$.

A very similar result has been obtained by J. Simon ([5], Lemma 3.1). His proof, however, does not work in our case; it ignores the extra problems encountered with indirect addressing. We restrict our proof to the non-trivial part for the special case $Q_1 = \{+,*\}$, $Q_2 = \emptyset$, i.e. we assume that $COMP_{=}(+,*)$ is decidable in polynomial time by some machine $M_0 \in \mathcal{M}_0$ (or by some Turing machine, equivalently) and describe how to simulate any given machine $M \in \mathcal{M}(+,*)$ by a suitable $M' \in \mathcal{M}_0$ in polynomial time. (In view of (2.1) such an M' can be

designed as to have all the string handling capabilities of a multi-tape Turing machine.)

When simulating M the machine M' generates a straight line program P (with current length k) such that all numbers produced by M internally are described by some initial segment of P . In addition a "contents" function c with domain D is maintained dynamically. D contains indices of those variables in P which are used (or have been used) as the address of a storage location of M . More precisely, $c(p)=q$ means $\langle x_p\rangle = x_q$, and for $p,s \in D$, $p \neq s$ shall always imply $x_p \neq x_s$. Similarly, an extra index r will indicate that x_r is the current accumulator value of M .

Initially, M' creates P as the straight line program

$$x_0 := 0;\ x_1 := x_0 + 1;\ \ldots\ x_m := x_{m-1} + 1;$$

where m denotes the maximum of 1 and all n's that occur as an explicit address in the program of M , and sets $k := m+1$; $r := 0$; $D := \{0,1,\ldots,m\}$; $c(p) := 0$ for all $p \in D$. Then M' simulates the steps of M as follows (we omit goto, ..., halt, which cause no problems here): a-load n is simulated by $r := n$; load n by $r := c(n)$; store n by $c(n) := r$; equal n by [$q := c(n)$; if $x_r = x_q$ then $r := 0$ else $r := 1$]. In case of skip , M' skips the simulation of the next M-instruction iff $x_r = x_0$. With suc the straight line program P is prolonged by $x_k := x_r+1$; correspondingly for add n , or mult n , after $q := c(n)$, P is prolonged by $x_k := x_r+x_q$, or $x_k := x_r * x_q$ resp.; then in all three cases M' does $r := k$; $k := k+1$.

The crucial point is, of course, the simulation if indirect addressing, namely i-load n by
[$q := c(n)$; $r := 0$; for $p \in D$ do if $x_p = x_q$ then $r := c(p)$] ,
and i-store n by
[$q := c(n)$; for $p \in D$ do if $x_p = x_q$ then goto fin ;
$D := D \cup \{q\}$; $p := q$; fin: $c(p) := r$].

In order to perform the various comparisons $x_i \stackrel{?}{=} x_j$ used in the preceding simulations M' has to apply the decision procedure of M_o to corresponding segments of P . This will require a number of steps polynomial in the length of P , hence polynomial in the number of M-steps simulated, which (up to a constant) also bounds the current size of D . This completes our proof of Theorem 3.

For the subsequent discussion we need the following

Lemma. There exists a constant $c > 0$ such that for any numbers $x_r \neq x_n$ generated by some $(+,*)$-straight line program of length n there are more than $2^{cn}/cn$ numbers $m < 2^{cn}$ with $x_r \not\equiv x_n \bmod m$.

Proof. The number of primes $m < 2^{cn}$ not dividing $|x_r - x_n| < 2^{2^n}$ is at least $\pi(2^{cn}) - 2^n > 2^{cn}/cn$, so for all n with suitable $c > 0$.

As an immediate corollary we get

Theorem 4. $COMP_{\neq}(+,*) \in NP$.

The corresponding TM simply guesses an appropriate number $m < 2^{cn}$ and verifies $x_r \not\equiv x_m \bmod m$ (hence $x_r \neq x_m$) in polynomial time by executing the straight line program mod m.

We did not succeed in proving $COMP_{=}(+,*) \in NP$. This together with Theorem 4 would supplement Theorem 1 with the inclusion $PTIME(\mathcal{M}(+,*)) \subseteq NP$, as the proof of Theorem 3 shows. We are able, however, to establish an important relationship to the class RP of problems which are randomly decidable by probabilistic TM's in polynomial time (a paradigm for this type of problem is given in the pioneer paper [6]).

Theorem 5. For any $\varepsilon > 0$ there is a polynomial f such that for every $M \in \mathcal{M}(+,*)$ there exists a probabilistic Turing machine M' which simulates t steps of M in time $f(t)$ correctly with probability greater than $1 - \varepsilon$; therefore $PTIME(\mathcal{M}(+,*)) \subseteq RP$.

The proof is the same as for Theorem 3, except for the tests $x_i \stackrel{?}{=} x_j$. Here M' performs the k-th test, say $x_r \stackrel{?}{=} x_n$, by randomly choosing v_k many values of m, independently and equally distributed in $1 \leq m < 2^{cn}$. Then for each of these the n-th segment of the straight line program P is executed mod m; if $x_r \not\equiv x_n \bmod m$ is observed, then certainly $x_r \neq x_n$. Otherwise M' assumes $x_r = x_n$. By our lemma this assumption can be wrong only with probability less than

$$\left(1 - \frac{1}{cn}\right)^{v_k} < \exp(-v_k/cn) < \varepsilon \cdot 2^{-k},$$

if M' chooses $v_k = \lceil cn(k + \lg \frac{1}{\varepsilon}) \rceil$. Thus M' simulates M correctly with overall probability greater than $\prod_k (1 - \varepsilon/2^k) > 1 - \varepsilon$, and the running time of M' grows only polynomially in the number of M-steps (and in $\lg \frac{1}{\varepsilon}$).

Now we are able to appreciate the left-hand reduction in (4.1): if $\leftarrow$ could be eliminated at polynomial cost, then our Theorems 1 and 5 would imply $NP \subseteq RP$, a relationship of rather great significance. We see that the classes NP and RP are intimately related to the complexity levels induced by the extra instructions $+,*,\leftarrow,\wedge$.

Finally we have to append the proofs for (2) and (4) in Theorem 2, which will be based on Theorem 3 in its general form. With respect to (2) we consider a straight line program with instructions suc, +, $\dot{-}$, *, $\leftarrow$ and show that it can be simulated in polynomial time by a machine $M \in \mathcal{M}(+,\dot{-},*,\wedge)$ which represents each x_k by a pair (y_k,z_k) such that z_k is a power of 2 and $x_k z_k = y_k$ (a similar idea was used in [1]). M starts with $y_0 := 0$, $z_0 := 1$; then it simulates

$x_k := x_i + 1$ by $z_k := z_i$; $y_k := y_i + z_i$;

$x_k := x_i + x_j$ by $z_k := z_i * z_j$; $y_k := z_j * y_i + z_i * y_j$;

$x_k := x_i \dot{-} x_j$ by $z_k := z_i * z_j$; $y_k := z_j * y_i \dot{-} z_i * y_j$;

$x_k := x_i * x_j$ by $z_k := z_i * z_j$; $y_k := y_i * y_j$;

for $x_k := \lfloor x_i / 2^{x_j} \rfloor$ at first $x_j \geq 2^i$ is checked: if $2^i * z_j \dot{-} y_j = 0$ then $y_k := 0$; $z_k := 1$ (since $x_i < 2^{2^i}$) ; otherwise 2^{x_j} is determined by an extra computation within $O(i)$ steps and then $z_k := z_i * 2^{x_j}$; $y_k := (2^{2^k} \dot{-} z_k) \wedge y_i$.

Finally $x_r = x_n$ can be answered by checking whether $z_n * y_r = z_r * y_n$.
With respect to the reduction (4) we show at first that for numbers $u \neq v$, $u,v < 2^{2^t}$ the comparison $u < v$ can be achieved in the following way (by means of $+,*,\leftarrow$): $O(t)$ steps of binary search yield the unique w such that $u' = \lfloor u/2^w \rfloor \neq \lfloor v/2^w \rfloor = v'$, but $\lfloor u/2^{w+1} \rfloor = \lfloor v/2^{w+1} \rfloor$; then $u < v$ iff $u' + 1 = v'$. Now it is fairly obvious how to simulate a straight line program with instructions suc, +, $\dot{-}$, *, $\leftarrow$ by representing each x_k as a pair (y_k,z_k) with $x_k + z_k = y_k$, except for the simulation of $x_k := \lfloor x_i / 2^{x_j} \rfloor$ which is a bit more delicate. Again it suffices to deal with the case $x_j < 2^i$, i.e. $y_j < 2^i + z_j$. Then x_j and 2^{x_j} are determined explicitly by $O(i)$ steps of binary search. The naive $y_k := \lfloor y_i / 2^{x_j} \rfloor$, $z_k := \lfloor z_i / 2^{x_j} \rfloor$ (computed by means of $\leftarrow$) does not always yield the desired result. After $y' := y_k * 2^{x_j}$, $z' := z_k * 2^{x_j}$ the correction $z_k := z_k + 1$ has to be applied iff $y_i + z' < y' + z_i$.
Finally $x_r = x_n$ can be answered by checking whether $z_n + y_r = z_r + y_n$.

References

[1] J. Hartmanis and J. Simon, On the power of multiplication in random access machines. IEEE Conf. Rec. 15th Symp. Switching Automata Theory (1974) 13-23.

[2] J. E. Hopcroft, W. J. Paul and L. G. Valiant, On time versus space and related problems. Proc. 16th Ann. IEEE Symp. Foundations Comp. Sci., Berkeley (1975) 57-64.

[3] V. Pratt, L. Stockmeyer, M. O. Rabin, A characterization of the power of vector machines. Proc. 6th Ann. ACM Symp. Theor. Comp. (1974) 122-134.

[4] A. Schönhage, Storage modification machines. Preprint, Universität Tübingen (1978), submitted to SIAM J. Comput.

[5] J. Simon, On feasible numbers. Proc. 9th ACM Symp. Theor. Comp., Boulder (1977) 195-207.

[6] R. Solovay, V. Strassen, A fast Monte-Carlo test for primality. SIAM J. Comput. 6 (1977), 84-85.