Chapter 8

Stochastic Choices

April 24, 2002

We continue investigating the choice-mode of computation. This chapter focuses on the stochastic choices, viz., coin-tossing $\oplus$, probabilistic-and $\otimes$ and probabilistic-or $\oplus$. In contrast to alternation, we see that a rich theory arises when we restrict computational errors in stochastic computation.

In this chapter, it is essential to revert to the use of intervals when discussing valuations. For convenience, an appendix on basic probabilistic vocabulary is included.

8.1 Errors in Stochastic Computation

Two new phenomena arise with stochastic choices:

- Infinite loops in stochastic computation is an essential feature, rather than one we seek to eliminate (cf. alternation). This will be evident when we study space-bounded computations in section 3.
- An extremely rich theory arises from quantifying the forms of computational error. We think of error as a new computational resource.

Forms of computational error. Let M be a choice machine and suppose $Val_M(w) = [b, c]$ where w is an input word. If M accepts w (*i.e.*, $b > \frac{1}{2}$) then both $1 - b$ and $1 - c$ are useful measures of error. Since $\frac{1}{2} > 1-b \geq 1-c \geq 0$, we may call $1-b$ and $1-c$ (respectively) the **pessimistic acceptance error** and **optimistic acceptance error**. Similarly, if M rejects w, then b and c are (respectively) the **optimistic rejection error** and **pessimistic rejection error**. We have two basic paradigms for classifing errors: for optimistic errors, we say M has "zero error" if for all inputs, its optimistic error is 0. For pessimistic errors, we say M has "bounded-error" if its pessimistic errors are bounded away from $\frac{1}{2}$ by a positive constant.

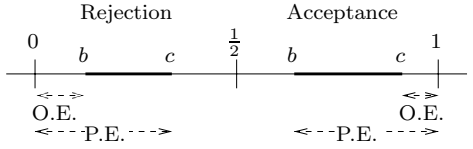


Figure 8.1: Forms of Error.

Stochastic computation has been studied since the early days of automata theory [9]. One original motivation[1] is the fundamental question of synthesizing reliable components from unreliable ones [22]. Stochastic computation in complexity theory started with the work of Gill [12].

Example 1 Gill was in turn motivated by some surprising probabilistic algorithms for primality testing algorithms due to Rabin [23] and Solovay-Strassen [33]. These primality testing algorithms share a common property: every computation path terminates and at each terminal configuration C,

[1] Modern computer hardware is remarkably reliable (compared to software). In the early days of computing, this reliability could not be taken for granted.

1) if C answers YES (i.e., claims that the input is prime) then there is a "small probability" that it made an error;

2) if C answers NO (i.e., claims that the input is composite) then it is surely correct.

The algorithm does not have YO-answers. It is not so easy to quantify the "small probability" of the YES answers. But we may rephrase this property, taking the viewpoint of the inputs:

1′) if the input is prime then all local answers are YES;

2′) if the input is composite then the local answer is NO with "high probability".

The reader should verify that 1′) and 2′) (the global perspective) are just reformulations of 1) and 2) (the local perspective). We elaborate on this because it is easy to become confused by a mixture of these two perspectives. The global formulation is more useful because we can explicitly quantify the "high probability" in it: it means with probability more than 3/4. We see that these primality algorithms make no errors whenever they (globally) accept.

> To remember which way the primality algorithms may have errors, it is helpful to know this: the algorithms answers NO (*i.e.*, claims that an input is composite) only if it finds a "witness" of compositeness. Hence NO-answers are never wrong by virtue of such witnesses. In contrast, the algorithm answers YES (*i.e.*, claims an input is prime) based on its failure to find a witness – such a YES answer could be wrong because an individual path does not exhaustively search for witnesses. In fact the algorithm looks for witnesses by randomly picking candidate witnesses and then checking each for the witness property. An example of a witness of the compositeness of a number n is a factor m $(1 < m < n)$ of n. It is easy to check if any proposed witness m is a factor. Unfortunately, there may be as few as two witnesses for some composite numbers; in this case, the method of randomly selecting candidate witnesses for testing is unreliable. The algorithms of Rabin and of Solovay-Strassen solve this by using more sophisticated concepts of "witnesses" where it can be shown that each composite number has a positive fraction of witnesses among the candidates, and a prime number has no witness.

This example will be used again to illustrate error concepts. ■

We now classify Turing machines according to their error properties.

Definition 1
(i) A non-empty interval g containing the value $\frac{1}{2}$ is called an *error gap*. Thus g has one of the forms

$$[a, b], \quad (a, b], \quad [a, b), \quad (a, b)$$

where $0 \leq a \leq \frac{1}{2} \leq b \leq 1$. We call a and b (respectively) the **lower** and **upper bounds** of g.
(ii) A choice machine M **accepts with error gap** g if for all accepted inputs w,

$$Val_M(w) \cap g = \emptyset.$$

Nothing is assumed if w is rejected or undecided. Similarly, we define **rejection with error gap** g. Finally, M **has error gap** g if for all input w, it accepts w or rejects with error gap g. Thus M is decisive.
(iii) We say M **accepts with bounded-error** if there exists e $(0 < e \leq \frac{1}{2}$ such that for all accepted inputs w, $Val_M(w) \cap [\frac{1}{2} - e, \frac{1}{2} + e] = \emptyset$. Similarly, M **rejects with bounded-error** if for all rejected inputs w, $Val_M(w) \cap [\frac{1}{2} - e, \frac{1}{2} + e] = \emptyset$. Also, M has **bounded-error** if for all inputs, $Val_M(w) \cap [\frac{1}{2} - e, \frac{1}{2} + e] = \emptyset$. Say M has **unbounded-error** if it does not have have bounded-error. ■

While bounded error focuses on pessimistic errors, the next set of definitions focus on optimistic errors.

Definition 2 (Continued)
(iv) We say M **accepts with zero-error** if for all accepted w, the upper bound of $Val_M(w)$ is 1. Similarly, M **rejects with zero-error** if for all rejected w, the lower bound of $Val_M(w)$ is 0. We say [2] M has **zero-error** if it is decisive, and it has zero-error acceptance and zero-error rejection.
(v) We combine bounded-error and zero-error: M has **bounded zero-error rejection** if it has bounded-error and zero-error rejection. Bounded zero-error rejection is also called **one-sided error**. By symmetry, we say M has **bounded zero-error acceptance** it has bounded-error and zero-error acceptance. Finally, M has **bounded zero-error** if it has bounded-error and zero-error. ■

[2] We emphasize that "zero-error" does not imply the absence of all errors: it only refers to the optimistic errors. It also seems that "errorless" would be preferable to "zero-error" in this set of terminology.

Let us briefly discuss the significance of these forms of error and their motivations. Although our error concepts treat acceptance and rejection with an even hand, we still favor acceptance when it comes to defining languages and complexity classes: for any machine M, the notation

$$L(M)$$

continues to refer to the language **accepted** by M. So a word $w \notin L(M)$ is either rejected or undecided by M.

Bounded Errors. Note that undecided intervals in INT are error gaps. Clearly a machine has error gap if and only it has the minimal error gap $[\frac{1}{2}, \frac{1}{2}]$, if and only if it is decisive. In this sense, bounded-error is a strengthening of halting deterministic computations (which are decisive). Students sometimes confuse the definition of "decisiveness" (a global condition) with the local condition that every path gives a YES or NO answer (and hence each path is finite). For instance, a decisive Turing machine could have infinite computation trees on each of its inputs.

To understand the significance of bounded-error, note that in general, acceptance and rejection errors can get arbitrarily close to $\frac{1}{2}$. This behavior is forbidden by bounded-error. The next section shows that with bounded-error we can modify a machine to yield error gaps of the form $[\epsilon, 1-\epsilon]$ for any desired constant $0 < \epsilon < \frac{1}{2}$, at a cost of increasing the computational time by a constant factor depending on ϵ. This yields a very important conclusion: *assuming we can tolerate constant factor slowdowns, bounded-error algorithms are practically as good as deterministic algorithms.* This is because any physical realization of a deterministic algorithm will still be subject to a small $\epsilon^* > 0$ probability of error, from quantum effects, manufacturing imperfections, etc. Thus, all real computer programs are bounded error algorithms.

Nondeterministic vs. probabilistic machines. Let N be a nondeterministic machine. The valuation function Val_N has values in $\{0, 1, \perp\}$. It follows that N accepts with no pessimistic error. Next, let us see what happens when N is converted into a probabilistic machine M, simply by declaring each state a toss-state. If N does not accept an input, M also does not accept. Hence we have

$$L(M) \subseteq L(N).$$

A further modification to M yields a probabilistic machine M' that accepts the same language as N: let M' begin by tossing a coin in the start state, and on one outcome it immediately answers YES, and with the other outcome it simulates N. So M' is undecided on input $w \notin L(N)$, since the lower bound of $Val_{M'}(w)$ is exactly $\frac{1}{2}$. This shows

$$NTIME(t) \subseteq PrTIME(t+1). \tag{1}$$

A simple modification to M' will ensure that it has zero-error acceptance: simply make all terminal which answer NO to answer YO instead. Notice that N, M and M' are not decisive in general.

Zero-error computation. The primality testing algorithms above accept with zero-error. The concept of zero-error is best understood in terms of stochastic machines with no negation states: in this case acceptance with zero-error means that if an input is accepted then no computation path leads to a NO-configuration. Similarly, "rejecting with zero-error" means that if an input is rejected, then no computation path leads to a YES-configuration. In either case, YO-configuration (or looping) is not precluded. Because of monotonicity properties, if a complete computation tree $T_M(w)$ accepts with zero-error then any prefix T' of $T_M(w)$ also accepts with zero-error, if T' accepts at all.

One-sided error. One-sided error is also motivated by the primality algorithms. These algorithms have no pessimistic errors on prime inputs, by virtue of property $1'$), and have bounded error on composite inputs, by property $2'$). Thus such an algorithm has bounded zero-error acceptance. Now with a trivial change, we can regard the same algorithm as a recognizer of composite numbers: it answers YES iff it finds a witness of compositeness. This new algorithm has bounded zero-error rejection, i.e., it has one-sided error.

This begs the question as to why we define one-sided error in favor of "zero-error rejection" over "zero-error acceptance". We suggest that the (non-mathematical) reason has to do with our bias towards nondeterministic machines: *a probabilistic machine M with one-sided error can be regarded as a nondeterministic machine N.* Let us clarify this remark. For, if M accepts w then some terminal configuration gives a YES-answer, and so N accepts w. Conversely, if M does not accept w, there is no YES-configuration because M has zero-error rejection. Hence, N does not accept w. We conclude that $L(M) = L(N)$. This proves

$$PrTIME_1(t) \subseteq NTIME(t). \tag{2}$$

The subscript "1" in "$PrTIME_1(t)$" refers to one-sided error (the general convention is described below).

The literature often discuss errors in the context of probabilistic machines that run in time $t(n)$ for some time-constructible t (e.g., t is a polynomial). In this case, we can simplify. For instance, we need not deal with intervals: if a stochastic machine does not terminate within t steps, we simply answer NO. This avoids the value $\perp$ and the pessimistic and optimistic errors coincide. This approach does not work in space-bounded computations, for instance.

Error-Restricted Complexity. Since error is viewed as another computational resource, we combine error bounds with other resource bounds. The number of logical possibilities is large, but happily, only a few forms are important. For instance, we exclusively use constant gap functions in complexity characteristics.

The following illustrates the combination of acceptance time with bounded-error or with zero-error:

Definition 3 Let t be a complexity function. A choice machine M **accepts in time t with bounded-error** if there is an error gap $g = [a, b]$, $a < \frac{1}{2} < b$ such that for all accepted inputs w, there is an accepting computation tree T such that $Val_T(w) \cap g = \emptyset$ and each path in T has length at most $t(|w|)$. ■

Note that the time bound and error gap are simultaneously achieved in a single computation tree T. To appreciate this, suppose another computation tree T' of height $t' < t$ is a prefix of T. Because of monotonicity, we have $Val_{T'}(w) \sqsubseteq Val_T(w)$. So it is possible that $Val_{T'}(w) \cap g$ is non-empty and yet T' is accepting. So M accepts in time t' but not necessary with the same gap g. In general, all complexity characteristics we specify for a machine are assume to be simultaneous unless otherwise stated.

We can add any of the error restrictions (bounded-error, zero-error, one-sided error, etc) to any of the usual complexity resource (time, space, reversal, simultaneous time-space, etc) bounds. Thus we can speak of M accepting in polynomial space with one-sided error.

We can also extend acceptance complexity to "rejection complexity" and to "running complexity". For rejection complexity, just replace acceptance by rejection. For running complexity, we make decisiveness a pre-requisite. Then a running complexity bound is simply a common bound on both accepting complexity and rejecting complexity.

Notation for error-restricted complexity classes. For simplity, we assume that *the machines used in defining error-restricted classes must be decisive and that running complexity is used.* However, we do not assume our machines to be halting (*i.e.*, halts on all computation paths). A machine can be decisive without being halting; conversely, it can be halting without being decisive. Indeed, in space bounded computation, halting is not necessarily desirable. To refer to such classes, it is sufficient to augment our previous convention for complexity classes, simply by introducing new subscripts. Until now, we have only one kind of subscript, 'r', denoting running complexity. We now introduce three new subscript z,

$$z \in \{b, 0, 1\},$$

to indicate the following restrictions: *bounded-error* ($z = b$) or *one-sided error* ($z = 1$) or *bounded zero-error* ($z = 0$). Note a linear hierarchy in these subscripts: for instance, for most complexity functions t, we have

$$PrTIME_0(t) \subseteq PrTIME_1(t) \subseteq_1 PrTIME_b(t+2) \subseteq PrTIME_r(t+2) \subseteq PrTIME(t+2).$$

The only case in the above inclusions where t must be restricted is $PrTIME_1(t) \subseteq_1 PrTIME_b(t+2)$ where we require t be to time-constructible. This inclusion follows from the following general result:

We could replace $PrTIME$ here by other suitable mode-resource pairs. These notations are illustrated in the last column of the following table.

The table below lists some important time-feasible (i.e., polynomial time) complexity classes, under the various choice and error modes:

Some Polynomial Time Stochastic Classes

Error Mode	*Choice Mode*	*Common Symbol*	*Generic Notation*
Unbounded error	$\{\oplus\}$	PP	$PrTIME(n^{O(1)})$
Bounded error	$\{\oplus\}$	BPP	$PrTIME_b(n^{O(1)})$
One-sided error	$\{\oplus\}$	RP (also denoted VPP or R)	$PrTIME_1(n^{O(1)})$
Bounded zero-error	$\{\oplus\}$	ZPP	$PrTIME_0(n^{O(1)})$
Bounded error	$\{\oplus, \vee\}$	IP,	$IpTIME_b(n^{O(1)})$
Bounded error	$\{\oplus, \vee\}$	AM	$IpTIME\text{-}(n^{O(1)}, O(1))$

Remark: PP is the only class in this table that is not error-restricted. Hence PP-machines are not *á priori* required to be decisive, and such machines have polynomial acceptance (not running) time.

Relationships Among Feasible Time Stochastic Classes. Let us show some known inclusions among these classes and the connection to canonical classes such P and NP. The following are trivial relationships.

$$P \subseteq ZPP \subseteq RP \subseteq BPP \subseteq PP \subseteq PSPACE,$$

$$BPP \subseteq IP \cap PP.$$

We also have

$$RP \subseteq_1 NP \subseteq_2 PP.$$

The inclusion ($\subseteq_2$) follows from equation (1) while inclusion ($\subseteq_1$) follows from equation (2).

From the previous chapter (corollary 25), we have:

$$PP \subseteq IP \subseteq PSPACE.$$

However, we shall see that $IP = PSPACE$. Recall that $Primes \in$ co-RP.

We now show the following, attributed to Rabin:

$$ZPP = RP \cap \text{co-}RP. \tag{3}$$

Some preliminary remarks are in order: recall that $\oplus$-operator (see Section 7.1) satisfies De Morgan's law: $(1 - x) \oplus (1 - y) = 1 - (x \oplus y)$. This means that if M is a halting probabilistic machine, and we negate its local answers (interchanging YES and NO), then we obtain a machine $\overline{M}$ such that $L(M) \cap L(\overline{M}) = \emptyset$. If M is decisive, then $L(\overline{M}) = \text{co-}L(M)$. Moreover, if M has 1-sided error, then $\overline{M}$ has bounded zero-error acceptance.

One direction of (3) is clear: $ZPP \subseteq RP \cap \text{co-}RP$. Conversely, suppose $L \in RP \cap \text{co-}RP$. By the preceding remarks, L is accepted by bounded error polynomial-time probabilistic machines M and N where M has 1-sided error and N has bounded zero-error acceptance. We may assume that M and N are halting. We then construct a probabilistic machine that dovetails the computation of M and N in a step-for-step fashion until one of the following two events: (a) if N answers YES, we answer YES; (b) if M answers NO, we answer NO. If N answers NO or M answers YES, we simply continue simulation of the other machine. If both machines halt without events (a) or (b) occurring, we loop. This dovetail process essentially gives us a computation tree whose paths can be made to correspond in a bijective fashion with the set of all pairs of paths (p, p') where p comes from M and p' from N. Then it is not hard to see that the simulation runs in polynomial time with zero-error (Exercise).

Figure 8.2 summarizes the relationship of the main feasible-time classes based on stochastic choices with the canonical classes:

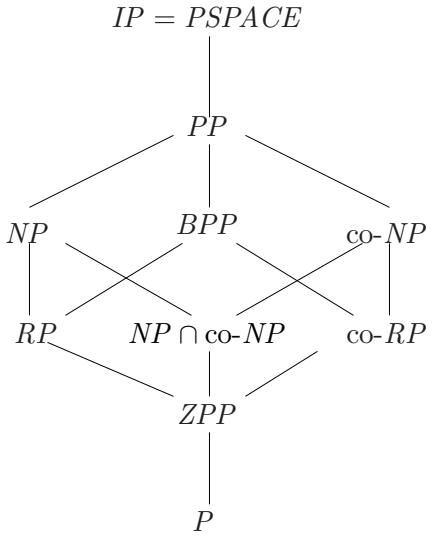


Figure 8.2: Feasible classes in the stochastic mode

Remark: The "error" terminology can be confusing but it is unfortunately well-established in the literature. We have attempted to systematize these concepts in the light of interval values. Our classification of errors into pessimistic versus optimistic may be helpful: thus, "error gap" and "bounded-error" *always* refers to pessimistic

errors, and "zero-error" *always* refers to optimistic errors. Since pessimistic and optimistic errors are independent constraints on a computation, the two concepts can be distinguished in our terminology: the word "error" is always accompanied by key words such as "bounded", "gap" or "zero" which unambiguously indicate the form of error. Most of the literature do not discuss these concepts in the generality given here, mostly focusing on polynomial-time classes. Indeed, while *BPP* is standard notation, there is no standard notation for classes such as $PrTIME_b(t(n))$.

Exercise

Exercise 8.1.1: Consider an alternative approach that distinguishes YO-answers from looping: assign the value $\frac{1}{2}$ to YO-configurations. (Thus, basis sets B for choice machines are required to the new constant function, $\frac{1}{2}$ in addition to the others. Also, the function $\frac{1}{2}$ adds nothing new if the toss-function $\textcircled{+}$ is already in B.) In the standard treatment of YO-configurations, it is not hard to see that a $\{\otimes, \oplus\}$-machine amounts to an alternating machine. In what sense does the alternative treatment of YO-configurations apparently increase the power of such machines? Is this apparent increase real? □

Exercise 8.1.2: Suppose a stochastic machine is not decisive. Prove that if it accepts in time bound $t(n)$ where t is time-constructible, then we can convert it into one accepting in time $O(t)$ with the minimal error gap $[\frac{1}{2}, \frac{1}{2}]$. Prove the same for space bounded acceptance.

□

Exercise 8.1.3: Let $t(n)$ be any complexity function and $K_1 = PrTIME_1(t(n))$. Also let K_2 be the class of languages accepted by probabilistic choice machines with bounded error and zero-error acceptance, running in time $t(n)$. Under what conditions on t would we obtain $K_1 = \text{co-}K_2$? □

Exercise 8.1.4: Complete the arguments showing $ZPP = RP \cap \text{co-}RP$. □

Exercise 8.1.5: (Gill)
(i) Show that *PP* and *BPP* are closed under complementation.
(ii) Show that *BPP* and *RP* are closed under union and intersection.

□

Exercise 8.1.6: (Gill)

We consider probabilistic transducers. For any probabilistic transducer M and input w, we may talk of the probability that M on w produces x as output. Let this (least fixed point) probability be denoted $\Pr\{M(w) = x\}$. Let t_M be the partial transformation such that for all w, $t_M(w) = x$ if $\Pr\{M(w) = x\} > 1/2$; and $t_M(w) \uparrow$ if there is no such x. Clearly t_M is uniquely determined by M and is called the transformation computed by M. Show that if M is $s(n)$-space bounded then $x = t_M(w)$ implies $|x| \leq f(s(|w|))$ where $f(n)$ is the number of configurations of M using space at most n.

□

End Exercise

8.2 How to Amplify Error Gaps

In this section, we seek techniques to convert a machine with error gap G into one with a strictly larger error gap G', $G \subset G'$. We now describe three such techniques, depending on the form of error and type of machine. The importance of such techniques is clear: if we can make such error gaps approach $(0, 1)$ (while keeping the running time feasible) then probabilistic algorithms may become indistinguishable from deterministic ones. For instance, a "theoretical" algorithm with a error probability of less than 2^{-1000} is much more reliable than any implemented algorithm will ever be, since implemented algorithms will contain many other (non-mathematical) sources of error such as the unreliability of hardware, not to speak of software.

(I) Halting Zero-error Rejection Machines. Let M be a halting probabilistic machine with zero-error rejection. Suppose M accepts[3] if with error gap $[0, b]$. We can boost the error gap very easily as follows. Fix some $k \geq 1$. Let N be the following machine:

> On any input w, simulate M on w for k times. This yields k answers, since M is halting. If any of the simulation answers YES, N answers YES. If all the k answers are NO or YO, N answers NO.

If w is rejected by M, then N always answer NO (this is because N has zero-error rejection). If w is accepted by M, the probability of a YES-computation path of M is at least b. Hence the probability that N accepts is at least $1 - (1 - b)^k$. So N accepts with error gap

$$[0, 1 - (1 - b)^k]$$

which strictly includes $[0, b]$. Thus, *halting zero-error rejection implies bounded-error acceptance.* The technique will not work if M can have pessimistic errors in rejection.

(II) Bounded-error Probabilistic Machines. For bounded-error machines, we can use the "majority voting" scheme: repeat for an odd number of times an experiment with binary outcome; we take the majority outcome (*i.e.*, the outcome occurring more than half the time) as output. We justify this procedure with a lemma [29]:

LEMMA 1 *(a) Consider an experiment in which an event E occurs with probability*

$$p \geq \frac{1}{2} + e$$

for some $0 < e < \frac{1}{2}$. Then in $2t + 1$ independent trials of the experiment, the probability that E is the majority outcome is greater than

$$1 - \frac{1}{2}(1 - 4e^2)^t.$$

(b) Similarly, if E occurs with probability

$$p \leq \frac{1}{2} - e$$

then the probability that E is the majority outcome is less than

$$\frac{1}{2}(1 - 4e^2)^t.$$

Proof. (a) Let $q = 1 - p$ and $i = 0, \ldots, t$. Then the probability p_i that E occurs exactly i times out of $2t + 1$ is given by the binomial distribution,

$$\begin{aligned}
p_i &= \binom{2t+1}{i} p^i q^{2t+1-i} \\
&= \binom{2t+1}{i} (\tfrac{1}{2} + e)^i (\tfrac{1}{2} - e)^{2t+1-i} \left[\frac{p}{\frac{1}{2}+e}\right]^i \left[\frac{q}{\frac{1}{2}-e}\right]^{2t+1-i} \\
&= \binom{2t+1}{i} (\tfrac{1}{2} + e)^i (\tfrac{1}{2} - e)^{2t+1-i} \left[\frac{pq}{(\frac{1}{2}+e)(\frac{1}{2}-e)}\right]^i \left[\frac{q}{\frac{1}{2}-e}\right]^{2t+1-2i} \\
&\overset{*}{\leq} \binom{2t+1}{i} (\tfrac{1}{2} + e)^i (\tfrac{1}{2} - e)^{2t+1-i} \\
&\leq \binom{2t+1}{i} (\tfrac{1}{2} + e)^i (\tfrac{1}{2} - e)^{2t+1-i} \left[\frac{\frac{1}{2}+e}{\frac{1}{2}-e}\right]^{t-i} \\
&= \binom{2t+1}{i} (\tfrac{1}{2} + e)^t (\tfrac{1}{2} - e)^{t+1} \\
&< \binom{2t+1}{i} (\tfrac{1}{4} - e^2)^t \frac{1}{2}.
\end{aligned}$$

[3]By definition, $b \geq 1/2$. But in some sense, this technique works as long as $b > 0$.

Note that our derivation is careful to justify the transition ($\stackrel{*}{\leq}$) from p to $\frac{1}{2}+e$. Therefore the probability that E occurs in more than t trials is at least

$$\begin{aligned} 1-\sum_{i=0}^{t} p_i &> 1-\sum_{i=0}^{t}\binom{2t+1}{i}(\frac{1}{4}-e^2)^t\frac{1}{2} \\ &= 1-2^{2t}(\frac{1}{4}-e^2)^t\frac{1}{2} \\ &= 1-\frac{1}{2}(1-4e^2)^t. \end{aligned}$$

(b) Similarly, if $p \leq \frac{1}{2}-e$ then for $i \geq t+1$ we have

$$p_i \leq \binom{2t+1}{i}(\frac{1}{4}-e^2)^t\frac{1}{2}$$

and hence the probability that E occurs in more than t trials will be at most

$$\begin{aligned} \sum_{i=t+1}^{2t+1} p_i &\leq 2^{2t}(\frac{1}{4}-e^2)^t\frac{1}{2} \quad (4) \\ &= \frac{1}{2}(1-4e^2)^t. \quad (5) \end{aligned}$$

Q.E.D.

Using this, we can boost an error gap $G_e = [\frac{1}{2}-e, \frac{1}{2}+e]$ $(0 < \frac{1}{2} < e)$ to $[\frac{1}{2}-e', \frac{1}{2}+e']$ where

$$e' = \frac{1}{2}(1-4e^2)^t$$

if we do the majority vote for $2t+1$ trials. For instance, with $e = 1/4$ and $t = 8$, we have $e' = \frac{1}{2}(3/4)^t < 0.051$.

An **error gap function** G assigns an error gap $G(n)$ to each $n \in \mathbb{N}$. A machine M has error gap G if for all inputs w, $Val_M(w) \cap G(|w|) = \emptyset$. Let G_0 be the error gap function given by

$$G_0(n) = [2^{-n}, 1-2^{-n}].$$

We have the following useful lemma:

LEMMA 2 *Each language in BPP is accepted by a probabilistic acceptor that runs in polynomial time with error gap G_0.*

Proof. We may assume that the language is accepted by some M that runs in time n^d with error gap $G = [\frac{1}{2}-e, \frac{1}{2}+e]$ for some $d \geq 1$ and $0 < e < \frac{1}{2}$. Applying the lemma, we want to choose t satisfying

$$\begin{aligned} 2^{-n} &\geq \frac{(1-4e^2)^t}{2} \\ 2^{n-1} &\leq \frac{1}{(1-4e^2)^t} \\ n-1 &\leq t\log\left(\frac{1}{1-4e^2}\right) \\ t &\geq \frac{n-1}{\log(1/(1-4e^2))}. \end{aligned}$$

The desired machine N, on each computation path, simulates M for at most $2t+1 = O(n)$ times and outputs the majority outcome. Clearly N runs in time $O_e(n^{d+1})$ with error gap G_0. **Q.E.D.**

Let us give an application of this lemma:

THEOREM 3 (Ko, 1982) *If NP $\subseteq$ BPP then NP = RP.*

Proof. Since $RP \subseteq NP$, it suffices to show inclusion in the other direction. It is easy to see that RP is closed under polynomial-time many-one reducibility, and hence we only have to show that the NP-complete language SAT belongs to RP. Suppose we want to check if a given CNF formula $F = F(x_1, \ldots, x_n)$ on n variables is satisfiable. For any sequence of Boolean values $b_1, \ldots, b_k$ ($k \leq n$), let $F_{b_1 b_2 \cdots b_k}$ denote the formula F with x_i replaced by b_i, for $i = 1, \ldots, k$. We show how to construct a sequence $b_1, \ldots, b_n$ such that if F is satisfiable then $F_{b_1 \cdots b_n}$ is true with very high probability. By our assumption that $NP \subseteq BPP$, there is a bounded-error probabilistic machine M accepting SAT in polynomial time. Moreover, by the preceding lemma, we may assume that M has error gap function G_0 and that M halts on every path in polynomial time.

We shall operate in n stages. At the start of stage k ($k = 1, \ldots, n$), inductively assume that we have computed a sequence of Boolean values $b_1, \ldots, b_{k-1}$. It will be shown that $F_{b_1,\ldots,b_{k-1}}$ is probably satisfiable. In stage k, we compute b_k:

1. Call M on input $F_{b_1 \cdots b_{k-1} 0}$.
2. If M answers YES, then set $b_k = 0$ and go to DONE.
3. Else call M on input $F_{b_1 \cdots b_{k-1} 1}$.
4. If M answers NO again, we answer NO and return.
5. Else set $b_k = 1$.
6. DONE: If $k < n$ we go to stage $k + 1$.
7. Else answer YES if $F_{b_1,\ldots,b_n} = 1$, otherwise answer NO.

Let us analyze this procedure. It is clearly polynomial time.

If $k < n$, we either terminate in stage k with a NO answer, or we proceed to stage $k+1$. If $k = n$, we will surely terminate in stage k with answer YES or NO, and this answer is never in error. Thus our YES answers are never wrong. So if F is unsatisfiable, we answer NO on every path. Thus we have zero-error rejection.

Finally, let us prove that if F is satisfiable, then our procedure answer YES with probability $> 1/2$. Write F_k for $F_{b_1,\ldots,b_k}$, assuming that $b_1, \ldots, b_k$ are defined. Let the event A_k correspond to " no mistakes up to stage k", *i.e.*, F_k is defined and satisfiable. Similarly, let event E_k correspond to "first mistake at stage k", *i.e.*, $E_k = A_{k-1} \cap \overline{A_k}$.

CLAIM: $\Pr(E_k) \leq 2 \cdot 2^{-|F|+1})$.

Proof: Note that $\Pr(E_k) \leq \Pr(E_k | A_{k-1})$. We will bound $\Pr(E_k | A_{k-1})$. Assuming A_{k-1}, we consider 2 cases: (A) CASE $F_{b_1 \cdots b_{k-1} 0}$ is not satisfiable. Then $F_{b_1 \cdots b_{k-1} 1}$ is satisfiable. With probability $\geq (1 - 2^{-|F|})$, M will (correctly) answer NO the first time we invoke M. Then with probability $\geq (1 - 2^{-|F|})$, M will (correctly) answer YES the second time. So $\Pr(A_k | A_{k-1}) \geq (1 - 2^{-|F|})^2$ and

$$\Pr(E_k | A_{k-1}) \leq 1 - (1 - 2^{-|F|})^2 \leq 2^{-|F|+1}).$$

(B) CASE $F_{b_1 \cdots b_{k-1} 0}$ is satisfiable. This case is even easier, and yields $\Pr(E_k | A_{k-1}) \leq 2^{-|F|}$. This proves the claim.

To conclude the theorem, the probability of making mistake at any stage is at most

$$\sum_{k=1}^{n} \Pr(E_k) \leq 2n \cdot 2^{-|F|} \leq 2n \cdot 2^{-2n}.$$

This is less than $1/2$ for n large enough. Hence F will be accepted. **Q.E.D.**

See Exercise for another proof.

(III) Stochastic machines. We now introduce a third technique that is applicable to stochastic machines. Motivated by a paper of Valiant, we introduce the following probability functions:

- $P(x) := x \otimes x = x^2$
- $Q(x) := x \oplus x = x(2 - x)$.
- $A(x) := Q(P(x)) \ominus P(Q(x))$.

Thus,

$$A(x) = \frac{x^2(2 - x^2) + x^2(2 - x)^2}{2} = x^2(3 - 2x).$$

These operators are extended to INT in the usual way: thus, $A([u, v]) = [A(u), A(v)]$. The exercises show other properties of these functions. We show that $A(x)$ has the following "amplification property":

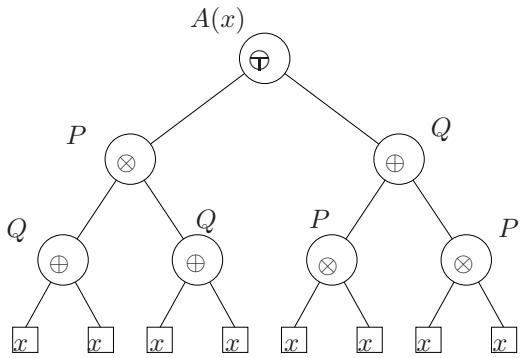


Figure 8.3: The operator $A(x)$

LEMMA 4 *If* $0 \leq e \leq \frac{1}{2}$, *then*

$$x \notin [\frac{1}{2} - e, \frac{1}{2} + e] \quad \Rightarrow \quad A(x) \notin [\frac{1}{2} - e', \frac{1}{2} + e']$$

where

$$e' = e(\frac{3}{2} - 2e^2) \geq e.$$

Proof. Since $\frac{dA}{dx} = 6x(1-x)$, $A(x)$ is monotone increasing for $0 \leq x \leq 1$. The lemma then follows from a calculation,

$$\begin{aligned} A(\frac{1}{2} + e) &= \frac{1}{2} + e[\frac{3}{2} - 2e^2] \\ A(\frac{1}{2} - e) &= \frac{1}{2} - e[\frac{3}{2} - 2e^2]. \end{aligned}$$

Q.E.D.

Note that the error gap has an "amplification factor" $\alpha(e) = \frac{3}{2} - 2e^2$ which decreases monotonically from 3/2 to 1 as e increases from 0 to 1/2. Note that

$$\alpha(1/4) = \frac{11}{8}.$$

So if the error bound e for a value x is less than $1/4$, then the error bound for $A(x)$ is at least $11e/8$. We conclude:

THEOREM 5
Let $t(n)$ *be a time-constructible function. Then*

$$PrTIME(t) \subseteq StTIME_b(O(t)).$$

In particular,

$$PP \subseteq StTIME_b(n^{O(1)}).$$

Proof. Suppose M is a probabilistic machine that decides its inputs in running time t. Since t is time-constructible, we can modify it always halt in time $O(t)$ and have error gap

$$G(n) = [\frac{1}{2} - 2^{-t}, \frac{1}{2} + 2^{-t}].$$

Let $s = \frac{t}{\log(11/8)}$. We construct a stochastic machine N that, on input w, operates by first "computing the iterated function $A^s(x)$" (the meaning should be clear) and then simulating M on w. One checks that N has gap at least $[\frac{1}{4}, \frac{3}{4}]$. Note that N still runs in time $O(t)$. **Q.E.D.**

EXERCISE

Exercise 8.2.1: This is a slight variation on boosting error gaps for machines with zero-error rejection. Let M be a machine with error gap $G = (0, b]$. We construct the following machine N to boost the error gap G:

1. Simulate M on input from beginning until it halts.
   (If M loops then we loop)
2. If M answers YES then answer YES; else toss a coin.
3. If coin-toss is heads then answer NO; else go to 1.

(i) Show that N has error gap $G' = (0, b']$ where $b' = \frac{2b}{1+b}$.
(ii) For any $\epsilon > 0$, show how to modify step 3 so that $1 - b' \leq \epsilon$.

□

Exercise 8.2.2: Let M be a bounded-error halting probabilistic machine. We construct a probabilistic machine N that keeps simulating M on its input until the "running score" reaches $+2$ or -2. The running score (at any moment) is defined to be the number of YES answers minus the number of NO answers up to that moment. If we reach $+2$, we answer YES, and if we reach -2, we answer NO.
(i) What is the language accepted by N?
(ii) Analyze the complexity and errors of N.

□

Exercise 8.2.3: Give another proof of Ko's theorem that $NP \subseteq BPP$ implies $NP = RP$ in §2. Recall the event A_k in the first proof.
(i) Show that $\Pr(A_n) \geq (1 - 2^{-2n})^{2n}$.
(ii) Show that $(1 - 2^{-n})^n \geq \frac{1}{2}$ for large enough n. HINT: you may use the following facts:

- $e^t \geq 1 + t$ for all $t \in \mathbb{R}$ with equality iff $t = 0$.
- $\left(1 + \frac{t}{n}\right)^n \geq e^t \left(1 - \frac{t^2}{n}\right)$ for all $t, n \in \mathbb{R}$, $n \geq 1$ and $|t| \leq n$.

□

Exercise 8.2.4:
(i) Redraw the class inclusion diagram in Figure 8.2 assuming that $NP \subseteq BPP$.
(ii) What consequences for the diagram of Figure 8.2 can you draw if we assume that $BPP \subseteq NP$?

□

Exercise 8.2.5: Let $g = [1/3 - e, 1/3 + e]$ for some $0 < e < 1/3$. Give an analog of majority voting to amplify this gap.

□

Exercise 8.2.6: Let $P(x)$ and $Q(x)$ be as defined for boosting error gaps in stochastic machines. For all $n = 0, 1, 2, \ldots$, let

$$P_n(x) := \begin{cases} x & \text{if} \quad n = 0 \\ P(P_{n-1}(x)) & \text{if} \quad n = odd \\ Q(P_{n-1}(x)) & \text{if} \quad n = even \end{cases}$$

$$Q_n(x) := \begin{cases} x & \text{if} \quad n = 0 \\ Q(Q_{n-1}(x)) & \text{if} \quad n = odd \\ P(Q_{n-1}(x)) & \text{if} \quad n = even \end{cases}$$

For example, $P_2(x) = Q(P(x)) = x^2(2 - x^2)$, and $Q_2(x) = P(Q(x)) = x^2(2 - x)^2$. The function $Q_2(x)$ was used by Valiant to give a non-constructive proof that the majority function for Boolean functions has a monotone formula of size $O(n \log n)$. The amplification function in the text is just $A(x) = P_2(x) \oplus Q_2(x)$. Now write $p_n^+(e)$ for $P_n(\frac{1}{2} + e)$, $p_n^-(e)$ for $P_n(\frac{1}{2} - e)$, and similarly for $q_n^+(e), q_n^-(e)$ relative to Q_n. For example,

$$\begin{aligned} p_1^+(e) &= \frac{1}{4} + e + e^2 \\ q_1^+(e) &= \frac{3}{4} + e - e^2 \end{aligned}$$

$$
\begin{aligned}
p_1^-(e) &= \frac{1}{4} - e + e^2 \\
q_1^-(e) &= \frac{3}{4} - e - e^2 \\
p_2^+(e) &= \frac{7}{16} + \frac{3e}{2} + \frac{e^2}{2} - 2e^3 - e^4 \\
q_2^+(e) &= \frac{9}{16} + \frac{3e}{2} - \frac{e^2}{2} - 2e^3 + e^4 \\
p_2^-(e) &= \frac{7}{16} - \frac{3e}{2} + \frac{e^2}{2} + 2e^3 - e^4 \\
q_2^-(e) &= \frac{9}{16} - \frac{3e}{2} - \frac{e^2}{2} + 2e^3 + e^4 \\
p_3^+(e) &= \frac{49}{256} + \frac{21e}{16} + \frac{43e^2}{16} - \frac{e^3}{4} - \frac{53e^4}{8} - 5e^5 + 3e^6 + 4e^7 + e^8 \\
q_3^+(e) &= \frac{207}{256} + \frac{21e}{16} - \frac{43e^2}{16} - \frac{e^3}{4} + \frac{53e^4}{8} - 5e^5 - 3e^6 + 4e^7 - e^8
\end{aligned}
$$

Show
(i) $p_n^+(e) + q_n^-(e) = 1$
(ii) $p_n^-(e) + q_n^+(e) = 1$
(iii) $p_n^+(e) - p_n^-(e) = q_n^+(e) - q_n^-(e)$
(iv) $x = \frac{1}{2}$ is a fixed point of $A_n(x) = P_n(x) \oplus Q_n(x)$, *i.e.*, $A_n(\frac{1}{2}) = \frac{1}{2}$. Are there other fixed points?
(v) The exact relationships between the coefficients of $p_n^+(e), q_n^+(e), p_n^-(e)$ and $q_n^-(e)$.

□

Exercise 8.2.7: Show that $RP = NP$ iff some NP-complete language is in RP. □

Exercise 8.2.8: Another proof that $NP \subseteq BPP$ implies $NP = RP$. Recall the event A_k in the first proof.
(i) Show that $\Pr(A_n) \geq (1 - 2^{-2n})^{2n}$.
(ii) Show that $(1 - 2^{-n})^n \geq \frac{1}{2}$ for large enough n. HINT: you may use the following facts:

- $e^t \geq 1 + t$ for all $t \in \mathbb{R}$ with equality iff $t = 0$.
- $\left(1 + \frac{t}{n}\right)^n \geq e^t \left(1 - \frac{t^2}{n}\right)$ for all $t, n \in \mathbb{R}$, $n \geq 1$ and $|t| \leq n$.

□

End Exercise

8.3 Probabilistic Feasible Time

Feasible time simply means polynomial time, of course. Here we consider two versions of feasible time, depending on the error concept: unbounded error (the class PP) and bounded error (the class BPP).

8.3.1 Unbounded Error Polynomial Time

We derive some basic properties of the class PP.

Lemma 6 *For any PP-machine M, there exists another PP-machine N such that $L(N) = L(M)$ where N is halting, decisive, and always answers YES or NO.*

Proof. (Sketch) The machine N simulates M. To make N halting, we clock the machine M for a polynomial number of steps and answer NO if M does not answer YES by the time the clock runs out. This means an YO answer is also turned into a NO answer. To ensure that N is decisive, we "shift" the gap $(\frac{1}{2}, \frac{1}{2} + e)$ to some error gap $(\frac{1}{2} - e', \frac{1}{2} + e')$. Here $e = e(n)$ depends on the length n of the input, and can be assumed to be a binary rational which is polynomial time computable. **Q.E.D.**

The class PP satisfies the following remarkable closure property:

LEMMA 7 *If $A, B \in PP$ then the symmetric difference $A \oplus B \in PP$.*

Proof. Let M_A, M_B be PP-machines that accept A and B (resp.). By the previous lemma, assume M_A and M_B are halting and decisive and always answer YES or NO. Consider the machine N that, on any input x, simulates M_A on x and then M_B on x. N will answer YES if exactly one of M_A and M_B answers YES, otherwise N answers NO. We note that

- If both M_A and M_B answered correctly (answers YES iff the global answer is ACCEPT), then N's answer is correct.
- If both M_A and M_B answered incorrectly then N's answer is (still) correct.
- If exactly one of M_A or M_B answered incorrectly, then N's answer is incorrect.

Assume that the probability of M_A (resp., M_B) being correct on any input x is *exactly* $(1/2) + \varepsilon_A$ (resp., $(1/2) + \varepsilon_B$). Notice that $\varepsilon_A > 0$ and $\varepsilon_B > 0$ exists since the machines are decisive. Moreover ε_A and ε_B does not depend on x. Then the probability of N being correct on input x is exactly

$$(\frac{1}{2} + \varepsilon_A)(\frac{1}{2} + \varepsilon_B) + (\frac{1}{2} - \varepsilon_A)(\frac{1}{2} - \varepsilon_B) = \frac{1}{2} + 2\varepsilon_A\varepsilon_B.$$

Q.E.D.

An alternative argument is as follows: if probability of M_A (M_B) answering YES is a (b), then the probability of N answering YES is $c = a(1-b) + (1-a)b$. Note that we can add $a(1-b)$ and $(1-a)b$ because these correspond to disjoint events. We see that

$$\begin{aligned} \frac{1}{2} - c &= \frac{1}{2}\left(\frac{1}{4} - \frac{a+b}{2} + ab\right) \\ &= \frac{1}{2}(\frac{1}{2} - a)(\frac{1}{2} - b). \end{aligned}$$

Since the machines are decisive, $a \neq 1/2$ and $b \neq 1/2$, and so $1/2 - c \neq 0$. Moreover, $1/2 - c < 0$ iff exactly one of the inequalities, $a < 1/2$ and $b < 1/2$, hold. This is exactly what we want to show.

The following is left as an exercise.

LEMMA 8 *PP is closed under complementation.*

The next result is from Beigel, Reingold and Spielman [3].

THEOREM 9 *PP is closed under intersection.*

The proof uses an interesting property of polynomials, omitted here.

We now investigate complete languages for PP. It should be noted that among the stochastic feasible classes (ZPP, RP, BPP, PP), only PP has known complete languages. For any Boolean formula F, let $\#(F)$ denote the number of satisfying assignments to variables that occur in F. Consider three related languages (we include the standard SAT in this table for comparision):

$$\begin{aligned} \text{MAJ} &:= \{\langle F\rangle : \#(F) > 2^{m-1} \text{ where } m \text{ is the number of variables in } F\} \\ \#\text{SAT} &:= \{\langle F, k\rangle : F \text{ is 3CNF and } \#(F) \geq k\} \\ \#\text{SAT}_0 &:= \{\langle F, k\rangle : F \text{ is 3CNF and } \#(F) = k\} \\ \text{SAT} &:= \{\langle F\rangle : F \text{ is 3CNF and } \#(F) \geq 1\} \end{aligned}$$

Here $\langle F\rangle$ and $\langle F, k\rangle$ denote any reasonable encoding of F and (F, k) as binary strings.

LEMMA 10 MAJ *and* #SAT *are both PP-hard under Karp reducibility.*

Proof. (a) It is easy to see that MAJ $\in PP$: construct a PP machine which accepts a Boolean formula F with probability equal to $\#(F)/2^m$ (if F has m variables. Thus the machine accepts precisely the formulas of MAJ.
(b) We show that MAJ is PP-hard. If M is a PP-machine and x is an input, we want a 3CNF formula F such that $\#(F)$ is equal to the number of YES-paths in the computation tree for x. But we note that the proof of the

NP-hardness of SAT (Cook's Theorem in Chapter 3) produces such a formula. See Exercise.
(c) Clearly #SAT is *PP*-hard since we can reduce MAJ to #SAT.
(d) Finally, to show #SAT $\in$ *PP* it is sufficient to construct a transformation $t : \langle F, k\rangle \mapsto t(F, k)$ such that $\langle F, k\rangle \in$ #SAT iff $t(F, k) \in$ MAJ. Suppose F has $m + 1$ free variables. If $k = 0$, then clearly $\langle F, k\rangle \in \#\mathrm{SAT}_1$. Hence we may assume $1 \leq k \leq 2^{m+1}$. We construct a CNF forumula C_k with these $m + 1$ variables of F such that exactly $2^{m+1} + 1 - k$ assignments satisfy C_k. Let z be a new variable. Then

$$t(\langle F, k\rangle) = (z \wedge F) \vee (\overline{z} \wedge C_k).$$

It is easy to see that $\langle F, k\rangle \in$ #SAT iff $t(\langle F, k\rangle) \in SAT_0$. The result is not quite CNF yet. Assuming F and C_k are CNF, then $z \wedge F$ and $\overline{z} \wedge C_k$ are each CNF. In general, to create a CNF formula G from the disjunction $G_1 \vee G_1$ of two CNF formulas G_1, G_2, we let each clause of G be written as the disjunct of a clause of G_1 with a clause of G_2. Of course, G need not be 3-CNF (it is a bit more work to achieve this, while ensuring that $G \in SAT_0$).

If $k = 1$, then C_k can be taken to be 1 (always true). Otherwise $k \geq 2$ and we consider the binary expansion of $2^{m+1} + 1 - k = \sum_{j=0}^{m} b_j 2^j$ ($b_j = 0$ or 1). We leave the details to the reader. **Q.E.D.**

The above techniques do not work with $\#\mathrm{SAT}_0$, and we also do not know any complete languages for any error-limited complexity classes such as *RP* or *BPP*.

EXERCISE

Exercise 8.3.1: Prove lemma 8. □

Exercise 8.3.2: The proof of Cook's theorem in Chapter 3 reduces any language $A \in NP$ to SAT. If M is an *NP*-machine for A, for any input w, the proof constructs a 3CNF formula F_w such that F_w is satisfiable iff $w \in A$. We claim that something stronger is true: $\#F_w$ is equal to the number of accepting computations of M on input w. Actually, this claim needs a mild condition on M. What is it? Prove this claim under this mild condition. **Hint:** if you do not see what this condition might be, we suggest the strategy of ignoring it at first, and trying to prove the claim unconditionally. □

Exercise 8.3.3*: Does *ZPP*, *RP* or *BPP* have complete languages under any reasonable reducibilities? □

END EXERCISE

8.3.2 Bounded Error Polynomial Time

See Balcazar' book.

This should go to Constructive Hierarchies Chapter:
(1) *BPP* is in $\Sigma_2 \cap \Pi_2$.
(I think the current results here are not the most general... see the proof)
(2) Approximate Counting
(3) $NTIME(n) \neq DTIME(n)$.
(4) *BPP* has polynomial size circuits
We use the following fact:

LEMMA 11 *Let M be a choice machine with only binary choices, $L(M) \subseteq \mathbb{B}^*$, and M runs in time $t(n)$. Then there is a polynomial $p(n)$ such that for all n, there is a Boolean circuit C_n on $n + t(n)$ inputs such that*
(1) C_n has size $p(t(n))$, and
(2) for all $x \in \mathbb{B}^n$ and all $y \in \mathbb{B}^{t(n)}$, $C_n(x, y) = 1$ iff M on input x and making the choices specified by y leads to a YES answer.

Proof. Sketch proof: There is a fixed size circuit that computes the "local transition" function, as given in Papadimitriou's book (theorem 8.1, p.168). Note: this function does not depend on M having only one worktape. **Q.E.D.**

THEOREM 12 *Every binary language in BPP has a polynomial size circuit.*

Proof. [As in Papadimitriou] Let $L \subseteq \mathbb{B}^*$ be accepted by a *BPP*-machine M. We will construct for each n, a Boolean circuit C_n^* of polynomial size such that $x \in L$ iff $C_n^*(x) = 1$.

We may assume that for any input x, $\Pr\{M(x) = error\} \leq 1/4$. As in the previous lemma, assume M runs in time $t(n)$ and there are circuits C_n of size $p(t(n))$ which simulates M on any input $x \in \mathbb{B}^n$ and any path choice $y \in \mathbb{B}^{t(n)}$. Let $T_n = \mathbb{B}^{12(n+1)t(n)}$. We consider a string $Y \in T_n$ as a $(12(n+1)$-tuple $(y_1, \ldots, y_{12(n+1)})$ of choices for a computation on an input of length n. We say that Y is "good" for $x \in \mathbb{B}^n$ if more than half of the $\{C_n(x, y_i) : i = 1, \ldots, 12(n+1)\}$ is correct. Here, $C_n(x, y)$ is correct if $C_n(x, y) = 1$ iff $x \in L$. For any x, and for a random choice of Y, the probability that $y \in \mathbb{B}^{t(n)}$ is incorrect for x is $\leq 1/4$. Hence the probability that $Y \in T_n$ is not good for x is, by Chernoff's bound at most $e^{-(n+1)}$.

[See Papadimitriou, or my lecture notes]

Now the probability that Y is bad for more any $x \in \mathbb{B}^n$ is therefore at more $2^n e^{-(n+1)} < 1/2$. Hence, there exists $Y^* \in T_n$ that is good for all $x \in B_n$. We now construct a circuit $C_n^*(x)$ to be simply $12(n+1)$ copies of C_n in which each C_n has the form $C_n(x, y_i^*)$ where $Y^* = (y_1^*, \ldots, y_{12(n+1)}^*)$. Finally, the output of C_n^* is just the majority output of all the C_n's. **Q.E.D.**

Note: a random construction of C_n^* will succeed with probability more than $1/2$. So there is a BPP algorithm for this construction? Not yet: you need to check if a given choice of Y is good for all x.

8.4 Average Time for Probabilistic Machines

So far, we have not discussed the very natural concept of "average time" for probabilistic machines. We systematically develop the concept as follows (the appendix contains the probabilistic terms used here). Let M be a probabilistic machine and $T_M(w)$ denotes the usual complete computation tree on input w. Without loss of generality, assume $T_M(w)$ is a full binary tree, *i.e.*, a binary tree in which every internal node has two children. Let T any prefix of $T_M(w)$.

We construct associate with T a probability space

$$(\Omega_T, \Sigma_T, \Pr_T)$$

in which the sample space Ω_T comprises all complete paths of T. A subset of Ω_T is a **basic set** if it is the collection of complete paths of T all sharing some common initial prefix π; denote this set by $B_T(\pi)$. In particular, Ω_T is the basic set $B_T(\epsilon)$ where $\pi = \epsilon$ is the empty path. Any singleton set consisting of a complete path is also a basic set. Let Σ_T^0 comprise all finite union and complement of basic sets: clearly Σ_T^0 forms a field. Let Σ_T be the Borel field generated by Σ_T^0. The probability measure $\Pr_T$ assigns to each basic set $B_T(\pi)$ the probability $\Pr_T(B_T(\pi)) = 2^{-|\pi|}$ where $|\pi|$ is the length of the path π. E.g., $\Pr_T(\Omega_T) = 2^0 = 1$, as expected. Notice that every element of Σ_T^0 is a finite union of disjoint basic sets. Extend the definition of $\Pr_T$ to sets in Σ_T^0 so as to preserve finite additivity of pairwise disjoint unions. One checks that the extension does not depend on how we partition sets in Σ_T^0 into a countable union of basic sets. Finally, a theorem of Carathéodory (appendix) gives us a unique extension of $\Pr_T$ to all of Σ_T.

Definition 4 (i) We introduce three random variables for Ω_T. For any complete paths of T,

$$\begin{aligned} \text{Time}_T(\pi) &= |\pi| \text{ possibly infinite,} \\ \text{Accept}_T(\pi) &= 1\text{iff } \pi \text{ ends in a YES-node in } T, \\ \text{Halt}_T(\pi) &= 1\text{iff } \pi \text{ is a finite path.} \end{aligned}$$

When T is the complete computation tree for input w, we write

$$\text{Time}_w, \text{Accept}_w, \text{Halt}_w, \Omega_w, etc,$$

for Time_T, Accept_T, etc.

(ii) The **average time** of T is the expected value of Time_T. A machine M **accepts/rejects in average time** $t(n)$ if for all accepted/rejects inputs w of length n,

$$\Pr\{\text{Time}_w \leq t(n), \text{Accept}_w = 1\} > \frac{1}{2}.$$

It **runs in average time** $t(n)$ if it is decisive and accepts and also rejects in average time $t(n)$.

■

An equivalent definition of average time is this: the average time of a computation tree T is equal to the sum over the weights of each edge in T where an edge from level $\ell - 1$ to level ℓ (the root is level 0) has weight $2^{-\ell}$. Naturally, **the probability that M halts on w** is the expected value of the random variable Halt_w. If M is halting (*i.e.*, there are no infinite computation paths) then it halts with probability 1; the converse is false. When we use running complexity for probabilistic machines, they are sometimes designated 'Monte Carlo algorithms'; when average complexity is used, they are designated 'Las Vegas algorithms'.

We first note an observation of Gill: *Every recursively enumerable language can be accepted by a probabilistic machine with constant average time.* In proof, suppose M is a deterministic Turing machine. Without loss of generality assume that M accepts whenever it halts. We construct N to simulate M as follows:

```
repeat
    Simulate one step of M;
    if the simulated step halts, we answer YES or NO following M;
until head = cointoss();
if head = cointoss() then answer YES else answer NO.
```

Here cointoss() is a random function that returns *head* or *tail* and there are two separate invocations of this function above. We note that if the input word is not accepted by M then N can only reject (since the probability of N saying YES is equal to the probability of saying NO). Hence N has zero-error rejection. If an input word is accepted by M, then we see that the probability of its acceptance by N is more than $\frac{1}{2}$ since each NO path can be uniquely paired with an YES path of the same length, but there is one YES path that is not paired with any NO path. These remarks are easy to see once we unroll the computation tree of N as shown in Figure 8.4.

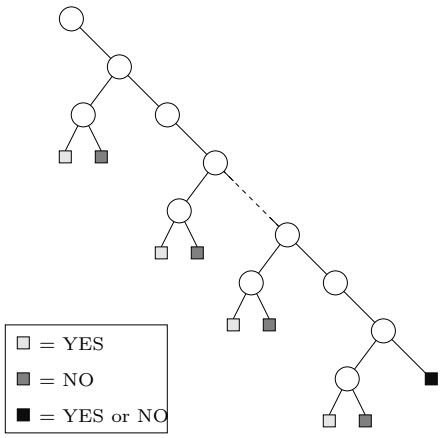


Figure 8.4: Computation Tree of N, simulating a deterministic Turing acceptor M.

The average time $\bar{t}$ spent in the repeat-loop satisfy the inequality

$$\bar{t} \leq 2 + \frac{\bar{t}}{2}$$

where the term '2+' comes from simulating a step of M and tossing a coin to decide on continuing inside the loop (it takes no time to decide to say YES if M says YES). Thus $\bar{t} \leq 4$. The average time of N is at most $1 + \bar{t} \leq 5$ (where the '1' for the final cointoss).

Gill notes that such pathological behaviour does not happen with bounded-error machines:

LEMMA 13 *Let M be a probabilistic machine accepting/rejecting with bounded-error. There is a constant $c > 0$ such that if M accepts/rejects in average time $\bar{t}(n)$ and accepts/rejects in time $t(n)$ then*

$$\bar{t}(n) \geq \frac{t(n)}{c}.$$

Proof. Suppose M accepts with probability at least $\frac{1}{2} + e$ for some $0 < e < \frac{1}{2}$. (The proof if M rejects with probability at most $\frac{1}{2} - e$ is similar.) Fix any input of length n and let $\bar{t} = \bar{t}(n)$. If $\bar{t} = \infty$ there is nothing to prove.

Otherwise, let T be the complete computation tree. Since Time_T is non-negative, Markov's inequality yields

$$\Pr\{\text{Time}_T \geq c\bar{t}\} \leq \frac{1}{c}$$

for any $c > 0$. Choosing $c = \frac{2}{e}$,

$$\begin{aligned} \Pr\{\text{Time}_T < c\bar{t}, \text{Accept}_T = 1\} &\geq \Pr\{\text{Time}_T < c\bar{t}\} - \Pr\{\text{Accept}_T = 0\} \\ &\geq (1 - \frac{1}{c}) - (\frac{1}{2} - e) \\ &\geq \frac{1}{2} + \frac{e}{2}. \end{aligned}$$

This proves that T, truncated below $c\bar{t}$, accepts with bounded error. **Q.E.D.**

In view of this lemma, let

$$AvgTIME(t(n))$$

denote the class of languages accepted by probabilistic machines M where M has bounded-error and M runs in average time $t(n)$. Note that both properties here are independently defined for the entire computation tree. We have thus proved:

COROLLARY 14 *For any $t(n)$,*

$$AvgTIME(t) \subseteq PrTIME_b(O(t)). \tag{6}$$

As further example, we provide deterministic time upper bounds for languages accepted in average time $t(n)$ with bounded-error.

COROLLARY 15 *If a bounded-error probabilistic machine M accepts with average time $\bar{t}(n)$ then $L(M) \in DTIME(O(1)^{\bar{t}(n)})$.*

Proof. We can simulate M by computing the least fixed point of a computation tree of depth $O(\bar{t}(n))$. **Q.E.D.**

LEMMA 16 *Let $s(n) \geq \log n$ be space constructible. Let M be any nondeterministic machine that accepts in space s. Then there is a probabilistic machine N with zero-error that accepts $L(M)$ in space s.*

Proof. Choose $c > 0$ such that there are at most $c^{s(n)}$ configurations using space at most $s(n)$. Fix any input of length n and let $s = s(n)$. First we mark out exactly s cells. The probabilistic machine N proceeds as follows:

repeat forever
1. Initialize M to its initial configuration.
2. Simulate M for c^s steps. Nondeterministic choices of M become coin-tossing choices of N.
3. If M answers YES in this simulation, we answer YES. (If M answers NO or YO or does not halt in c^s steps, then we go to back to 1.)

end

Clearly N loops if M does not accept. If M accepts then the probability of N answering YES is easily seen to be 1. **Q.E.D.**

This lemma implies that probabilistic space-bounds with zero-error is as powerful as nondeterministic space:

THEOREM 17 *For any space-constructible $s(n) \geq \log n$,*

$$NSPACE(s) = PrSPACE_0(s) = PrSPACE_1(s).$$

Proof. The above lemma shows that $NSPACE(s) \subseteq PrSPACE_0(s)$. The converse is easy since for any probabilistic machine M with zero error, when viewed as a nondeterministic machine N, accepts the same language with the same space bound. We check that the same construction applied to probabilistic one-sided error machines in place of nondeterministic machines show $PrSPACE_1(s) \subseteq PrSPACE_0(s)$, and hence they are equal. **Q.E.D.**

This result can be generalized to log-space alternating machines, but we now have two-sided error [28].

The simulation in the above proofs can be modified so that the simulating machine N halts with probability 1. However, N is no longer zero-error. The technique will be introduced in section 6.

Probabilistic Simulating of Alternation. Consider how a probabilistic machine can simulate an alternating machine M. We want our probabilistic machine to have bounded error. Suppose C is a configuration of M and $C \vdash (A, B)$. Let T_C, T_A, T_C denote the subtree at these nodes.

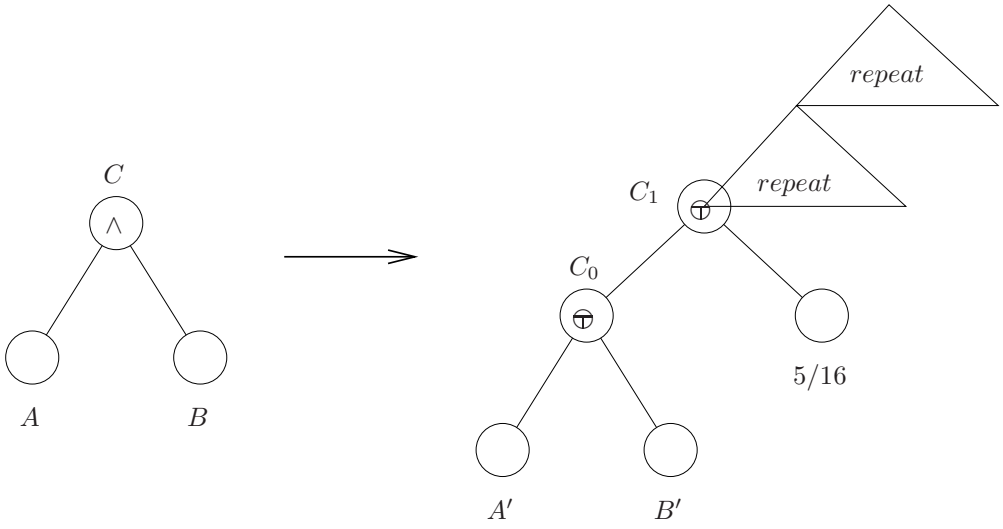


Figure 8.5: Simulating a $\wedge$-configuration C.

Inductively, assume that our recursive construction gives us probabilistic computation trees $T_{A'}$ and $T_{B'}$ (rooted at A' and B') which emulates T_A and T_B (respectively) with error gap

$$g_0 = [1/4, 3/4].$$

This means that if A accepts, then the value of A' is at least $3/4$ and if A rejects, the value of A' is at most $1/4$. Similarly for B and B'. Let us see how to carry the induction through.

CASE: C is a $\wedge$-configuration. Let C_0 be a $\oplus$-configuration such that $C_0 \vdash (A', B')$ (see figure 8.5). Then the value at C_0 has an error gap of

$$[5/8, 3/4].$$

This is because, if at least one of A' or B' rejects, then value of C_0 is at most $(1/4) \oplus 1 = 5/8$. And if both A' and B' accepts, the value is at least $3/4$. Then, we 'shift' the gap so that it is centered, by averaging it with the value $5/16$. This gives us a new gap (of half the size!)

$$g_0 = [15/32, 17/32].$$

We now use majority voting to boost this gap back to at least $[1/4, 3/4]$. We need to take the majority of $2k + 1$ votes, where k is a constant that can be computed.

CASE: C is a $\vee$-configuration. In this case, C_0 has error gap of

$$[1/4, 3/8].$$

Now we shift this gap by averaging it with $11/16$, yielding the gap g_0 above. We again boost it back to $[1/4, 3/4]$.

Note that if the original tree has height h and this procedure produces a tree of height $\tau(h)$, then

$$\tau(h) = k(\tau(h-1) + 2) \leq (2k)^h.$$

Of course, we need to make this recursive transformation something that can be carried out by a suitable probabilistic machine. This is left as an exercise. We have thus shown:

THEOREM 18

$$ATIME(t) \subseteq PrTIME_b(2^{O(t)}).$$

EXERCISE

Exercise 8.4.1: Let $t(n)$ be time-constructible. Determine the smallest function $t'(n)$ as a function of $t(n)$ such that

$$\text{co-}NTIME(t(n)) \subseteq PrTIME_b(t'(n)).$$

Hint: Simulate each step of a universal machine, but at each step, ensure bounded-error by using majority votes. □

Exercise 8.4.2: Let M be a probabilistic machine that runs in time $t(n)$ and which uses $\leq \log t(n)$ coin tosses along each computation path. Give an upper bound for the function $t'(n)$ such that $L(M) \in PrTIME_b(t'(n))$. □

End Exercise

8.5 Interactive Proofs

This section introduces interactive proofs [13] and Arthur-Merlin games [2]. The class of languages recognized in polynomial time by such machines is denoted IP. We show that IP contains two languages NONISO and #SAT. This gives some indication of the power of IP because NONISO is not known to be in NP, and #SAT is complete for PP. The main result is $IP = PSPACE$, which provides yet another characterizaiton of $PSPACE$.

Graph Non-Isomorphism. A motivating example is the graph isomorphism problem: given a pair $\langle G_0, G_1\rangle$ of graphs, decide if they are isomorphic,

$$G_0 \sim G_1. \tag{7}$$

We could use digraphs or bigraphs in the following. To be specific assume bigraphs (undirected graphs). Let $\mathcal{G}_n$ denote the set of bigraphs on the vertex set $\{1, 2, \ldots, n\}$. Formally define the languages

$$\begin{aligned} \text{ISO} &= \{\langle G_0, G_1\rangle \in \mathcal{G}_n^2 : n \geq 1, G_0 \sim G_1\}, \\ \text{NONISO} &= \{\langle G_0, G_1\rangle \in \mathcal{G}^2 : n \geq 1, G_0 \not\sim G_1\}. \end{aligned}$$

These are basically complementary languages. Let S_n denote the set of n-permutation (*i.e.*, permutations of $\{1, \ldots, n\}$). For $\pi \in S_n$, let $\pi(G_0)$ denote the graph G_0 when its vertices are renamed according to π: (i, j) is an edge of G_0 iff $(\pi(i), \pi(j))$ is an edge of $\pi(G_0)$. Then (7) holds iff there exists π such that $\pi(G_0) = G_1$. Call π a **certificate** for $\langle G_0, G_1\rangle$ in this case. Thus $\langle G_0, G_1\rangle \in$ ISO iff $\langle G_0, G_1\rangle$ has a certificate. This concept of certificate has two important properties:

- (Succinctness) The certicate π has size polynomial in n.
- (Verifiability) There is a deterministic polynomial-time algorithm V to decide if a given π is a certificate for $\langle G_0, G_1\rangle$.

These two properties characterize languages in NP. Hence, ISO $\in NP$. We can next try to improve our upper bound on ISO (is it in P ?) or prove a lower bound (is it NP-hard?). Both of these questions are open. Another related upper bound question (is it in co-NP) is also open.

It easily follows NONISO $\in$ co-NP. Unfortunately, co-NP does not have a characterization by certificates. Although certificates are not necessarily easy to find, they are easy to verify. In this sense, they have practical utility. We next introduce a generalization of certificates verifiability, and eventually show that NONISO is verifiable in this more general sense.

Concept of Interactive Proofs. We generalize certificate verifiability in two ways: first, we allows the verifying algorithm V to be probabilistic, and second, we allow interaction between the verifying algorithm with another algorithm called the "prover", denoted P. Thus there are two communicating processes (sometimes called *protocols*), an *interactive prover* P and an *interactive verifier* V which are Turing machines that send each other messages,

$$m_0, m_1, m_2, \ldots.$$

Message m_i is written by V if i is even, and by P if i is odd, and these are written on a common worktape. We assume some convention for each process to indicate that it is done writing its message (say, by entering a special state) and for some external agent to prompt the other process to continue. The computation ends when V answers YES or NO. The input is originally on V's input tape. We place complexity bounds on V alone. Thus the time and space in a (V, P)-computation refer solely to the time and space incurred by V alone. We place no restriction

on P (which need not even have to be computable), except that P must respond to each message of V in finite time. For instance, suppose if we say that V accepts in polynomial time, and P turns out to write exponentially long messages, then V will not be able to read the long messages of P. Intuitively, V is sceptical about what the process P is communicating to it, and needs to be "convinced" (with high probability). For any input w, let $\Pr(V, P, w)$ be the probability that V accept. We will assume that V halt on every computation path, thus avoiding any discussion of probability intervals. Languages will be defined with respect to V alone: writing

$$\Pr(V, w) := \sup_P \Pr(V, P, w),$$

then the language accepted by V is

$$L(V) := \{w : \Pr(V, w) > 1/2\}.$$

Say V has **bounded-error** if, in addition to the preceding requirements, we have $\Pr(V, w) \geq 2/3$ or $\Pr(V, w) \leq 1/3$ for all input w, The class IP comprises those languages that are accepted by bounded-error polynomial-time verifiers.

Interactive Verifier for Graph Non-Isomorphism. We want to describe an interactive verifer V such that $L(V) = \text{NONISO}$. Here is a well-known V_0 from the literature:

INPUT: STRING w
1. Reject unless $w = \langle G_0, G_1\rangle \in \mathcal{G}_n^2$, $n \geq 1$.
2. Randomly generate an n-permutation π and a binary bit b.
3. Let $H \leftarrow \pi(G_b)$.
4. Send message $m_0 = \langle H, G_0, G_1\rangle$. This message asks P whether $H \sim G_0$ or $H \sim G_1$.
5. (Pause for P to reply with message m_1)
6. If $b = m_1$ answer YES, else answer NO.

Note that V_0 concludes in two message rounds (sends and receives a message). Assume $w = \langle G_0, G_1\rangle$. There are two cases to consider.

- $w \in \text{NONISO}$: We claim $\Pr(V_0, w) = 1$. To see this, suppose P_0 is the prover who sends the message $m_1 = c$ such that $H \sim G_c$. Since c is unique, V_0 always answer YES, so $\Pr(V_0, P_0, w) = 1$.
- $w \notin \text{NONISO}$: We wish to claim $\Pr(V_0, w) = 1/2$. Intuitively, an "honest prover" P_0 cannot distinguish whether the answer should be $H \sim G_0$ or $H \sim G_1$. It is reasonable for P_0 to flip a coin and answer $m_1 = 0$ and $m_1 = 1$ with equal probability. This will establish our claim. But suppose we have a "dishonest prover" P_1 whose goal is to mislead V_0 into accepting w. P_1 knows something about about π and b it may be able to mislead V_0. For instance, if P_1 knows the value of b, then it will always fool V_0. How can we be sure that such information has not leaked in our definition of message m_0? This justification is non-trivial (see [20, p. 175]) and may be based on the so-called Principle of Deferred Decisions.

This example points out that informal descriptions of interactive proofs (with suggestive language such as "V is convinced", "P knows", etc) can be tricky to formalize. For this reason, we prefer to view interactive proofs as choice computations. The idea is this: we can combine V and P into one choice machine denoted, loosely,

$$\text{M} = \text{“}V + P\text{”},$$

where the states of M is the disjoint union of the states of V and P (so each state of M may be classified as a P-state or a V-state). We will let the choice function at each V-state q be $\gamma(q) = \oplus$. But what about P? We need to simulate all possible behavior for P (recall that we define $\Pr(V, w)$ as the maximization of $\Pr(V, P, w)$). This is not hard (we can essentially make all possible choices for the message). Furthermore, we let the choice function at each P-state q be $\gamma(q) = \vee$. Thus M is a $\{\oplus, \vee\}$-machine. Unfortunately, there are two issues.

One issue is that, although P is powerful, it seems that we do not want it to know about the coin tosses of V. Such a verifier is said to use "private coins". The formulation "$\text{M} = V + P$" apparently use "public coins". As noted, the use of public coins in V_0 above would be disastrous for the NONISO protocol above. Verifiers with private coins seems more powerful. It turns out, for polynomial-time computations, a verifier with private coins can be simulated by one with public coins, at the cost of of two extra rounds [14]:

$$IP[k] \subseteq AM[k + 2]. \tag{8}$$

The parameters k and $k + 2$ bound the number of message rounds; the full explanation for this notation is given below,

The second issue is this: the 0/1-message m_1 returned by the prover is not easily modeled by $\oplus$- and $\vee$-choices alone. The ability to pass a 0/1-message from P to V seems more powerful than simply allowing V to ask P question and receiving a YES/NO answer. For instance, V upon receipt of the Boolean message m_1, can trivially compute the negation of m_1. But a $\{\oplus, \vee\}$-computation cannot trivially negate the value at a node. Thus it seems we need a $\{\oplus, \vee, \neg\}$-machine (equivalently, a $\{\oplus, \vee, \wedge\}$-machine) to efficiently simulate an interactive proof. But this would make the interactive prover for NONISO uninteresting (NONISO is trivially accepted by a $\wedge$-machine in polynomial-time.) It turns out $\neg$ can be avoided, but this is a non-trivial result. We can avoid all these complications of interactive proofs by using the Arthur-Merlin formulation of Babai and Moran. The advantage, besides its simplicity, is the direct connection to choice computation.

Arthur-Merlin Games. Let M be an $\{\oplus, \vee\}$-machine, and $\pi = (C_0, C_1, \ldots, C_m)$ be a computation path of M. A computation sequence

$$\pi' = (C_i, C_{i+1}, \ldots, C_j), \quad (1 \leq i \leq j \leq m)$$

is called a $\oplus$**-round** (or **Arthur round**) if it contains at least one $\oplus$-configuration but no $\vee$-configurations. Notice that π' could contain deterministic configurations. Similarly, π' is called a $\vee$**-round** (or **Merlin round**) if we interchange $\oplus$ and $\vee$. We say π has k **rounds** if π can be divided into k subpaths,

$$\pi = \pi_1; \pi_2; \cdots; \pi_k \quad (k \geq 1)$$

where ";' denotes concatenation of subpaths such that π_i is an Arthur round iff π_{i+1} is a Merlin round. Note that k is uniquely determined by π. The definition generalizes in a natural way to $k = 0$ and $k = \infty$. We say M is a k**-round** *Arthur-Merlin game* if

- M has bounded error and runs in polynomial time
- every computation path of M has at most k rounds in which the first round (if any) is an Arthur round

A k**-round** *Merlin-Arthur game* is similarly defined, with the roles of Arthur and Merlin interchanged. Let

$$AM[k] = \{L(\mathrm{M}) : \mathrm{M} \text{ is an Arthur-Merlin game with at most } k \text{ rounds}\}$$

The class $MA[k]$ is similarly defined using Merlin-Arthur games instead. Of course, we can generalize this to $AM[t(n)]$ and $MA[t(n)]$ where $t(n)$ is a complexity function.

We can identify[4] Arthur with the verifier V, and Merlin with the prover P, of interactive proofs. We can similar define the classes $IP[k]$ and $IP[t(n)]$ accepted by interactive proofs in k or $t(n)$ rounds. This is the notation used in (8).

So it turns out that the apparent gap between interactive provers (V, P) and choice machines is non-existent. But this result is non-trivial since the pair (V, P) can communicate as in true parallelism. As discussed in Chapter 1, the choice mechanism is, in general, weaker than true parallelism.

8.5.1 Arthur-Merlin Game for Graph Non-Isomorphism.

The new idea for checking non-isomorphism is as follows. Let $G_0, G_1 \in \mathcal{G}_n$. Consider the set

$$\texttt{ALIKE}_1(G_0, G_1) = \{H : H \sim G_0 \text{ or } \ H \sim G_1\}.$$

Intuitively, $|\texttt{ALIKE}_1(G_0, G_1)|$ is either $n!$ or $2(n!)$, depending on whether $G_0 \sim G_1$ or not. Unfortunately, this is not always true and relates to the notion of an automorphism: we call $\pi \in S_n$ an **automorphism** of G if $\pi(G) = G$. Of course, the identity **1** permutation is always an automorphism, but this is the trivial case. We initially assume that G_i $(i = 0, 1)$ has only the trivial automorphism, so that $|\texttt{ALIKE}_1(G_0, G_1)| = n!$ or $2(n!)$,

Example: Let $n = 3$ and G_0 has the single edge $(1, 2)$ and G_1 has the single edge $(2, 3)$. Thus the transposition $(1, 2)$ is an automorphism of G_0. We see that the set $\texttt{ALIKE}_1(G_0, G_1) = \{G_0, G_1, G_2\}$ where G_2 has the single edge $(1, 3)$. So $|\texttt{ALIKE}_1(G_0, G_1)| = 3 < 3!$. Thus, our assumption that the G_i's have only the trivial automorphism is essential.

Continuing, suppose we randomly pick $H \in \mathcal{G}_n$, then there is some constant c such that $\Pr\{H \in \texttt{ALIKE}_1(G_0, G_1)\}$ is c or $2c$, depending on whether $G_0 \sim G_1$ or not. This probabilistic gap is the basis for recognizing NONISO. However, the constant c is exponentially small in n since $|\mathcal{G}_n| = 2^{\binom{n}{2}}\, n!$. We need to modify this gap by a hashing trick: the idea is to map $\mathcal{G}_n$ into a smaller set of size $\Theta(n!)$.

[4] As in the legend of King Arthur, the magician Merlin is more powerful than Arthur. Merlin, as the $\vee$-player, can use existential guesses (which is magically correct). Arthur, all too human, can only roll his dice and take his chances.

Let us understand the problem more generally: we have a set C whose size is either $n!$ or $2(n!)$. You want to computationally determine whether it is $n!$ or $2(n!)$. We cannot count explicitly since $n!$ is too large. Without loss of generality, let $C \subseteq \mathbb{B}^m$ (bit strings of length m). Here $\mathbb{B} = \{0,1\}$ is the Boolean field of two elements. If $n!$ is comparable to $|\mathbb{B}^m| = 2^m$, say $3(n!) < 2^m < 4(n!)$, then we can randomly sample elements $x \in \mathbb{B}^m$ and test if $x \in C$ (we assume testing membership in C is easy). If $|C| = 2(n!)$, probability that $x \in C$ is greater than $1/2$. If $|C| = n!$, then the probability is less than $1/3$. This gap implies that our problem is in BPP. In our application, $n!$ is not comparable to 2^m but exponentially smaller. So we define a hash function $h : \mathbb{B}^m \to \mathbb{B}^k$ where $n!$ is comparable to $|\mathbb{B}^k| = 2^k$. If h is $1-1$ when restricted to C, then the previous BPP-algorithm will work. Unfortunately, we do not know how to find such an h efficiently. The next lemma say that if h is a random hash function, we can achieve much the same result.

LEMMA 19 (BOPPANA) *Let B be a $k \times m$ Boolean matrix, and let*

$$h_B : \mathbb{B}^m \to \mathbb{B}^k$$

be defined by $h_B(x) = B \cdot x$ where all arithmetic is in $\mathbb{B}$. Fix $C \subseteq \mathbb{B}^m$ and write $h_B(C) = \{h_B(x) : x \in C\}$. Assume $c = |C|/2^k$ and $k \geq 2$. If $z \in \mathbb{B}^k$ and $B \in \mathbb{B}^{m \times k}$ are both random then

$$\Pr\{z \in h_B(C)\} > c - \frac{c^2}{2} = c\left(1 - \frac{c}{2}\right).$$

Proof. We will show that for all $x \neq y$,
(a) $\Pr\{z = h_B(x)\} = 1/2^k$, and
(b) $\Pr\{z = h_B(x) = h_B(y)\} \leq 1/4^k$.
The lemma then follows by the inclusion-exclusion principle,

$$\Pr\{z \in h_B(C)\} \geq \sum_{x \in C} \Pr\{z = h_B(x)\} - \sum_{\{x,y\} \in \binom{C}{2}} \Pr\{z = h_B(x) = h_B(y)\}.$$

(a) There are two cases for showing $\Pr\{z = h_B(x)\} = 1/2^k$: fix $x = (x_1, \ldots, x_m)$. If $x = \mathbf{0}_m$ (an m-vector of 0's) then $\Pr\{z = h_B(x)\} = \Pr\{z = \mathbf{0}_k\} = 1/2^k$. Otherwise, we may assume $x_1 \neq 0$. Let B_i denote the ith column of B. First fix $z' \in \mathbb{B}^k$; then for any choices of $B_2, \ldots, B_m$ there is a unique choice of B_1 such that $Bx = z'$. Thus

$$\Pr\{Bx = z'\} = \sum_{B_2, \ldots, B_m} 2^{-mk} = 2^{(m-1)k} 2^{-mk} = 2^{-k}. \tag{9}$$

If z is also random, then $\Pr\{Bx = z\} = \sum_{z'} 2^{-k} \Pr\{Bx = z'\} = 2^{-k}$. This proves (a).

(b) There are also two cases in showing $\Pr\{z = Bx = By\} \leq 1/4^k$. First, suppose $x = \mathbf{0}_m$. Then

$$\begin{aligned} \Pr\{z = Bx = By\} &= \Pr\{z = \mathbf{0}_k, By = \mathbf{0}_k\} \\ &= \Pr\{z = \mathbf{0}_k\} \Pr\{By = \mathbf{0}_k\} \\ &= 2^{-k} 2^{-k}, \end{aligned}$$

where the last equality uses part (a). Next suppose $x \neq \mathbf{0}_m$ and, by symmetry, $y \neq \mathbf{0}_m$. In this case, since $x \neq y$, there is some $1 \leq i < j \leq m$ such that

$$M = \begin{bmatrix} x_i & y_i \\ x_j & y_j \end{bmatrix}$$

is non-singular. To see this, without loss of generality, assume $i = 1, j = 2$ and $x_i = 1, y_i = 0$. Then we can choose $y_2 = 1$, and it does not matter what x_2 is. For any fixed $z' \in \mathbb{B}^k$ and any choice of columns $B_3, \ldots, B_m$, there is a constant $k \times 2$ matrix C such that the equation $z' = Bx = By$ can be rewritten as

$$[B_1 | B_2] M = C.$$

Since M is invertible, this uniquely determines B_1, B_2. There are $2^{(m-2)k}$ choices for $B_3, \ldots, B_m$ and hence $\Pr\{z' = Bx = By\} = 4^{-k}$. Again, when z is randomly chosen, this leads to $\Pr\{z = Bx = By\} = 4^{-k}$. This proves (b). **Q.E.D.**

The next idea is to get rid of the assumption that G_i has only the trivial automorphism. We use elementary facts of group theory. For any digraph G, let $\mathtt{aut}(G)$ denote the automorphism group of G. It is standard to look at the cosets of $\mathtt{aut}(G)$: these are sets of the form

$$\pi \circ \mathtt{aut}(G) = \{\pi \circ \sigma : \sigma \in \mathtt{aut}(G)\}$$

for each $\pi \in S_n$. These cosets form a partition of S_n. Here is the standard argument: (1) It is clear that S_n is the union of these cosets since $\mathbf{1} \in \mathtt{aut}(G)$. (2) $\pi \circ \mathtt{aut}(G)$ and $\pi' \circ \mathtt{aut}(G)$ are either disjoint or equal: for, if $\pi \circ \sigma = \pi' \circ \sigma'$ for some $\sigma, \sigma' \in \mathtt{aut}(G)$, then $\pi = \pi' \circ \sigma' \circ \sigma^{-1}$ so that $\pi \in \pi' \circ \mathtt{aut}(G)$. Then $\pi \circ \mathtt{aut}(G) \subseteq \pi' \circ \mathtt{aut}(G)$; reversing the argument implies $\pi \circ \mathtt{aut}(G) = \pi' \circ \mathtt{aut}(G)$.

We also note that each coset C has size $|\mathtt{aut}(G)|$ since $\pi^{-1}C = \mathtt{aut}(G)$ for some π. It follows that the number d of distinct cosets must divide $|S_n| = n!$. In fact, $d = n!/|\mathtt{aut}(G)|$. If the distinct cosets are C_i $(i = 1, \ldots, d)$, choose a representative $\pi_i \in S_n$ for each i such that $\pi_i \mathtt{aut}(G) = C_i$. Now consider the

$$\mathtt{iso}(G) = \{\pi_i(G) : i = 1, \ldots, d\}$$

It is clear that $\mathtt{iso}(G)$ is a complete list of all the graphs isomorphic to G. Moreover, it is not hard to check that $\mathtt{aut}(\pi_i(G)) = \pi_i \mathtt{aut}(G) \pi_i^{-1}$ [pf: $\sigma(\pi_i(G)) = \pi_i(G)$ iff $\pi_i^{-1}\sigma\pi_i(G) = G$].

It follows that the set

$$|\mathtt{iso}(G) \times \mathtt{aut}(G)| = n! \tag{10}$$

So if we count each graph $H \in \mathtt{iso}(G)$ with multiplicity equal to $|\mathtt{aut}(G)|$, the size of the multiset $\mathtt{iso}(G)$ would be $n!$. Equivalently, we "tag" each H with one of its automorphisms π, and count the corresponding set $\mathtt{iso}(G) \times \mathtt{aut}(G)$ set instead of $\mathtt{iso}(G)$. This tagging idea will provide a suitable replacement for $\mathtt{ALIKE}_1(G_0, G_1)$: define $\mathtt{ALIKE}(G_0, G_1)$ to be

$$\mathtt{ALIKE}(G_0, G_1) := \{\langle H, \pi \rangle : \pi(H) = H \text{ and } H \sim G_0 \text{ or } H \sim G_1\}.$$

LEMMA 20

$$|\mathtt{ALIKE}(G_0, G_1)| = \begin{cases} n! & \text{if } G_0 \sim G_1, \\ 2 \cdot n! & \text{if } G_0 \not\sim G_1. \end{cases} \tag{11}$$

Proof. If $G_0 \sim G_1$, then this is just a restatement of (10). If $G_0 \not\sim G_1$, then the $\mathtt{ALIKE}(G_0, G_1)$ is the disjoint union of $\mathtt{iso}(G_0) \times \mathtt{aut}(G_0)$ and $\mathtt{iso}(G_1) \times \mathtt{aut}(G_1)$. **Q.E.D.**

We are ready to prove that NONISO belongs to AM.

THEOREM 21 NONISO $\in AM[2]$.

Proof. Assume each element of $\mathcal{G}_n$ is given by a string in $\mathbb{B}^{\binom{n}{2}}$. Hence each element of $\mathtt{ALIKE}(G_0, G_1)$ can be represented by a string in $\mathbb{B}^m$ where $m = \binom{n}{2} + \lceil n \lg n \rceil$. Choose $k = \lceil \lg(n!) \rceil + 2$. As in Boppana's lemma, let $z \in \mathbb{B}^k$ and $B \in \mathbb{B}^{m \times k}$ be randomly chosen. Let us define $c := 2(n!)/2^k$. Note that if $G_0 \sim G_1$, then $|\mathtt{ALIKE}(G_0, G_1)| \leq n!$ and

$$\Pr\{z \in h_B(\mathtt{ALIKE}(G_0, G_1))\} \leq \frac{n!}{2^k} = c/2.$$

On the other hand, if $G_0 \not\sim G_1$ and applying lemma 19 with $C = \mathtt{ALIKE}(G_0, G_1)$ yields

$$\Pr\{z \in h_B(\mathtt{ALIKE}(G_0, G_1))\} \geq c\left(1 - \frac{c}{2}\right)$$

where $|C|/2^k = c$. Since $c \leq \frac{1}{2}$, we have

$$\Pr\{z \in h_B(\mathtt{ALIKE}(G_0, G_1))\} \geq 3c/4.$$

This gives rise to the following $\{\textcircled{\exists}, \vee\}$-machine M_0:

INPUT: $\langle G_0, G_1 \rangle \in \mathcal{G}_n^2$.
1. Randomly choose B and z, as above.
2. Existentially choose a bit b, $H \in \mathcal{G}_n$ and two n-permutations π, σ.
3. Answer YES if $\pi(H) = H$ and $\sigma(H) = G_b$ and $z = h_B(H)$; else Answer NO.

This machine has probability gap $(c/2, 3c/4)$. It does not accept NONISO yet because the gap is not an error gap (which would contain $\frac{1}{2}$). We need to "shift" this gap by modifying M_0 as follows: begin by tossing a coin to spawn 2 branches. On one branch, perform the computations of M_0. On the other, perform a probabilistic computation whose valuation is exactly $1 - (5c/8)$. Let us see how the second branch can be accomplished. First, we can compute the binary representation of c in polynomial time in a straightforward manner (in fact, $O(n^3 \log^2 n)$ time suffices). Next, compute the binary representation of $1 - (5c/8)$. Once this is available, we can make a series of coin tosses to achieve a valuation of exactly $p = 1 - (5c/8)$. In general, let the binary representation of a fraction $0 \leq p \leq 1$ is $0.p_1 p_2 \ldots p_k$ for some k. Note that $1 - (5c/8)$ is of this form. Then following randomized algorithm accepts with probability exactly equal to p:

L_0 : for i=1 to k
Randomly choose labels L_0 or L_1
L_1 : if $p_i = 1$ then Answer YES else Answer NO.

Note that the modified machine M_1 has the error gap

$$\left(\frac{c}{2}, \frac{3c}{4}\right) \oplus \left[1 - \frac{5c}{8}, 1 - \frac{5c}{8}\right] = \left(\frac{1}{2} - \frac{c}{16}, \frac{1}{2} + \frac{c}{16}\right).$$

In fact, $c/16 \geq 2^{-6}$. Thus M_1 is an $AM[2]$-game. **Q.E.D.**

Note that using (8) and the original interactive verifier V_0, we only infer NONISO $\in AM[4]$. Hence this direct proof yields a sharper result.

8.5.2 Arithmetization of Quantified Boolean Formulas

The next two subsections uses a technique called "arithmetization" of quantified Boolean formulas. We recall the notion of a **quantified Boolen formula** F: In the base case, F is a Boolean variable x or the constants 0 or 1. Inductively, if F_1, F_2 are quantified Boolean formulas, then F has the form in the left column of the following table,

F	$\|F\|$
$x, 0, 1$	1
$\neg F_1$	$1 + \|F_1\|$
$(F_1 \vee F_2)$	$\|F_1 + \|F_2\|\|$
$(F_1 \wedge F_2)$	$\|F_1\| + \|F_2\|$
$(\exists x)[F_1]$	$1 + \|F_1\|$
$(\forall x)[F_1]$	$1 + \|F_1\|$

F_1, F_2 are quantified Boolean formulas. The **length** $|F|$ of F is given in the right column of the preceding table.

In the following, we simply say "formulas" for quantified Boolean formulas. Parenthesis or brackets may be dropped from formulas if this does not lead to ambiguity. Any occurence of the variable x in F_1 is said to be **bound** in $(\exists x)F_1$ and $(\forall x)F_1$. An occurence of the variable x is **free** if it is not bound. When we write $F = F(x_1, \ldots, x_n)$, this means that any variable that occurs free in F is among $x_1, \ldots, x_n$ (but it does not mean that each x_i actually occurs free in F). In this case, when we write $F(a_1, \ldots, a_n)$, it means we replace each free occurence of x_i in F by a_i. When F has no free variables, it is called a **quantified Boolean sentence** (or simply, "sentence"). We assume that the reader knows what it means for sentence to be true. If $F(x_1, \ldots, x_n)$, then F is **valid** if for every choice of Boolean values $a_i \in \mathbb{B}$, $F(a_1, \ldots, a_n)$ (which is a sentence) is true. A formula is **quantifier-free** when it does not have any quantifiers ($\forall$ or $\exists$). We say F is in **prenex form** if it has the form

$$(Q_1 x_1)(Q_2 x_2) \cdots (Q_m x_m)[\phi]$$

where $Q_i \in \{\forall, \exists\}$ and ϕ is quantifier-free.

Arithmetization. Suppose $F = F(x_1, \ldots, x_n)$ is a formula. Its "arithmetization" is an integer polynomial $\widetilde{F}(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ defined as follows:

- (Base Case) If $F \in \{0, 1, x_1, \ldots, x_n\}$, then $\widetilde{F} = F$.
- (Induction) If F_1, F_2 be quantified Boolean formulas, and $F(x_1, \ldots, x_n)$ has the form in the left column, then $\widetilde{F}$ has the form on the right column:

F	$\widetilde{F}$
$\neg F_1$	$1 - \widetilde{F_1}$
$(F_1 \wedge F_2)$	$\widetilde{F_1} \otimes \widetilde{F_2} = \widetilde{F_1}\widetilde{F_2}$
$(F_1 \vee F_2)$	$\widetilde{F_1} \oplus \widetilde{F_2} = 1 - (1 - \widetilde{F_1})(1 - \widetilde{F_2})$
$(\forall x)[F_1]$	$\widetilde{F_1}(x_1, \ldots, x_{n-1}, 0) \otimes \widetilde{F_1}(x_1, \ldots, x_{n-1}, 1)$
$(\exists x)[F_1]$	$\widetilde{F_1}(x_1, \ldots, x_{n-1}, 0) \oplus \widetilde{F_1}(x_1, \ldots, x_{n-1}, 1)$

For example, if $F = (\neg x)(x \vee y)$ then

$$\widetilde{F} = (1-x)(x \oplus y) = x + y - x^2 - 2xy + x^2 y. \tag{12}$$

Here we use a common convention where x_1, x_2, x_3 are synonymous with x, y, z. Recall that $\mathbb{B} = \{0, 1\}$. Our polynomials have two simple properties:
(i) If each $a_i \subseteq \mathbb{B}$ value then $\widetilde{F}(a_1, \ldots, a_n) \in \mathbb{B}$.
(ii) $\widetilde{F}(a_1, \ldots, a_n) = 1$ iff the sentence $F(a_1, \ldots, a_n)$ is true.

Define $\mathcal{P}_n$ to be the set of polynomials $\widetilde{P}$ that arise from formulas F over the variables $x_1, \ldots, x_n$. Alternatively, $\mathcal{P}_n$ is the smallest set of integer polynomials such that (1) $\{0, 1, x_1, \ldots, x_n\} \subseteq \mathcal{P}_n$, and (2) if $p, q \in \mathcal{P}_n$ then $1 - p \in \mathcal{P}_n$ and $pq \in \mathcal{P}_n$. Viewing a polynomial p as a sum of **terms** (or monomials) where each term has the form $t = c \prod_{i=1}^n x_i^{e_i}$ $(c \neq 0)$. Let $e = (e_1, \ldots, e_n)$ and $|e| := \sum_{i=1}^n e_i$. Then we call $e = (e_1, \ldots, e_n)$ the **exponent**, $|e|$ the **degree**, $c \in \mathbb{Z}$ is the **coefficient** and $\prod_{i=1}^n x_i^{e_i}$ $(e_i \geq 0)$ is a **power product** of the term t. We also write x^e for the power product $x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}$. The **degree** of p, denoted $\deg(p)$, is defined as the maximum of the degrees of its terms.

Let $\deg_i(p)$ be the largest exponent $d \geq 0$ such that x_i^d divides a term in p Then the **maximum degree** of p is the vector

$$\texttt{MaxDeg}(p) = (e_1, \ldots, e_n)$$

where $e_i = \deg_i(p)$ for each i. We say p is **principal** when $x^{\texttt{MaxDeg}(p)}$ occurs in p; then the coefficient of $x^{\deg(p)}$ in p is called the **leading coefficient** of p. Define[5]

$$M(e) := \frac{|e|!}{e!}$$

where $e! := e_1! e_2! \cdots e_n!$ and $|e| = \sum_{i=1}^n e_i$ as above. For instance if $e = (0, 2, 1, 2)$ then $M(e) = \frac{5!}{0!2!1!2!} = 30$. We note a basic identity of multinomials:

LEMMA 22 *Let $g \in \mathbb{N}^n$ and $|g| = a + b$ $(a, b \in \mathbb{N})$. Let $I = I(g, a, b)$ be the set of all pairs of the form (e, f) such that $e, f \in \mathbb{N}^k$, $|e| = a, |f| = b$ and $e + f = g$.*

$$M(g) = \sum_{(e,f) \in I} M(e) M(f). \tag{13}$$

Proof. This proof exploits a counting interpretation of $M(g)$: $M(g)$ is the number of ways to color a set of $|g|$ elements with k colors. Let A, B be disjoint sets of a and b elements, respectively. The number of ways to color $A \cup B$ with k colors is therefore $M(g)$. But for each $(e, f) \in I$, there are $M(e)$ ways to k-color A and $M(f)$ ways to k-color B. Combining them, this gives rise to $M(e)M(f)$ ways to k-color $A \cup B$. Let $C(e, f)$ be the k-colorings of $A \cup B$ that is associated in this way with (e, f). Note that $C(e, f) \cap C(e', f') = \emptyset$ for $(e, f) \neq (e', f')$, and hence

$$M(g) \geq \sum_{(e,f) \in I} M(e) M(f).$$

It is also easy to see that every k-coloring of $A \cup B$ is a member of $C(e, f)$ for some (e, f). Hence the inequality is in fact an equality. **Q.E.D.**

A **homogeneous polynomial** is one in which every term has the same degree. For any polynomial p which does not involve x_0, id $d = \deg(p)$, we define its **homogeneous version** $\widehat{p}$ to be the result of replacing each term $c_e x^e$ in p by the term $c_e x^e x_0^{d-|e|}$. For instance, if $p(x, y) = p(x_1, x_2)$ is the polynomial in (12), then

$$\widetilde{p} = (x_0 - x_1)(x_0 x_1 + x_0 x_2 - x_1 x_2) = x_0^2 x_1 + x_0^2 x_2 - x_0 x_1^2 - 2 x_0 x_1 x_2 + x_1^2 x_2.$$

If $e = (e_1, \ldots, e_n)$ and $|e| \leq d$, define

$$M_d(e) = \frac{d!}{(d - |e|)! e_1! \cdots e_n!}$$

For each term with exponent e in a polynomial p of degree d, the corresponding term in $\widetilde{p}$ has exponent $e' = (d - |e|, e_1, \ldots, e_n)$. Then $M_d(e)$ is just $M(e')$. It is easy to see that

$$M(e) \leq M_d(e). \tag{14}$$

[5] $M(e)$ is also known as a **multinomial**, but this is terminology should be distinguished from the "monomial" terminology.

LEMMA 23 *Let $p \in \mathcal{P}_n$.*
(i) p is principal.
(ii) The leading coefficient of p is ± 1, and its constant term is either 0 or 1.
(iii) If $c_e x^e$ is a term in p, then $|c_e| \leq M_d(e)$ where $d = \deg(p)$.
(iv) For any formula F, the coefficients of $\widetilde{F}$ has bit size $O(|F| \log |F|)$.

Proof. (i) and (ii) are immediate.
(iii) (Basis) If $|e| = 0$, then $M_d(e) = 1$, and so the result holds when $p = 0$ or $p = 1$. If $|e| = 1$, then $M_d(e) \geq 1$, and thus it holds when $p = x_i$ or $p = 1 - x_i$. (Induction) Suppose $p, q \in \mathcal{P}_n$. By induction, the result is true for $1 - p$, and we want to verify the result for pq. Consider their homogeneous versions $\widehat{p}, \widehat{q}$. For $e = (e_0, \ldots, e_n) \in \mathbb{N}^{n+1}$ let $X^e = x_0^{e_0} x_1^{e_1}, \ldots, x_n^{e_n}$ and α_e be the coefficient of X^e in $\widehat{pq}$. Similarly, let β_e and γ_e be the coefficients of X^e in $\widehat{p}$ and in q, respectively. Then for any $e \in \mathbb{N}^{n+1}$,

$$\begin{aligned} \alpha_e &= \sum_{f+g=e} \beta_f \gamma_g, \\ |\alpha_e| &\leq \sum_{(f,g)\in I} |\beta_f| \cdot |\gamma_g| \\ &\leq \sum_{(f,g)\in I} M(f) M(g) \\ &= M(e), \end{aligned}$$

by the previous lemma. But if $e = (e_0, e_1, \ldots, e_n)$ and $e' = (e_1, \ldots, e_n)$ then $M(e)$ is just $M_d(e')$, as desired.
(iv) Let F be a formula and $d \leq \deg(\widetilde{F})$. If $e \in \mathbb{N}^n$ then from (iii), the coefficient of x^e in $\widetilde{F}$ has bit size at most $\log M_d(e) \leq \log(d!)$. The lemma follows since $d = O(|F|)$. **Q.E.D.**

8.5.3 *IP* contains *PP*.

In order to prove $PP \subseteq IP$, it is enough to show that a *PP*-complete problem (under many-one polynomial-time reducibility, say) is in *IP*. The same approach is used in the next section to show that $PSPACE \subseteq IP$, so this proof serves as warm-up. Our main result here is.

THEOREM 24 #SAT*is in IP.*

Let F be a quantifier-free formua over $x_1, \ldots, x_n$. From the definition of $\#F$ as the number of satisfying assignments to the formula F, we see that

$$\#F = \sum_{a_1=0}^{1} \sum_{a_2=0}^{1} \cdots \sum_{a_n=0}^{1} \widetilde{F}(a_1, \ldots, a_n).$$

We define the following polynomials for $i = 0, \ldots, n$:

$$F_i(x_1, \ldots, x_i) := \sum_{a_{i+1}\in\mathbb{B}} \sum_{a_{i+2}\in\mathbb{B}} \cdots \sum_{a_n\in\mathbb{B}} \widetilde{F}(x_1, \ldots, x_i, a_{i+1}, a_{i+2}, \ldots, a_n). \tag{15}$$

Notice that

- $F_n(x_1, \ldots, x_n) = \widetilde{F}(x_1, \ldots, x_n)$.
- For $i = 1, 2, \ldots, n$, $F_{i-1}(x_1, \ldots, x_{i-1}) = F_i(x_1, \ldots, x_{i-1}, 0) + F_i(x_1, \ldots, x_{i-1}, 1)$.
- $F_0 = \#F$.

Clearly, each $F_i \in PP_n$ and $\deg(F_i) \leq \deg F_n$. In particular, the coefficients of F_i has bit size $O(|F| \log |F|)$. The idea of the algorithm is to reduce the search for F_i to F_{i+1}, and hence reducing the search for $\#F = F_0$ to F_n (which we know). Moreover, since F_i is a i-variate polynomial, by random substitution for $i - 1$ of these variables, we can reduce the checking to a univariate problem.

An *IP*-Algorithm for #SAT. Let the input for #SAT be $\langle F, k\rangle$ where F is a Boolean formula. The following algorithm is to accept iff $\#F \geq k$. Assume the polynomial $\widetilde{F}$ has n variables and degree $\leq d = |F|$.

1. Construct the polynomial
$$F_n = \widetilde{F}(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n].$$
We represent F_n as an expression using the inductive rules in the definition of $\widetilde{F}$. Note that we do not expand F_n into an explicit sum of terms, which would be prohibitive. In the following, we refer to $F_i(x_1, \ldots, x_i)$ $(i = 0, \ldots, n-1)$, defined as in (15). Note that these F_i's will are never computed.
Let $S \subseteq \mathbb{Z}$ be any finite set of size at least $8dn$. For instance, we may let $S = \{0, 1, \ldots, 2^{\lceil lg(8dn)\rceil} - 1\}$. In the following, whenever we choose a random number r, it will be taken from S.

2. Stage 0: Existentially guess some k_0 where $k \leq k_0 \leq 2^n$. Our main task is to verify $F_0 = k_0$ (recall $F_0 = \#F$). How can we do this, seeing that we do not have F_0? We this indirectly, by guessing the coefficients of a polynomial $G_1(x) \in \mathbb{Z}[x]$ of degree $\leq d = |F|$. We intend[6] $G_1(x)$ to be equal to $F_1(x)$. Unlike the multivariate F_n, we can afford to represent the univariate G_1 as a sum of its terms. We then check if $G_1(0) + G_1(1) = k_0$. If this fails, we answer NO (our guess is wrong). If our guess were correct, then $G_1(0) + G_1(1) = F_0$ and our main task is next reduced to to verifying $G_1(x) = F_1(x)$, which is addressed in stage 1.

3. Stage $i = 1, \ldots, n-1$: Inductively assume that, at the beginning of stage i, we have a univariate polynomial $G_i(x)$ and a sequence of elements $r_1, \ldots, r_{i-1} \in \mathbb{Z}$. Each r_j is randomly chosen a previous stage (in fact, stage j). The task for this stage is to verify that
$$G_i(x) = F_i(r_1, \ldots, r_{i-1}, x). \tag{16}$$
Here, $G_i(x)$ is explicitly represented by its sequence of coefficients, but $F_i(r_1, \ldots, r_{i-1}, x)$ is implicitly represented by $F_n(x_1, \ldots, x_n)$ (from step 2) and the values $r_1, \ldots, r_{i-1}$.
To verify (16), we *existentially guess* (the coefficients of) a polynomial $G_{i+1}(x) \in \mathbb{Z}[x]$ of degree $\leq d$. Then we *randomly choose* $r_i \in S$. Notice that the modes for choosing $G_{i+1}(x)$ and for choosing r_i are different. Again, $G_{i+1}(x)$ is intended to be $F_{i+1}(r_1, \ldots, r_i, x)$. Since $F_i(r_1, \ldots, r_i) = F_{i+1}(r_1, \ldots, r_i, 0) + F_{i+1}(r_1, \ldots, r_i, 1)$, instead of (16), we verify that
$$G_i(r_i) = G_{i+1}(0) + G_{i+1}(1). \tag{17}$$
If this fails, we answer NO. Otherwise, we proceed to the next stage.

4. Stage n: from stage $n-1$, we have inherited $G_n(x)$ and $r_1, \ldots, r_{n-1}$. We now need to verify $G_n(x) = F_n(r_1, \ldots, r_{n-1}, x)$ (cf. equation (16)). We randomly[7] guess $r_n \in S$, and answer YES if $G_n(r_n) = F_n(r_1, \ldots, r_{n-1}, r_n)$, otherwise answer NO. This final check is possible because we can easily evaluate $F_n(r_1, \ldots, r_{n-1}, r_n)$ from our representation of $F_n(x_1, \ldots, x_n)$ and $r_1, \ldots, r_n$.

Correctness. We prove that this procedure accepts #SAT. One direction is easy: suppose $\langle F, k\rangle \in \#\text{SAT}$. We may assume that all our existential guesses are correct. In particular, k_0 is correctly chosen to be $\#F$, and at each stage, the guessed polynomial $G_i(x)$ is indeed equal to $F_i(r_1, \ldots, r_{i-1}, x)$. Under these assumptions, regardless of the choice of the r_i's, we always answer YES in stage n. In fact, this shows that we accept $\langle F, k\rangle$ with no pessimistic acceptance error. Moreover, there is a computation tree of polynomial height, since each G_i has degree at most $d = |F|$ and its coefficients are $O(|F| \log |F|)$ bits.

In the harder direction, where $\langle F, k\rangle \notin \#\text{SAT}$, we have $\#F \neq k_0$ for all choices of $k_0 \geq k$. Fix any k_0. We will show that $\Pr(E_n) \geq 3/4$ where E_n be the event that we answer NO in stage n. For $i = 0, \ldots, n-1$, define E_i to be the event that we enter stage $i+1$ with
$$G_{i+1}(x) \neq F_{i+1}(r_1, \ldots, r_i, x). \tag{18}$$

For events A and B, we have $\Pr(A) \geq \Pr(A|B)\Pr(B)$. Iterating this,

$$\begin{aligned}\Pr(E_n) &\geq \Pr(E_n|E_{n-1})\Pr(E_{n-1}) \\ &\geq \Pr(E_n|E_{n-1})\Pr(E_{n-1}|E_{n-2})\Pr(E_{n-2})\end{aligned}$$

[6] Mnemonic: think of the G's as "guesses" for the F's.
[7] We could actually perform this step in deterministic polynomial time.

$$\begin{aligned} &\geq \cdots \\ &\geq \Pr(E_n|E_{n-1})\Pr(E_{n-1}|E_{n-2})\cdots\Pr(E_1|E_0)\Pr(E_0) \\ &= \Pr(E_0)\prod_{i=1}^{n}\Pr(E_i|E_{i-1}). \end{aligned} \tag{19}$$

In stage 0, the fact that $G_1(0)+G_1(1)=k_0$ and $\#F = F_1(0)+F_1(1) \neq k_0$ implies that $G_1(x) \neq F_1(x)$. Hence $\Pr(E_0)=1$.

In stage 1, we $\exists$-guessed $G_2(x)$ of degree $\leq d$, and randomly choose a number r_1 such that $G_1(r_1) = G_2(0) + G_2(1)$. Since the total degrees of G_1, F_1 are $\leq d$, the fact that $G_1(x) \neq F_1(x)$ implies follows that there are at most d choices of $r \in \mathbb{Z}$ such that $G_1(r) = F_1(r)$. This follows from a simple fact of algebra that a non-zero polynomial $p(x) \in \mathbb{Z}[x]$ of degree d has at most d zeros. Here, $p(x) = G_1(x) - F_1(x)$. Thus $\Pr(E_1|E_0) \geq 1-(d/p)$.

The same argument works for stage $i+1$ ($i = 1, \ldots, n-2$): assuming (18), $\Pr\{G_{i+1}(r_{i+1}) \neq F_{i+1}(r_1, \ldots, r_i, r_{i+1})\} \leq d/p$. Thus $\Pr(E_{i+1}|E_i) \geq 1-(d/p)$. It is also true that $\Pr(E_n|E_{n-1}) \geq 1-(d/p)$. From (19), we conclude that

$$\begin{aligned} \Pr(E_n) &\geq \left(1-\frac{d}{p}\right)^n \\ &>_1 (e^{-2d/p})^n \\ &>_2 1-\frac{2nd}{p} \\ &> 3/4 \qquad (\text{since } \ p > 8dn). \end{aligned}$$

See the Appendix for the inqualities $>_1$ and $>_2$. This proves the correctness of our *IP*-algorithm. Since #SAT is *PP*-complete (say, under Karp-reducibility) and *IP* is clearly closed under Karp-reducibility, we have shown:

Theorem 25

$$PP \subseteq IP.$$

Neither of the inclusions $NP \subseteq IP$, and co-$NP \subseteq IP$ are obvious because of the requirement of bounded error in *IP*. But as $3SAT$ is trivially Karp-reducible to #SAT, it follows that $NP \subseteq IP$. Since *IP* is closed under complementation, this also means that co-$NP \subseteq IP$.

Corollary 26

$$NP \cup \text{co-}NP \subseteq IP.$$

Remark: In the above proof, the equality of polynomials is always taken in the abstract mathematical sense, not in terms of their representations. Thus $F(a_1, \ldots, a_n)$ is equal to 0 or 1 when the $a_i \in \mathbb{B}$. On the other hand, we need to address the issue of representation when we construct explicit polynomials (e.g., F_n or the guessed G_i's).

8.5.4 *IP = PSPACE*

In Chapter 7, we proved $PrA\text{-}TIME(t) \subseteq ATIME(t \log t)$. Thus,

$$IP \subseteq PrA\text{-}TIME(n^{0(1)}) \subseteq ATIME(n^{O(1)}) = PSPACE.$$

We now prove the converse, $PSPACE \subseteq IP$, a result of Shamir [30]. The algebraic technique for showing $PP \subseteq IP$. in the previous section will be extended. In particular, we now show that a particular *PSPACE*-complete language belongs to *IP*. Define the set of **valid quantified Boolean formulas** to be

$$\text{QBF} = \{F : F \text{ is a valid formula}\}.$$

This can be viewed as a language, after choosing some standard encoding of formulas. We may assume $F \in$ QBF are in prenex form.

Lemma 27 QBF *is PSPACE-complete.*

The proof is left as an Exercise. Hence our desired result amounts to showing an *IP*-algorithm for QBF. We initially try to imitate the previous proof, as this helps to locate the new difficulties. Let F be a formula. Unlike the previous proof, F may now have quantifiers. We may assume that

$$F = (Q_1 x_1 Q_2 x_2 \cdots Q_n x_n)[\phi] \tag{20}$$

where $\phi = \phi(x_1, \ldots, x_n)$ is quantifier-free. Define formulas F_i for $i = 0, \ldots, n$ as follows:

$$\begin{aligned}
F_n(x_1, \ldots, x_n) &= \phi \\
F_{n-1}(x_1, \ldots, x_{n-1}) &= (Q_n x_n)[F_n] \\
&\vdots \\
F_{i-1}(x_1, \ldots, x_{i-1}) &= (Q_i x_i \cdots Q_n x_n)[\phi] = (Q_i x_i)[F_i] \\
&\vdots \\
F_1(x_1, x_2) &= (Q_3 x_3 Q_4 x_4 \cdots Q_n x_n)[\phi] \\
F_1(x_1) &= (Q_2 x_2 Q_3 x_3 \cdots Q_n x_n)[\phi] \\
F_0() &= F
\end{aligned}$$

If $\widetilde{F}_i$ is the arithmetization of F_i, then we have

$$\widetilde{F}_i(x_1, \ldots, x_i) = \begin{cases} \widetilde{F}_{i+1}(x_1, \ldots, x_i, 0) \otimes \widetilde{F}_{i+1}(x_1, \ldots, x_i, 1) & \text{if } Q_i = \forall \\ \widetilde{F}_{i+1}(x_1, \ldots, x_i, 0) \oplus \widetilde{F}_{i+1}(x_1, \ldots, x_i, 1) & \text{if } Q_i = \exists \end{cases}$$

The problem is that the degree of $\widetilde{F}_i$ is double that of $\widetilde{F}_{i+1}$. Hence the degree of $\widetilde{F}$ may be exponential in $|F|$.

A Linearization Quantifier. To solve this problem, we introduce a new kind of quantifier denote L. The class of quantified Boolean formulas are now enlarged to include this new quantifier. In particular, for any formula $\phi(x_1, \ldots, x_n, x)$ and variable x, the following is also a formula,

$$F = Lx[\phi].$$

Furthermore, its arithmetization is defined via

$$\widetilde{F} := (1 - x)\widetilde{\phi}(x_1, \ldots, x_n, 0) + x\widetilde{\phi}(x_1, \ldots, x_n, 1).$$

Note the following properties:

- If $a \in \mathbb{B}$, then $\widetilde{F}(x_1, \ldots, x_n, a) = \widetilde{\phi}(x_1, \ldots, x_n, a)$.
- $\widetilde{F}$ has the same set of free variables as $\widetilde{\phi}$ (namely, $x_1, \ldots, x_n, x$).
- $\widetilde{F}$ is linear in x.

The last two properties are quite unlike the arithmetization of the other two quantifiers. Indeed, the name of the L-quantifier is taken from the last property. An Exercise below shows that the linearlization transformation

$$\phi(x_1, \ldots, x_n, x) \mapsto (1 - x)\phi(x_1, \ldots, x_n, 0) + x\phi(x_1, \ldots, x_n, 1)$$

amounts to replacing any power of x by a plain x in every term of the polynomial $\phi(x_1, \ldots, x_n, x)$. We now transform the formula F in (20) to a new formula H,

$$\begin{aligned}
H &:= (Q_1 x_1)(Lx_1 Q_2 x_2)(Lx_1 Lx_2 Q_3 x_3) \cdots (Lx_1 Lx_2 \cdots Lx_{n-1} Q_n x_n)[\phi] \\
&= (\overline{Q}_1 y_1 \overline{Q}_2 y_2 \cdots \overline{Q_m} y_m)[\phi]
\end{aligned}$$

where $\overline{Q}_i \in \{L, \forall, \exists\}$ and $y_i \in \{x_1, \ldots, x_n\}$. Here $m = \binom{n+1}{2}$ is the number of quantifiers in H. These m quantifiers are placed into n groups, as indicated by the matched pairs of parentheses. In each group (reading from right to left), a normal quantifier Q_i is followed by $i - 1$ linearization operators to ensure that the degrees of the remaining variables are linear. As usual, let

$$\begin{aligned}
H_m(x_1, \ldots, x_m) &= \phi \\
H_{m-1}(x_1, \ldots, x_m) &= (\overline{Q}_m y_m)[H_m] \\
H_{m-2}(x_1, \ldots, x_m) &= (\overline{Q}_{m-1} y_m)[H_{m-1}] = (\overline{Q}_{m-1} y_{m-1} \overline{Q}_m y_m)[H_m] \\
&\cdots \\
H_1(x_1) &= (\overline{Q}_2 y_2)[H_2] \\
H_0() &= (\overline{Q}_1 y_1)[H_1] = H
\end{aligned}$$

Note that $H_i = H_i(x_1, \ldots, x_{n(i)})$ for some $n(i)$. The H_i's are formulas, but in our algorithm, we will simulate their arithmetized[8] versions $\widetilde{H}_i$. In particular, we will make $\exists$-guesses of $G_i = G_i(x_1, \ldots, x_{n(i)})$ which are intended to be equal to $\widetilde{H}_i$. The critical fact is that

$$\deg(\widetilde{H}_i) \leq |F|^2.$$

To see this, let $d_i = \deg(\widetilde{H}_i)$. Then we have $d_m \leq |F|$. However, d_{m-1} could be as large as $(d_m)^2$. However, the next $n-1$ linearization quantifiers reduces this degree back to at most $|F|$. This pattern is repeated for each group of linearization quantifiers that follows the next $\forall$ or $\exists$ quantifier (Q_{m-1}). We now see why the linearization quantifier is an antidote to the exponential degree of F_i's.

Here now is the *IP*-algorithm for QBF. The description will be abbreviated where similarities to the previous *IP*-algorithm for #SAT is clear.

1. The input is a quantified Boolean formula F. Let H be the corresponding linearized version, and the formulas $H_0, \ldots, H_m$ is defined as above. Again, we assume a finite set S of size $|S| \geq$???. Whenever we choose a random number, it will be taken from S.

2. Stage 0: We guess $G_1(x)$, intended to be equal to $\widetilde{H}_1(x)$. Now $H = H_0 = (\overline{Q}_1 x_1)[H_1]$ and $\overline{Q}_1 = \forall$ or $\exists$. If $\overline{Q}_1 = \forall$, then we check if $G_1(0) = G_1(1) = 1$. If $\overline{Q}_1 = \exists$, then we check if $G_1(0) = 1$ or $G_1(1) = 1$. If the check fails, we answer NO; else we go to Stage 1.

3. Stage i ($i = 1, \ldots, m-1$). Inductively, we have random values $r_1, \ldots, r_{n(i)-1}$ and $G_i(x)$. Intuitively, $G_i(x)$ is meant to be $H_i(r_1, \ldots, r_{n(i)-1}, x)$. Now $H_i = (\overline{Q}_{i+1} y_{i+1})[H_{i+1}(x_1, \ldots, x_{n(i+1)})]$.

   Note that if $\overline{Q}_{i+1} = L$ then $n(i+1) = n(i)$; otherwise $\overline{Q}_{i+1} = \forall$ or $\exists$ and $n(i+1) = n(i) + 1$. Existentially guess $G_{i+1}(x)$. If $\overline{Q}_{i+1} \neq L$ then we randomly choose $r_{n(i+1)-1} = r_{n(i)}$. Regardless of $\overline{Q}_{i+1}$, the intention is that $G_{i+1}(x)$ is $H_i(r_1, \ldots, r_{n(i+1)-1}, x)$.

   We consider three cases:
   (i) If $\overline{Q}_i = \forall$, we check if $G_i(r_{n(i)}) = G_{i+1}(0) \otimes G_{i+1}(1)$
   (ii) If $\overline{Q}_i = \exists$, we check if $G_i(r_{n(i)}) = G_{i+1}(0) \oplus G_{i+1}(1)$
   (iii) If $\overline{Q}_i = \exists$, we check if $G_i(r_{n(i)}) = G_{i+1}(r_{n(i)})$

4.

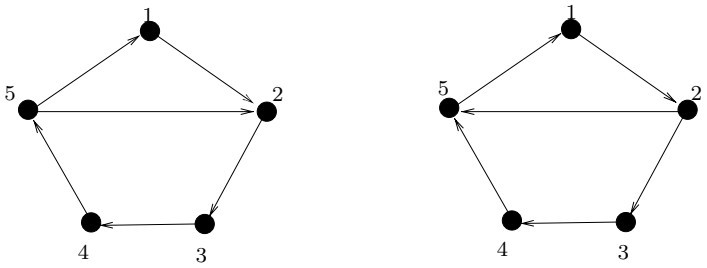
Notes. Our approach to interactive proofs via choice computation is unconventional, but it provides a more satisfactory and general foundation than the customary treatment. This allows interactive proofs to seen to be a natural part of a wide spectrum of computational choices. A survey on interactive proofs may be found in [24]. The place of ISO in the complexity landscape is explored in the monograph of Köbler, Schöning and Torán [17]. Another closely related question is GRAPH AUTOMORPHISM: Given G, does it have a non-trivial automorphism?

EXERCISE

Exercise 8.5.1: Show that *IP* is closed under Karp- and Cook-reducibility. □

Exercise 8.5.2: This exercise helps you gain some facility with the group theoretic ideas in the NONISO $\in$ *IP* proof. Let $V = V_n = \{1, \ldots, n\}$ and S_n be the set of permutations on V_n. The trivial permutation is denoted $\mathbf{1}_n$ (or simply $\mathbf{1}$). Write the composition of $\sigma, \sigma' \in S_n$ in the form of a product $\sigma\sigma'$, instead of $\sigma \circ \sigma'$.
(i) Let $2 \leq k \leq n$. If $\{a_1, \ldots, a_k\} \subseteq \binom{V}{k}$, then $(a_1, \ldots, a_k) \in S_n$ denotes the permutation which takes each a_i to $a_{(i+1 \bmod k)}$, called a **cyclic permutation**. Two special cases are $k = 2$ or $k = n$. Then $(a_1, \ldots, a_k)$ is **transpose** or a **Hamiltonian permutation**, respectively. Two cyclic permutations $(a_1, \ldots, a_k)$, $(b_1, \ldots, b_\ell)$ are disjoint if $a_i \neq b_j$ for all i, j. For instance, $(132)(45) = (45)(132)$ is a product of two disjoint cycles. The order of writing disjoint products does not matter. Show that every non-trivial permutation is a product of disjoint permutations.
(ii) Let G_0 be the digraph shown in Figure **??**. etermine $\texttt{iso}(G_0)$ and $\texttt{iso}(G_1)$. What the sizes of these two sets?
(iii) Determine $\texttt{aut}(G_0)$ and $\texttt{aut}(G_1)$. What the sizes of these two sets? □

[8] We really ought to write "$\widetilde{H_i}$". But this is uglier than the "$\widetilde{H}_i$" which we will use.



(a) G_0 (a) G_1

Figure 8.6: Two labeled digraphs G_0, G_1.

Exercise 8.5.3: If $p \in \mathcal{P}_n$ has degree d, then there is a constant $C > 0$ such that the magnitude of each coefficient of p is at most

$$C\left(\frac{d}{\exp(1)}\right)^{n+(1/2)}\left(\frac{n+1}{d}\right)^{n+1}.$$

This sharpens the $d!$ bound used in the text. □

Exercise 8.5.4: Prove that QBF is *PSPACE*-complete. Since this proof can be found in the literature, we will enforce some originality in your solution by asking you to use the *same* framework as the proof of Cook's theorem in Chapter 3. **Hint:** The additional idea you need is found in the proof of Savage's theorem: if $C \vdash^{2m} C'$ (i.e., there is an $2m$-step path from C to C' then $C \vdash^m C''$ and $C'' \vdash^m C'$ for some C''. □

Exercise 8.5.5: Show that for any polynomial $\phi(x_1, \ldots, x_n, x)$, the linearization transformation

$$\phi(x_1, \ldots, x_n, x) \mapsto (1-x)\phi(x_1, \ldots, x_n, 0) + x\phi(x_1, \ldots, x_n, 1)$$

amounts to replacing any power of x by a plain x in every term of the polynomial $\phi(x_1, \ldots, x_n, x)$. □

Exercise 8.5.6: Are there reasons to believe that co-$NP \subseteq AM$, based on NONISO $\subseteq AM[2]$? Derive nontrivial consequences under the assumption co-$NP \subseteq AM$. □

Exercise 8.5.7: If $L \in AM[k]$ then L is accepted by an Arthur-Merlin game in $k+1$ rounds with zero-error acceptance. □

Exercise 8.5.8: How does one amplify error gaps for languages in $AM[k]$? □

End Exercise

8.6 Markov Chains and Space-bounded Computation

We want to study computations by space-bounded probabilistic machines. The behavior of such computations can be analyzed in terms of finite[9] *Markov chains*. We develop the needed results on Markov chains (see also the appendix in this chapter). For further reference on Markov chains, see [16, 10].

The main result of this section is

Theorem 28 *For all $s(n) \geq \log n$,*

$$PrSPACE(s) \subseteq DSPACE(s^2)$$

Notice that this result is yet another strengthening of Savitch's theorem! We follow the proof of Borodin, Cook and Pippenger [6]; Jung [15] independently obtained the same result using different techniques[10]. This result

[9] *i.e.*, discrete time, homogeneous Markov processes, with finitely many states.

[10] Borodin, Cook and Pippenger uses redundant arithmetic techniques while Jung uses modular arithmetic techniques. Borodin, Cook and Pippenger states the result in a stronger form (in terms of circuit depth, see chapter 10), but it has essentially the proof to be presented here.

improves earlier simulations by Gill [12] (which uses exponential space) and by J. Simon [31] (which uses $s(n)^6$ space).

A sequence of non-negative real numbers $(p_1, p_2, \ldots, p_i, \ldots)$ is *stochastic* if the sum $\sum_{i\geq 1} p_i = 1$; it is *substochastic* if $\sum_{i\geq 1} p_i \leq 1$. A matrix is stochastic (resp., substochastic) if each row is stochastic (substochatic). In general, stochastic sequences and stochastic matrices may be denumerably infinite although we will only consider finite matrices. An n-state *Markov process* (or *Markov chain*) is characterized by an $n \times n$ stochastic matrix $A = (p_{i,j})_{i,j=1}^n$. Call A the *transition matrix*. The states of A will be called *Markov states*, as distinguished from machine states. We interpret this chain as an n-state finite automaton where $p_{i,j}$ is the probability of going from state i to state j. For any integer $k \geq 0$, the kth power $A^k = (p_{i,j}^{(k)})_{i,j}^n$ of A is defined inductively: A^0 is the identity matrix and $A^{k+1} = A \cdot A^k$. It is easy to check the product of stochastic matrices is stochastic; hence each A^k is stochastic. Clearly $p_{i,j}^{(k)}$ denotes the probability of a transition from state i to state j in exactly k steps.

Markov states admit a straight forward combinatorial classification. From the transition matrix $A = (p_{i,j})_{i,j=1}^n$ of the Markov chain, construct the Boolean matrix $B = (b_{i,j})_{i,j=1}^n$ where $b_{i,j} = 1$ iff $p_{i,j} > 0$. We view B as the adjacency matrix of a directed graph G_A, called the *underlying graph* of the matrix A. We may form the transitive closure $B^* = (b_{i,j}^*)_{i,j=1}^n$ of B (see chapter 2, section 6). As usual, define states i and j to be *strongly connected* if

$$b_{i,j}^* = b_{j,i}^* = 1.$$

This is easily seen to be an equivalence relationship and the equivalence classes form the *(strongly connected) components* of G_A. These strongly connected components in turn are related by the *reachability relation*: if C and C' are components, we say C *can reach* C' if there are states $i \in C$ and $j \in C'$ such that $b_{i,j}^* = 1$. It is easy to see that this definition does not depend on the choice of i and j. Furthermore, if C can reach C' and C' can reach C then $C = C'$. Thus the reachability relation induces an acyclic graph F on the components where F has an edge from C to C' iff C can reach C'. Those components C that cannot reach any other components are called *essential components* and the states in them known as *essential states*. The other components are called *inessential components* and their members known as *inessential states*.[11] We say state i is *absorbing* if $p_{i,i} = 1$. Such a state is clearly essential and forms a component by itself. A Markov chain is *absorbing* if all essential states are absorbing.

The above classification depends only on the underlying graph G_A. Let us now classify states by their stochastic properties. These notions properly belong to a subarea of probability theory called renewal theory. We introduce an important concept in renewal theory: let $f_{i,j}^{(n)}$ denote the probability that, starting from state i, we enter state j for the first time after n steps. We call these $f_{i,j}^{(n)}$ the *first entrance probabilities*. Write $f_i^{(n)}$ for $f_{i,i}^{(n)}$. It is not hard to see that for $n = 1, 2, \ldots$,

$$f_{i,j}^{(n)} = p_{i,j}^{(n)} - \sum_{k=1}^{n-1} f_{i,j}^{(k)} p_{j,j}^{(n-k)}$$

or,

$$p_{i,j}^{(n)} = \sum_{k=0}^{n} f_{i,j}^{(k)} p_{j,j}^{(n-k)} \tag{21}$$

where we conventionally take

$$f_{i,j}^{(0)} = 0, \quad p_{i,j}^{(0)} = \delta_{i,j}.$$

Here $\delta_{i,j}$ is Kronecker's delta function that assumes a value of 1 or 0 depending on whether $i = j$ or not. The sum

$$f_{i,j}^* = \sum_{n=1}^{\infty} f_{i,j}^{(n)}$$

clearly denotes the probability of ever reaching state j from i. Let f_i^* abbreviate $f_{i,i}^*$. We now define a state to be *recurrent* if $f_i^* = 1$ and *nonrecurrent* if $f_i^* < 1$.

LEMMA 29 *An inessential state is nonrecurrent.*

Proof. By definition, if state i is inessential, there is a finite path from i to some state outside the component of i. Then $f_i^* \leq 1 - c$ where $c > 0$ is the probability of taking this path.

Q.E.D.

[11] The reader should be aware that the classification of Markov states are not all consistent in the literature. The essential/inessential distinction is due to Chung [8]. His terminology is justified in the sense that every chain has at least one essential component; but it also seems to reflect an attitude in probabilistic theory that the most interesting phenomena occur in the essential components. This is unfortunate because we will see that the inessential components are more interesting for us!

The converse does not hold in general (Appendix and Exercise). But in the case of finite Markov chains, essential states are recurrent. To show this result, we proceed as follows: let $g_{i,j}^{(n)}$ denote the probability of the event $G_{i,j}^{(n)}$ that starting from state i we will visit state j at least n times. Note that $G_{i,j}^{(n+1)} \subseteq G_{i,j}^{(n)}$ and so we may define the limit

$$g_{i,j} := \lim_{n\to\infty} g_{i,j}^{(n)} = \Pr(\bigcap_{n=0}^{\infty} G_{i,j}^{(n)}).$$

It is not hard to see that $g_{i,j}$ is the probability that starting from state i we visit state j infinitely often. Again, let $g_{i,i}$ be abbreviated to g_i.

LEMMA 30
(i) $g_i = 1$ or 0 according as i is recurrent or not.
(ii) In a finite Markov chain, essential states are recurrent.

Proof. (i) Note that

$$g_i^{(n+1)} = f_i^* g_i^{(n)}.$$

Since $g_i^{(1)} = f_i^*$, we get inductively

$$g_i^{(n+1)} = (f_i^*)^n.$$

Taking limits as $n \to \infty$, we see that $g_i = 1$ if $f_i^* = 1$ and $g_i = 0$ if $f_i^* < 1$.
(ii) Let $E_i^{(n)}$ be the event that starting from state i, there are no returns to state i after n steps. Clearly

$$E_i^{(1)} \subseteq E_i^{(2)} \subseteq \cdots$$

and $E_i := \bigcup_{n\geq 0} E_i^{(n)}$ is the event that there are only finitely many returns. But

$$\Pr(E_i^{(n)}) \leq 1 - e$$

where $e > 0$ is the minimum probability that any state in the component of i can get to state i. (To see this,

$$\Pr(E_i^{(n)}) = \sum_j p_{i,j}^{(n)}(1 - f_{j,i}^*) \leq (1-e)\sum_j p_{i,j}^{(n)}$$

which is at most $1 - e$.) Hence $\Pr(E_i) \leq 1 - e < 1$. But $g_i = 1 - \Pr(E_i)$. Hence $g_i > 0$ and by part (i), $g_i = 1$. This means state i is recurrent. **Q.E.D.**

We now see that for finite Markov chains, the combinatorial classification of essential/inessential states coincides with the stochastic classification of recurrent/nonrecurrent states. The appendix describe some refined classifications.

The *stochastic completion* of $A = (p_{i,j})_{i,j=1}^n$ is the matrix $A^* = (p_{i,j}^*)_{i,j=1}^n$ where

$$p_{i,j}^* = \sum_{k=0}^{\infty} p_{i,j}^{(k)}$$

with the understanding that the sum is ∞ when it diverges. The completion operation is defined even if A is not a stochastic matrix.[12]

The entries of A^* has this natural interpretation:

LEMMA 31
(i) $p_{i,j}^*$ is the expected number of steps that the automaton spends in state j if it started out in state i.
(ii) Furthermore, if j cannot return to itself in one or more steps then $p_{i,j}^*$ is the probability that the automaton ever enters state j.

[12]The terminology is from in [6]. The notation $p_{i,j}^*$ is not to be confused with the limiting value of $p_{i,j}^{(k)}$ as $k \to \infty$. Unfortunately, a stochastic completion is no longer a stochastic matrix. This is obvious from the interpretation of $p_{i,j}^*$ as the expected number of steps in state j.

Proof. Interpretation (i) follows when we note $p_{i,j}^{(n)}$ is the expected fraction of time that the automaton spends in state j during nth unit time period, assuming that it started out in state i. For (ii), under the stated assumptions on state j, we see that $p_{i,j}^{(n)} = f_{i,j}^{(n)}$ and hence $p_{i,j}^* = f_{i,j}^*$. **Q.E.D.**

Let us introduce the following generating functions (see appendix) for state i:

$$\begin{aligned} F_i(s) &:= \sum_{n=0}^{\infty} f_i^{(n)} s^n \\ G_i(s) &:= \sum_{n=0}^{\infty} p_i^{(n)} s^n \end{aligned}$$

Using the relation (21), we see that

$$G_i(s) - 1 = F_i(s) G_i(s)$$

or,

$$G_i(s) = \frac{1}{1 - F_i(s)}$$

Now if we take the limit as $s \to 1^-$, the left hand side approaches $p_{j,j}^*$ and the right hand side approaches $\frac{1}{1-f_j^*}$. This proves

LEMMA 32

$$p_{j,j}^* < \infty \iff f_j^* < 1.$$

To relate this to other values of $p_{i,j}^*$, we have

LEMMA 33

$$p_{i,j}^* = \delta_{i,j} + f_{i,j}^* p_{j,j}^*.$$

Proof.

$$\begin{aligned} p_{i,j}^* &= \sum_{n=0}^{\infty} p_{i,j}^{(n)} \\ &= \delta_{i,j} + \sum_{n=1}^{\infty} \sum_{k=1}^{n} f_{i,j}^{(k)} p_{j,j}^{(n-k)} \\ &= \delta_{i,j} + \sum_{k=1}^{\infty} f_{i,j}^{(k)} \sum_{n=k}^{\infty} p_{j,j}^{(n-k)} \\ &= \delta_{i,j} + \sum_{k=1}^{\infty} f_{i,j}^{(k)} \sum_{n=0}^{\infty} p_{j,j}^{(n)} \\ &= \delta_{i,j} + f_{i,j}^* p_{j,j}^* \end{aligned}$$

Q.E.D.

COROLLARY 34 *For all i, j, if $f_{i,j}^* > 0$ then*

$$p_{i,j}^* < \infty \iff p_{j,j}^* < \infty.$$

We need one more basic fact [16].

LEMMA 35 *Let A be a square matrix such that $A^n \to 0$ as $n \to 0$, i.e., each entry of the nth power of A approaches zero as n approaches infinity. Then the matrix $I - A$ is nonsingular where I is the square matrix with the same dimensions as A. Moreover the infinite sum*

$$\sum_{n=0}^{\infty} A^n$$

converges, and this sum is given by

$$(I - A)^{-1} = \sum_{n=0}^{\infty} A^n.$$

Proof. We begin with the identity

$$(I - A)(I + A + A^2 + \cdots + A^n) = I - A^{n+1}.$$

Now the right hand side approaches I for large n. So for sufficiently large n, $\det(I - A^{n+1}) \neq 0$. This means $\det(I - A)\det(I + A + \cdots + A^n) \neq 0$. Thus $I - A$ is nonsingular, as asserted. So we may multiply both sides of the identity on the left with $(I - A)^{-1}$, giving a new identity

$$(I + A + A^2 + \cdots + A^n) = (I - A)^{-1}(I - A^{n+1}).$$

Now as $n \to \infty$, the left hand side approaches the infinite sum of the lemma and the right hand side approaches $(I - A)^{-1}$. Since the right hand side approaches a definite limit, so the left hand side approaches the same limit. **Q.E.D.**

For any transition matrix A, let B be obtained by deleting the rows and columns of A corresponding to essential states. Following Kemeny and Snell, we call B the *fundamental part* of A. Note that B is a substochastic matrix. Then after permuting the rows and columns of A, we have

$$A = \begin{pmatrix} B & T \\ 0 & C \end{pmatrix}$$

where '0' denotes a matrix of zeroes of the appropriate dimensions. Moreover, the nth power of A is

$$A^n = \begin{pmatrix} B^n & T_n \\ 0 & C^n \end{pmatrix}$$

where B^n, C^n are the nth powers of B, C (respectively) and T_n is some matrix whose form need not concern us. Hence, the stochastic completion

$$A^* = \begin{pmatrix} B^* & T_* \\ 0 & C^* \end{pmatrix}$$

where B^*, C^* are the stochastic completions of B, C (respectively). In our applications, we only need B^*. Note that the entries in C^*, T_* are 0 or ∞, by what we have proved.

From the above development, the entries $p_{i,j}^{(n)}$ in B^n satisfy the property $\sum_{n=0}^{\infty} p_{i,j}^{(n)} < \infty$. This means $p_{i,j}^{(n)} \to 0$ as $n \to \infty$. Hence $B^n \to 0$ as $n \to \infty$ and the preceding lemma shows that B^* converges to $(B - I)^{-1}$. Hence computing B^* is reduced to the following:

THEOREM 36 *Let A be the transition matrix of an absorbing chain. The stochastic completion of the fundamental part B of A can be computed in deterministic space $\log^2 n$ where B is n by n and each entry of B are rational numbers represented by a pair of n-bit binary number.*

The proof of this theorem requires several preparatory results that are interesting in their own right. Therefore we defer it to the next section. We are now ready to prove the main result (theorem 28) of this section. Although we only compute the fundamental part B^*, with a bit more work, one can compute all the remaining entries of the stochastic closure in the same complexity bounds (see [6]).

Proof of main result (theorem 28). Basically the proof amounts to reducing a space-bounded probabilistic computation to computing the stochastic closure of the fundamental part of an absorbing Markov chain.

Let M be a probabilistic machine accepting in space $s(n)$. We analyze the probability of M accepting an input w by considering the Markov chain whose (Markov) states correspond to those configurations of M on w using space at most $s(|w|)$. We introduce an extra Markov state. Number these Markov states from 1 to r, where we may assume that Markov state 1 is the initial configuration on input w, and r is the extra Markov state (viewed as a NO-configuration of M). The corresponding transition matrix is $A = (p_{i,j})_{i,j=1}^{r}$ where

$$p_{i,j} = \begin{cases} \frac{1}{2} & \text{if configuration } i \text{ non-uniquely derives configuration } j \\ & \quad \text{(i.e., } i \vdash (j,k) \text{ for some } k \neq j) \\ 1 & \text{if either configuration } i \text{ uniquely derives } j, \text{ i.e., } i \vdash (j,j) \\ & \quad \text{or if } i = j \text{ and } i \text{ is terminal} \\ 0 & \text{else.} \end{cases}$$

This is not quite all: how shall we treat state i if $i \vdash (j,k)$ where j and/or k uses more than $s(|w|)$ space? Now assign for such an i, $p_{i,r} = \frac{1}{2}$ or 1, depending on whether one or both of j,k use more than $s(|w|)$ space.

We derive from A an absorbing Markov chain with transition matrix B as follows: say a Markov state in A is **useful** if it is has a path to a YES-state. Clearly useful states are inessential, but some inessential states may not be useful. In B, we retain all the useful states of A and also their transition probabilities among themselves. We renumber the useful Markov states from 1 to some $m-1$ ($m < r$). In addition to the $m-1$ useful states inherited from A, B has two essential states, m and $m+1$. Basically, we collapse all essential YES-states into m and the remaining states in A (essential or not) are collapsed into $m+1$. States m and $m+1$ are both absorbing. More precisely, for each useful state $i = 1, \ldots, m-1$, if the sum of the transition probabilities into the YES-states is p then we set the $(i, m+1)$th entry $[B]_{i,m} := p$. If the sum of the transition probabilities from i to the non-YES and non-useful states is q then we make $[B]_{i,m+1} = q$. Also, we have

$$[B]_{m,m} = [B]_{m+1,m+1} = 1$$

We do one more small transformation: let now C be the matrix that is identical to B except that

$$[C]_{m,m} = 0, \quad [C]_{m,m+1} = 1.$$

So state m is now a transient state. For future reference, call C the *reduced transition matrix* (for input w). If D is the fundamental part of C (obtained by deleting the last row and last column of C) then by theorem 36, we can compute the stochastic completion D^* in $O(\log^2 m)$ space. Now m is $O(1)^{s(|w|)}$ and hence $O(\log^2 m) = O(s^2)$ space suffices.

Our 'interpretation' (lemma 31) of the entries of a stochastic completion suggests that the entry $[D^*]_{1,m}$ is the probability that starting out in state 1 we reach m (since state m cannot return to itself in 1 or more steps, by construction). It is instructive to carry out the proof that $[D^*]_{1,m}$ is indeed the least fixed point value $Val_\Delta(w)$ where Δ is the set $\{1, \ldots, r\}$ of configurations that uses space at most $s(|w|)$. A valuation V on Δ amounts to an r-vector $V = (v_1, \ldots, v_r)$ where v_i is the value of configuration i. (We may assume here that the values v_i are real numbers in $[0,1]$ rather than intervals.) The valuation operator τ_Δ is the linear transformation given by the transition matrix A, and $\tau_\Delta(V)$ is equal to $A \cdot V^T$ (V^T is the column vector obtained by transposing V). Let $V_0 = \tau_\Delta(V_\perp)$ be the row vector that assigns a 1 to the YES state and a zero to the NO state. (Note that V_0 is not stochastic in general.) The valuation $\tau^n_\Delta(V_0)$ is given by $V_n = A^n \cdot V_0^T$. We conclude: *$Val_\Delta(w)$ is the limiting value of the first component of $A^n \cdot V_0^T$, as $n \to \infty$.* Alternatively, if the set of YES-configurations are $S \subseteq \Delta$, then $Val_\Delta(w)$ is the limiting value of $\sum_{i \in S} [A^n]_{1,i}$.

It is not hard to see that our transformation of A to B does no harm and we have a slightly simpler picture: $Val_\Delta(w)$ is given by the limiting value of $[B^n]_{1,m}$ **Exercise:**.

The transformation from B to C is less obvious. Let us compare their nth powers, B^n and C^n, for each $n \geq 0$. The first $m-1$ columns of both powers are seen to be identical. We claim that the first $m-1$ entries in the mth column of B^n is equal to the corresponding entry in the sum $\sum_{i=1}^n C^i$: *for each $j = 1, \ldots, m-1$, and for all $n \geq 0$,* $[B^n]_{j,m} = \sum_{\ell=1}^n [C^\ell]_{j,m}$. In proof, this is clearly true for $n = 1$. For $n \geq 1$, we have

$$\begin{aligned}
[B^{n+1}]_{j,m} &= \sum_{k=1}^{m} [B^n]_{j,k}[B]_{k,m} \\
&= [B^n]_{j,m}[B]_{m,m} + \sum_{k=1}^{m-1} [B^n]_{j,k}[B]_{k,m} \\
&= [B^n]_{j,m} + \sum_{k=1}^{m-1} [C^n]_{j,k}[C]_{k,m} \\
&= [B^n]_{j,m} + [C^{n+1}]_{j,m} \\
&= \sum_{\ell=1}^{n+1} [C^\ell]_{j,m}
\end{aligned}$$

This essentially gives us the theorem.

There are several other details that we must defer to the next section: in particular we cannot afford to explicitly store the matrices A or C. Instead, they are represented 'procedurally' in the sense that each entry of such matrices can be obtained by invoking suitable subroutines. For instance, this means that our 'application' of theorem 36 is really not in the form as stated. We need to show that the techniques implicit in that theorem can be modified to accomodate the implicit representation of C. Another clarification is needed: to form matrix C, we need to

determine the useful states in A (one efficient way to detect such states uses the the original Savitch's theorem technique). Modulo these details, we are done with the proof.

EXERCISE

Exercise 8.6.1: (1-dimensional random walk)

Analyze the 1-dimensional random walk with parameter $0 < p < 1$.
(i) Show that the generating functions $G(s)$ for the probability $p_{0,0}^{(n)}$ $(n = 0, 1, \ldots)$ of returning to the origin in n steps is given by $G(s) = (1 - 4pqs^2)^{1/2}$ where $q = 1 - p$.
(ii) Conclude that each Markov state is recurrent if and only if $p = \frac{1}{2}$.
(iii) In case $p = \frac{1}{2}$, show that the mean recurrence time is infinite. **Hint:** Use the relation that the generating function $F(s)$ for first re-entrance probability $f_{0,0}^{(n)}$ is related to $G(s)$ by $G(s) - 1 = F(s)G(s)$ and the mean recurrence time is given by $\lim_{s \to 1^{-1}} \frac{dF(s)}{ds}$. □

Exercise 8.6.2: (Erdös, Feller, Pollard)

Let $(f_0, f_1, \ldots)$ be a stochastic vector with $f_0 = 0$ and period is equal to 1. (The period is the largest d such that $f_n > 0$ implies d divides n.) Now define $p_0 = 1$ and $p_n = \sum_{k=0}^{n} f_k p_{n-k}$. Prove that $\lim_{n \to \infty} p_n = \frac{1}{\mu}$ where $\mu = \sum_{n=0}^{\infty} n f_n$. **Hint:** Note that the relation between pair of sequences $\{f_n\}$ and $\{p_n\}$ is identical with the relation between the first entrance probabilities $f_i^{(n)}$ and n-step transition probabilities $p_{i,i}^{(n)}$ for a fixed state i in section 3. □

Exercise 8.6.3: Define a transition matrix $A = (p_{i,j})_{i,j \geq 0}$ to be *doubly stochastic* if the row sums as well as column sums are equal to 1. Show that each space-bounded computation of a reversible probabilistic Turing machine gives rise to such a matrix. Study the properties of such machines.

□

END EXERCISE

8.7 Efficient Circuits for Ring Operations

By an *algebraic structure* we means a set A together with a finite set of partial functions f_i $(i = 1, \ldots, k)$ of the form

$$f_i : A^{\alpha(i)} \to A$$

where $\alpha(i) \geq 0$ are integers called the *arity* of f_i. We call f_i a *constant* if $\alpha(i) = 0$ and in this case, f_i is identified with an element of A. We write $(A; f_1, \ldots, f_k)$ for the algebraic structure. This is abbreviated to 'A' when the functions f_i are understood. In general, by an *operation f over an algebraic structure A* we mean a partial function $f : A^m \to A$ for some $m \geq 0$, where m is called the arity of f.

Example 2 a) Of course, for any set A, there is the trivial algebraic structure on A which no functions at all.
b) The integers $\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$ with the usual operations ($+$, $-$, $\times$ and $0, 1$) is an algebraic structure. It is, in fact, a unitary commutative ring (see below).
c) The rational numbers $\mathbb{Q}$, with the operations of $\mathbb{Z}$ but also including the division $\div$ operation, is an algebraic structure called a commutative field. Here division is a partial function since division by zero is not defined.
d) The set of all n-square matrices with entries from $\mathbb{Q}$ forms a matrix ring with the usual matrix operations of $+$, $-$ and $\times$.
e) Consider the Boolean algebra on two elements $(\{0, 1\}; \vee, \wedge, \neg, 0, 1)$ where $\vee, \wedge, \neg$ are interpreted as the usual Boolean operations.
f) A class of finite rings is $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ with usual arithmetic operations modulo n. In case n is a prime, $\mathbb{Z}_n$ is a field, also called $GF(p)$. The case $GF(2)$ has special interest. ■

We are mainly interested in computing over various unitary commutative rings R, henceforth simply called 'rings'[13]. Our goal is to show how operations in common rings can be implemented efficiently. A computational model that is appropriate for algebraic structures is circuits.

[13]Commutative rings are simply algebraic structures satisfying certain axioms. The student unfamiliar with rings simply need remember two main examples of such structures given above: the integers $\mathbb{Z}$ and the set of n-square matrices with rational number entries. So a ring comes with the five total operation $+, -, \times, 0, 1$ with the usual properties (inverse relation between plus and minus, associativity, commutativity, distributivity, and properties of 0 and 1) are satisfied. If one writes down these properties, they would constitute an axiomatization of unitary commutative rings (it is a good exercise to try this and compare your results with a standard algebra book). Here 'unitary' serves to to warn that, in general, rings are defined without assuming the existence of element 1.

The following definitions gather most of our notations and terminology related to circuits in one place. It will serve as reference beyond just the immediate concern of this section.

Definition 5 Let $(A; f_1, \ldots, f_k)$ be an algebraic structure.

(i) Any set Ω of operations over A obtained by functional composition from $f_1, \ldots, f_k$ is called a *basis* of A. In general Ω may have infinite cardinality.

(ii) A *circuit C for A over the basis Ω* a finite directed acyclic graph (called the *underlying graph* of C) with the following properties: A node with indegree 0 is called an *input* node, and if there are n input nodes, we label each with a distinct integer between 1 and n. The remaining nodes are called *gates* and are each labeled by a basis function $f \in \Omega$. If a gate is labeled by f, we say its *type* is f or, equivalently, it is called an *f-gate*. Each f-gate u has indegree exactly equal to the arity $\alpha(f)$ of f. Furthermore the incoming edges to u are labeled with distinct integers between 1 and $\alpha(f)$. So we may speak of the jth *incoming edge* of u for $j = 1, \ldots, \alpha(f)$.

(iii) Each node u of a circuit C is said to *compute* the function

$$\mathtt{res}_C(u) \; : \; A^n \to A$$

defined as follows: an input node u labeled by i computes the projection function $\mathtt{res}_C(u)(x_1, \ldots, x_n) = x_i$. An f-gate u computes the function

$$\mathtt{res}_C(u)(\overline{x}) = f(\mathtt{res}_C(u_1)(\overline{x}), \ldots, \mathtt{res}_C(u_m)(\overline{x}))$$

where $\overline{x} = (x_1, \ldots, x_n)$ and the ith incoming edge of u leads from node u_j $(j = 1, \ldots, m)$, m is the arity of f.

(iv) A *circuit family* (over the basis Ω) is an infinite sequence of circuits $\overline{C} = (C_n)_{n=0}^{\infty}$ such that each C_n is an n-input circuit over Ω.

(v) A *problem instance of size n* (over A) is a set P_n of functions $g \; : \; A^n \to A$ (so each $g \in P_n$ has arity n). An *aggregate problem* $P = (P_n)_{n=0}^{\infty}$ over A is an infinite sequence of problem instances P_n, each P_n of size n. When no confusion arises, we may omit the qualification 'aggregate'. Often, $P_n = \{f_n\}$ is a singleton set, in which case we simply write $P = (f_n)_{n \geq 0}$.

(vi) Let P_n be a problem instance of size n over A. A circuit C over A is said to *solve* or *realize* P_n if if C has n inputs and for each $g \in P_n$, there is a node $u \in C$ such that $\mathtt{res}_C(u) = g$. A circuit family $\overline{C} = (C_n)_{n=0}^{\infty}$ is said to *solve* a problem $P = (P_n)_{n=0}^{\infty}$ if C_n solves P_n for each n.

(vii) The *size* of a circuit C is the number of gates in the circuit[14]. The *size* of a circuit family $\overline{C} = (C_n)_{n=0}^{\infty}$ is the function $SIZE_{\overline{C}}$ where $SIZE_{\overline{C}}(n)$ is the size of C_n.

(viii) Two other complexity measures for circuits are as follows: the *depth* of C is the length of the longest path in C. The *width* of C is the maximum cardinality of an edge anti-chain[15] in C. As for the size-measure, we let $DEPTH_{\overline{C}}(n)$ and $WIDTH_{\overline{C}}(n)$ denote the depth and width of C_n, where $\overline{C} = (C_n)_{n \geq 0}$.

(ix) For any problem instance P_n, let $SIZE(P_n)$ denote the smallest sized circuit that realizes P_n. If $P = (P_n)_{n \geq 0}$ is an aggregate problem, the size function $SIZE_P$ is the function given by $SIZE_P(n) = SIZE(P_n)$. Similarly for

$$DEPTH(P_n), WIDTH(P_n), DEPTH_P, WIDTH_P.$$

(x) For any complexity function $f(n)$, let $SIZE(f)$ denote the family of aggregate problems $\{P \; : \; \forall n, SIZE_P(n) \leq f(n)\}$. Similarly for $DEPTH(f), WIDTH(f)$. We can extend this to simultaneous measures, for instance $SIZE - DEPTH - width(f_1, f_2, f_3)$.

(xi) For any non-negative integer k, a circuit family $\overline{C}$ is said to be NC^k if $SIZE_{\overline{C}}(n) = n^{O(1)}$ and $DEPTH_{\overline{C}}(n) = O(\log^k n)$. An aggregate problem is said to be NC^k if it can be realized by an NC^k circuit family.

■

[14] It is unfortunate that we have to use the word 'size' for problem instances as well as for circuits, both of which appear in the same context. Since the usage is well accepted and there seems to be no better alternative, we will continue this usage. But for emphasis, we could say 'circuit size' or 'problem size'.

[15] An *edge anti-chain* in a directed acyclic graph is a set of edges such that no two edge in the set belongs to a common path. Of course, one can define *node anti-chain* as well.

Remark: We are often interested in problems for which there is really no problem instances of size n for certain values of n (e.g., multiplying square Boolean matrices only has interesting input sizes of the form $2n^2$). In these cases, we artificially create the trivial problem instance of size n, such as the identically zero function of arity n. Also, the above definition of circuits do not allow constant values as inputs. The definition can trivially be changed to accommodate this.

We are mainly interested in circuits for two types of algebraic structures: (a) where A is a ring and (b) where $A = \{0, 1\}$ is the Boolean algebra in the above example. We call a circuit for A an *arithmetic circuit* or a *Boolean circuit* in cases (a) or (b), respectively.

8.7.1 The Parallel Prefix Problem.

We begin with a basic but important technique from Ladner and Fischer [19] for the so-called *parallel prefix problem*. In this problem, we assume that we are given an algebraic structure $(A; \circ)$ where the only operation $\circ$ is a binary associative operation (which we will call 'multiplication' or 'product'). As is usual with multiplicative notations, when convenient, we may write xy and $\prod_{i=1}^{n} x_i$ (respectively) instead of $x \circ y$ and $x_1 \circ x_2 \circ \cdots \circ x_n$. We call a circuit over such an A a *product circuit*. The parallel prefix problem instance (of size $n \geq 0$) amounts to computing the set of n functions

$$f_i(x_1, \ldots, x_n) := x_1 \circ x_2 \circ \cdots \circ x_i, \quad \text{for each } i = 1, \ldots, n.$$

We may call these f_i's the set 'iterative-$\circ$ functions' (on n variables). We shall apply this in two cases: where $\circ$ is addition and where $\circ$ is multiplication in a ring. Then, we call parallel prefix the *iterative addition* and *iterative multiplication* problems, respectively.

LEMMA 37 *There is a recursive construction of a family of product circuits $(C_n)_{n=0}^{\infty}$ of linear size and logarithmic depth that solves the parallel prefix problem.*

Proof. We may assume that n is a power of 2. C_1 is trivial, consisting of just an input node for x_1. So let $n > 1$. The following figures shows the recursive construction of C_n from $C_{n/2}$:

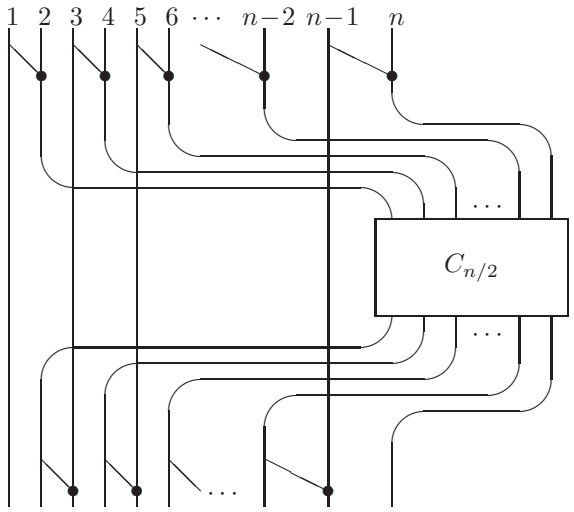


Figure 8.1 Construction of C_n ('•' indicates a gate)

In this figure, the edges of the circuit are implicitly directed downwards. It is easy to see that the size $s(n)$ of a circuit C_n satisfies the relation $s(1) = 1$ and $s(n) = s(n/2) + n - 1$. The solution is $s(n) = 2n - \log n - 2$. The depth of C_n is also easily seen to be $2 \log n$. **Q.E.D.**

8.7.2 Detecting Useless Markov States.

Recall in the previous section, in the context of a Markov chain whose nodes are machine configurations, we define a Markov state to be useless if it cannot reach a YES-configuration. To detect such useless states, we can formulate the following general problem: given the adjacency matrix of a digraph G, we want to determine its "useless nodes", defined to mean that those nodes that cannot reach a distinguished node in G.

LEMMA 38 *There is an NC^2 Boolean circuit family which computes, on any input n by n matrix A_n which represents the adjacency matrix of directed graph, the set of 0/1-functions $\{f_i : i = 1, \ldots, n\}$ where $f_i(A_n) = 1$ iff i cannot reach node n.*

Proof. This is rather straight forward: we can compute the products of Boolean matrices in NC^1. The transitive closure of A_n is given by $(A_n)^m$ for any $m \geq n$. By the usual doubling method, the transitive closure is obtained by $\log n$ matrix multiplications, hence by NC^2 circuits. Finally a node i can reach node n if and only if $A^*(i, n) = 1$. **Q.E.D.**

8.7.3 Computing the characteristic polynomial.

The determinant of an $n \times n$ matrix $A = (a_{i,j})$ can be expanded by its ith row (for any i) in the standard fashion:

$$\det(A) = a_{i,1}D_{i,1} + a_{i,2}D_{i,2} + \cdots + a_{i,n}D_{i,n}$$

where $(-1)^{i+j}D_{i,j}$ is the determinant[16] of the $(n-1)$-square matrix obtained by deleting the ith row and the jth column of A. $D_{i,j}$ is called the (i,j)-*cofactor* (or (i,j)-*complement*) of A. Let the *adjoint* $\mathtt{adj}(A)$ of A be the $n \times n$ matrix whose (i,j)th element $[\mathtt{adj}(A)]_{i,j}$ is the (j,i)-cofactor of A (notice the transposition of subscripts i and j). It is not hard to see that[17] that the following is valid for all A:

$$A \cdot \mathtt{adj}(A) = \mathtt{adj}(A) \cdot A = \det(A) \cdot I$$

(We only have to see that the off-diagonal elements of $A \cdot \mathtt{adj}(A)$ are of the form

$$a_{i,1}D_{j,1} + a_{i,2}D_{j,2} + \cdots + a_{i,n}D_{j,n}$$

where $i \neq j$. But this sum is also seen to be zero since it is the determinant of the singular matrix obtained by replacing the jth row of A with the ith row. On the other hand, the diagonal entries are equal to $\det(A)$, which may be zero if A is singular.) The *characteristic polynomial* $P_A(x)$ of A is the determinant

$$\begin{aligned} P_A(x) &:= \det(xI_n - A) \\ &= x^n + \lambda_1 x^{n-1} + \cdots + \lambda_{n-1}x + \lambda_n \end{aligned}$$

where I_n is the $n \times n$ identity matrix. (We will omit the subscript in I_n when convenient.) A fundamental identity is the Cayley-Hamilton theorem **Exercise:** that states that A is a root of the polynomial $P_A(x)$

$$P_A(A) = A^n + \lambda_1 A^{n-1} + \cdots + \lambda_{n-1}A + \lambda_n I = 0.$$

Our goal is to develop a space-efficient algorithm for computing the characteristic polynomial. To compute the characteristic polynomial of A means to determine the above coefficients $\lambda_1, \ldots, \lambda_n$. Note that this computation is a generalization of the problem of computing determinant since the constant term in P_A is (up to sign) equal to $\det(A)$. We present an efficient parallel implementation of Samuelson's method[18] by Berkowitz [4]. For this, we use the notation $red(A)$ ('reduction' of A) which is the $(n-1)$-square matrix obtained from A by deleting the first row and first column. Define the column $(n-1)$-vector $\mathtt{col}(A)$ and row $(n-1)$-vector $\mathtt{row}(A)$ so that

$$A = \begin{pmatrix} a_{1,1} & \mathtt{row}(A) \\ \mathtt{col}(A) & \mathtt{red}(A) \end{pmatrix}$$

LEMMA 39 *The characteristic polynomial of A is related to the matrix $\mathtt{red}(A)$ as follows:*

$$P_A(x) = (x - a_{1,1}) \det(xI_{n-1} - \mathtt{red}(A)) - \mathtt{row}(A) \cdot \mathtt{adj}(xI_{n-1} - \mathtt{red}(A)) \cdot \mathtt{col}(A). \tag{22}$$

Proof.

$$\begin{aligned} P_A(x) &= \det(xI_n - A) \\ &= (x - a_{1,1}) \det(xI_{n-1} - A) + \sum_{j=2}^{n} a_{1,j}D_{1,j} \end{aligned}$$

[16] This determinant is called the (i,j)-*minor* of A

[17] See [11] for most basic results on matrices.

[18] The importance of this method (as opposed to an earlier method of Csanky that has comparable complexity) is that it uses no divisions, so it is applicable to any unitary commutative ring.

where $D_{1,j}$ is the $(1,j)$-cofactor of $xI_n - A$. Write

$$D[i_1, i_2, \ldots; j_1, j_2, \ldots]$$

for the determinant of the square matrix obtained by deleting rows $i_1, i_2, \ldots$ and columns $j_1, j_2, \ldots$. The lemma follows from the following identity

$$\begin{aligned}
\sum_{j=2}^{n} a_{1,j} D_{1,j} &= \sum_{j=2}^{n} a_{1,j}(-1)^{1+j} D[1; j] \\
&= \sum_{j=2}^{n} a_{1,j}(-1)^{1+j} \sum_{i=2}^{n} a_{i,1}(-1)^{i} D[1, i; 1, j] \\
&= -\sum_{j=2}^{n}\sum_{i=2}^{n} a_{1,j} a_{i,1}(-1)^{i+j} D[1, i; 1, j] \\
&= -\sum_{j=2}^{n}\sum_{i=2}^{n} a_{1,j} a_{i,1} D'_{i,j}
\end{aligned}$$

where $D'_{i,j}$ is the $(i-1, j-1)$-cofactor of $\mathtt{red}(xI_n - A) = xI_{n-1} - \mathtt{red}(A)$. Since $D'_{i,j}$ is the $(j-1, i-1)$th entry of $\mathtt{adj}(xI_{n-1} - \mathtt{red}(A))$, the lemma follows. **Q.E.D.**

The adjoint of $xI - A$ can be expressed with the help of the the next lemma.

Lemma 40

$$\mathtt{adj}(xI - A) = \sum_{i=0}^{n-1} B_i x^{n-1-i} \tag{23}$$

where

$$B_i = A^i + \lambda_1 A^{i-1} + \cdots + \lambda_{i-1} A + \lambda_i I$$

and λ_i *are the coefficients of the characteristic polynomial*

$$P_A(x) = x^n + \lambda_1 x^{n-1} + \cdots + \lambda_{n-1} x + \lambda_n.$$

Proof. We observe that $B_0 = I_n$ and for $i = 1, \ldots, n$,

$$B_i = AB_{i-1} + \lambda_i I.$$

Hence

$$\begin{aligned}
&\det(xI - A) \cdot I \\
&= [x^n + \lambda_1 x^{n-1} + \cdots + \lambda_{n-1} x + \lambda_n] \cdot I \\
&= [x^{n-1} B_0](xI - A) + x^{n-1} AB_0 + [\lambda_1 x^{n-1} + \lambda_2 x^{n-2} + \cdots + \lambda_{n-1} x + \lambda_n] \cdot I \\
&= [x^{n-1} B_0 + x^{n-2} B_1](xI - A) + x^{n-2} AB_1 + [\lambda_2 x^{n-2} + \lambda_3 x^{n-3} + \cdots + \lambda_n] \cdot I \\
&= \cdots \\
&= \left[\sum_{i=0}^{n-1} x^{n-1-i} B_i\right](xI - A) + x^0 AB_{n-1} + \lambda_n I \\
&= \left[\sum_{i=0}^{n-1} x^{n-1-i} B_i\right](xI - A)
\end{aligned}$$

where the last equality follows from

$$x^0 AB_{n-1} + \lambda_n I = B_n = P_A(A) = 0$$

by the Cayley-Hamilton theorem. On the other hand $\det(xI - A) \cdot I$ can also be expressed as

$$\det(xI - A) \cdot I = \mathtt{adj}(xI - A) \cdot (xI - A).$$

Since $xI - A$ is nonsingular (x is a indeterminate), we can cancel $xI - A$ as a factor from the two expressions for $\det(xI - A) \cdot I$, giving the desired equality for the lemma. **Q.E.D.**

The last two lemmas show that the characteristic polynomial $P_A(x)$ of A can be computed from the characteristic polynomial $P_{\mathtt{red}(A)}(x)$ of $\mathtt{red}(A)$ and the matrix products $\mathtt{red}(A)^i$ (for $i = 1, \ldots, n-1$). Let us derive this precisely. Let the coefficients of $P_{\mathtt{red}(A)}(x)$ be given by μ_i $(i = 0, \ldots, n-1)$ where $P_{\mathtt{red}(A)}(x) = \sum_{i=0}^{n-1} \mu_{n-1-i} x^i$. Then we see that

$$\begin{aligned}
P_A(x) &= (x - a_{1,1}) P_{\mathtt{red}(A)}(x) - \mathtt{row}(A) \cdot \left(\sum_{i=0}^{n-1} B_i x^{n-1-i} \right) \cdot \mathtt{col}(A) \\
&= (x - a_{1,1}) \sum_{i=0}^{n-1} \mu_i x^{n-1-i} + \sum_{i=0}^{n-1} x^{n-1-i} \mathtt{row}(A) \cdot \left(\sum_{j=0}^{i} (\mathtt{red}(A))^j \mu_{i-j} \right) \cdot \mathtt{col}(A) \\
&= \sum_{i=0}^{n-1} \mu_i x^{n-i} + \sum_{i=0}^{n-1} x^{n-1-i} \left(-a_{1,1}\mu_i + \sum_{j=0}^{i} \beta_j \mu_{i-j} \right) \\
&\qquad (\text{where } \beta_j = \mathtt{row} \cdot (A) (\mathtt{red}(A))^j \cdot \mathtt{col}(A)) \\
&= \sum_{j=0}^{n} \lambda_{n-j} x^j
\end{aligned}$$

where

$$\lambda_j = \begin{cases} \mu_0 & \text{if } j = 0 \\ \mu_j - a_{1,1}\mu_{j-1} + \sum_{k=0}^{j-1} \beta_k \mu_{j-1-k} & \text{if } j = 1, \ldots, n-1 \\ -a_{1,1}\mu_{n-1} + \sum_{k=0}^{n-1} \beta_k \mu_{n-1-k} & \text{if } j = n \end{cases}$$

We can write this in matrix form. Let C_0 be the following $(n+1) \times n$ matrix:

$$C_0 := \begin{pmatrix}
1 & & & & & & \\
\beta_0 - a_{1,1} & 1 & & & & & \\
\beta_1 & \beta_0 - a_{1,1} & 1 & & & & \\
\beta_2 & \beta_1 & \beta_0 - a_{1,1} & & & & \\
\vdots & & & \ddots & & & \\
\beta_{n-2} & \cdots & & & \beta_0 - a_{1,1} & 1 & \\
\beta_{n-1} & \beta_{n-2} & \cdots & & & \beta_0 - a_{1,1} & 1 \\
0 & \beta_{n-1} & \beta_{n-2} & \cdots & & \beta_1 & \beta_0 - a_{1,1}
\end{pmatrix}$$

Then we have

$$(\lambda_0, \ldots, \lambda_{n-1}, \lambda_n)^T = C_0 \cdot (\mu_0, \ldots, \mu_{n-1})^T$$

where $(\cdots)^T$ indicates matrix transpose. We repeat this procedure to express the μ_i's in terms of the characteristic polynomial of $\mathtt{red}(\mathtt{red}(A)) = \mathtt{red}^2(A)$, etc. In general, let C_i be the $(n+1-i) \times (n-i)$ matrix that reduces the characteristic polynomial of $\mathtt{red}^i(A)$ to that of $\mathtt{red}^{i+1}(A)$. The entries[19] of C_i consists of 0, 1, the diagonal elements $a_{i+1,i+1}$ of the matrix A, and elements that can be constructed from

$$\mathtt{row}(\mathtt{red}^i(A)) \cdot \mathtt{red}^{i+1}(A) \cdot \mathtt{col}(\mathtt{red}^i(A)).$$

Putting these together, we get

$$(\lambda_0, \ldots, \lambda_n)^T = C_0 C_1 \cdots C_{n-1}$$

LEMMA 41 *Given an $n \times n$ matrix A we can construct in deterministic log-space an arithmetic circuit C of depth $O(\log^2 n)$ and size $n^{O(1)}$ such that C computes (the coefficients of) $P_A(x)$.*

Proof. We proceed as follows:

[19] In fact C_i has the property that each (j,k)th entry is equal to the $(j+1, k+1)$th entry provided, of course, that the $(j+1, k+1)$th entry is defined. This property is the defining characteristic of *Toeplitz matrices* and it is known that the multiplication of Toeplitz matrices can be done more efficiently than we care to exploit here **Exercise:**.

1. We first construct a circuit to compute the set of polynomials

$$\{(\mathtt{red}^i(A))^j \ : \ i = 1, \ldots, n-1 \text{ and } j = 1, \ldots, i\}.$$

Note that to compute a matrix means to compute each of its entries and to compute a polynomial means to compute each of its coefficients. It suffices to show that for each i, we can compute $\{(\mathtt{red}^i(A))^j \ : \ j = 1, \ldots, i\}$ with a circuit of polynomial size and depth $O(\log^2 n)$. But this amounts to the parallel prefix computation on i copies of the matrix $\mathtt{red}^i(A)$. Parallel prefix, we saw, uses an $O(\log n)$ depth product circuit. Each gate of the product circuit is replaced by an arithmetic circuit of depth $O(\log n)$ since we can multiply two $n \times n$ matrices in this depth (straightforward). Hence the overall depth is $O(\log^2 n)$.

2. Next, we compute the elements

$$\{\mathtt{row}(\mathtt{red}^{i-1}(A) \cdot (\mathtt{red}^i(A))^j \cdot \mathtt{col}(\mathtt{red}^{i-1}(A) \ : \ i = 1, \ldots, n-1 \text{ and } j = 1, \ldots, i\}.$$

This takes $O(\log n)$ depth. We have now essentially computed the entries of the matrices $C_0, \ldots, C_{n-1}$.

3. We can compute the product $\prod_{i=0}^{n-1} C_i$ using a balanced binary tree of depth $O(\log n)$ to organize the computation. Each level of this binary tree corresponds to the multiplication (in parallel) of pairs of matrices. Since each matrix can be multiplied in $O(\log n)$ depth, the overall circuit depth is $O(\log^2 n)$.

One can easily verify that the circuit is polynomial in size. **Q.E.D.**

8.7.4 Computing the Matrix Inverse

We want to compute the inverse of a matrix A. As before, let

$$P_A(x) = x^n + \lambda_1 x^{n-1} + \cdots + \lambda_{n-1} x + \lambda_n$$

be the characteristic polynomial. Since $P_A(x) = \det(xI - A)$ we see that

$$\lambda_n = P_A(0) = \det(-A) = (-1)^n \det(A).$$

Next, by lemma 40, we have that

$$\begin{aligned} \mathtt{adj}(-A) &= B_{n-1} \\ &= A^{n-1} + \lambda_1 A^{n-2} + \cdots + \lambda_{n-2} A + \lambda_{n-1}. \end{aligned}$$

Note that $\mathtt{adj}(-A) = (-1)^{n-1}\mathtt{adj}(A)$. Putting these together, we get

$$\begin{aligned} A^{-1} &= \frac{1}{\det(A)}\mathtt{adj}(A) \\ &= -\frac{1}{\lambda_n}(A^{n-1} + \lambda_1 A^{n-2} + \cdots + \lambda_{n-2} A + \lambda_{n-1}). \end{aligned}$$

It should now be easy to deduce:

LEMMA 42 *We can detect if an n-square matrix A is nonsingular using an NC^2 arithmetic circuit. Moreover, in case A is nonsingular, we can compute its inverse A^{-1} with another NC^2 arithmetic circuit.*

8.7.5 Balanced p-ary Notations

In applying the preceding results, we will use matrices A whose entries are rational numbers. Assuming the usual binary representation of integers, if the integers involved in the computation are k-bits long, then an $O(\log^2 n)$ depth arithmetic circuits translate into Boolean circuits of depth $\Omega(\log^2 n \log k)$, at least (and in our applications $k = \Omega(n)$). To obtain a depth of $O(\log^2 n)$ on Boolean circuits for computing characteristic polynomials, we need one new idea: by a suitable choice of representing integers, we can implement the operations of addition with constant depth (i.e., NC^0) circuits. This in turn yields an improved depth for several other problems. Circuits in this subsection shall mean Boolean circuits unless otherwise stated.

Definition 6
(i) A *representation* r of a algebraic structure A is an onto function

$$r : \{0,1\}^* \to A.$$

We say $u \in \{0,1\}^*$ is an *r-representative* of $r(u)$.
(ii) We say an operation $f : A^n \to A$ is in NC^k with respect to r if there is an NC^k *Boolean* circuit family $\overline{C} = (C_m)_{m \geq 0}$ such that for each m, C_m computes f in the sense that for all $u_1, \ldots, u_n \in \{0,1\}^m$,

$$r(C_m(u_1, \ldots, u_n)) = f(r(u_1), \ldots, r(u_n))$$

■

We usually like to ensure that each element can be represented by arbitrarily large binary strings. For instance, as in the binary representation of numbers, we may have the property $r(0u) = r(u)$ for all $u \in \{0,1\}^*$. In other words, we can left pad an representation by 0's. In practice, ring elements have some natural notion of "size" and we may insist that the representation r 'respect' this size function within some bounds (e.g., $|r(x)| \geq size(x)$). We shall not concern ourselves with such considerations but the interested reader may consult [6].

Our goal is to find a representation of integers so that addition and negation is in NC^0, multiplication and iterated addition is in NC^1 and iterated multiplication is in NC^2. We resort to the *balanced p-ary representation* of integers of Avizienis [1]. Here $p \geq 2$ is an integer and a *balanced p-ary number* is a finite string

$$u_1 u_2 \cdots u_n$$

where u_i is an integer with $|u_i| \leq p-1$. This string represents the integer $(u_1, \ldots, u_n)_p$ given by

$$(u_1, \ldots, u_n)_p := \sum_{i=0}^{n-1} u_{i+1} p^i.$$

So u_1 is the least significant digit. Clearly, the usual p-ary representation of a number is a balanced p-ary representation of the same number. This representation is redundant in a rather strong sense. We will implicitly assume that strings such as $u_1, \ldots, u_n$ are ultimately encoded as binary strings, so that they fit our formal definition of representations.

LEMMA 43 *With respect to the balanced p-ary representation of integers, for any $p \geq 3$:*
(i) Addition and negation of integers are in NC^0.
(ii) Iterated addition and multiplication of integers are in NC^1.
(iii) Iterated multiplication is in NC^2.

Proof. (i) Suppose we want to add $(u_1, \ldots, u_n)_p$ to $(v_1, \ldots, v_n)_p$. Note that for each $i = 1, \ldots, n$, we can express the sum $u_i + v_i$ as

$$u_i + v_i = p x_i + y_i$$

with $|x_i| \leq 1$ and $|y_i| \leq p-2$. To see this, note that $|u_i + v_i| \leq 2p-2$ and if $|u_i + v_i| \leq p-2$ or $|u_i + v_i| \geq p$ then it is clear that the desired x_i, y_i can be found. The remaining possibility is $|u_i + v_i| = p-1$. In that case we could let $x_i = 1$ and $y_i = \pm 1$, but note that this is possible only because $p \geq 3$ (we would violate the constraint $|y_i| \leq p-2$ if $p = 2$). Now the sum of the two numbers is given by $(w_1, \ldots, w_n w_{n+1})_p$ where

$$w_i = x_{i-1} + y_i$$

for $i = 1, \ldots, n+1$ (taking $x_0 = y_{n+1} = 0$). Clearly this can be implemented by an NC^0 circuit.
(ii) To show iterated addition is in NC^1 we simply use part (i) and the technique for parallel prefix.

Now consider multiplication of integers. Suppose we want to form the product of the numbers $(u_1, \ldots, u_n)_p$ and $(v_1, \ldots, v_n)_p$. For each $i, j = 1, \ldots, n$, we form the product $u_i v_j$ and we can express this in the form

$$u_i v_j = p x_{i,j} + y_{i,j}$$

where $|x_{i,j}| \leq p-1$ and $|y_{i,j}| \leq p-1$ (since $|u_i v_j| \leq (p-1)^2 \leq p(p-1) + (p-1)$). For each $i = 1, \ldots, n$, form the number

$$X_i = (00 \cdots 00 x_{i,1} x_{i,2} \cdots x_{i,n-1} x_{i,n})_p$$

where X_i has a prefix of i zeroes. Similarly, form the number

$$Y_i = (00\cdots 00 y_{i,1} y_{i,2} \cdots y_{i,n-1} y_{i,n})_p$$

where Y_i has a prefix of $i-1$ zeroes. It is then easy to see that the product is given by the sum

$$\sum_{i=1}^{n} (X_i + Y_i).$$

But each summand has at most $2n$ digits and there are $2n$ summands. We can form a balanced binary tree T on $2n$ leaves to organize this summation process: each leaf is labeled with one of these summands and each interior node is labeled with the sum of the labels at the leaves below. Clearly the root of T has the desired product. This tree converts into a Boolean circuit of depth $O(\log n)$. This shows multiplication is in NC^1.
(iii) We leave this as exercise. **Q.E.D.**

Next we extend the lemma to matrix rings. By the *balanced p-ary representation of matrices with integer entries* we mean that each entry is encoded by balanced p-ary notation, and matrices are stored (to be specific) in row-major order.

LEMMA 44 *With respect to the balanced p-ary representation ($p \geq 3$) of integer matrices:*
(i) Addition and negation are in NC^0.
(ii) Multiplication is in NC^1.
(iii) Interated multiplication is in NC^2.

Proof. It is clear addition and negation of integer matrices can be implemented efficiently by the previous lemma. For multiplication, suppose we want to compute the product of two $n \times n$ matrices A and B, and each entry has at most n bits. We note that each entry of AB is the sum of at most n products of pairs of entries. These individual products can be viewed at the sum of at most $2n$ numbers of n bits, as revealed in the proof of the previous lemma. So for each entry, we need to sum $O(n^2)$ numbers, each of n bits. Again, we can arrange these as a balanced binary tree of depth $O(\log n)$. This gives us the efficient NC^1 circuit we seek.

To get an NC^2 circuit family for iterated multiplication of integer matrices we simply apply parallel prefix to the previous part. **Q.E.D.**

Finally we consider matrices whose entries are rational numbers. A rational number is represented by a pair of balanced p-ary representation, extended to matrices as before. Unfortunately, we no longer know how to do addition of rational numbers in NC^0. Nevertheless, we have the following:

LEMMA 45 *With respect to the balanced p-ary ($p \geq 3$) representation of matrices with rational number entries:*
(i) Iterated multiplication is in NC^2.
(ii) Characteristic polynomial computation is in NC^2.

Proof. (i) Suppose we want to compute the iterated product

$$A_1, A_1A_2, \ldots, A_1A_2\cdots A_n$$

where each A_i is a $n \times n$ matrix with rational number entries, and each entry is represented by pairs of n-bit integers. We first convert each A_i to integer matrices B_i and compute an integer D_i such that $A_i = \frac{1}{D_i} B_i$. To do this, first form the product $D_{i,j}$ of the denominators in the jth row of A_i; then multiply each entry in the jth row of A_i by $D_{i,j}$. Doing this for all rows, we get B_i; of course D_i is just the product of all the $D_{i,j}$'s. It is clear that we can obtain each of the $D_{i,j}$ and D_i by iterated multiplication in NC^2. Notice that $D_{i,j}$ and D_i are $O(n^3)$-bit integers and so we can compute B_i from A_i and $D_{i,j}$'s in NC^1.

Next, we compute the iterated integer products $\{D_1, D_1D_2, \ldots, D_1D_2\cdots D_n\}$ in NC^2. Similarly, we compute the iterated matrix product $\{B_1, B_1B_2, \ldots, B_1B_2\cdots B_n\}$ in NC^2. It is clear that

$$A_1A_2\cdots A_i = \frac{1}{D_1D_2\cdots D_i} B_1B_2\cdots B_i$$

for each $i = 1, \ldots, n$. This can be computed in NC^1 since each of the integer involved in polynomial in size.
(ii) We imitate the proof of lemma 41. Details are left as exercise. **Q.E.D.**

One more computational problem: we need to be able to check the sign of a balanced p-ary number. (In our application, we want to compare such a number with $\frac{1}{2}$, which is easily reduced to checking if a number is positive.) But after the preceding development, the reader should have no trouble devising an NC^1 solution **Exercise:**.

Putting it all together. We must tidy up the loose bits in the proof of the main theorem in the last section. In particular, we must address the issue of how to implicitly construct and represent the reduced transition matrices C described in the last section. Let A be the transition matrix from which we derive C. Since we want to do all this using $O(s^2)$ space, we cannot afford to write A or C explicitly. Then we must see how the techniques given in this section can be adapted to some implicit representation. All this is tedious but it is crucial that the reader understands how this can be done. So let us assume a given probabilistic machine M accepting in space $s(n)$, and w is an input. Let us begin by constructing matrix C: the proof of lemma 38 shows a transitive closure circuit of depth $O(\log^2 n)$ applied to the underlying graph G_A of the transition matrix A to determine the inessential states. But note that the transitive closure circuit is relatively systematic that we can assume some numbering of its gates such that given any gate number g, we can determine the gates at the other end of incoming as well as outgoing edges at g, and given g, we also know the Boolean function labeling g. The gate numbers can be stored in $O(s)$ space and we can determine these information also in $O(s)$ space. It is now not hard to see that we can determine the output of any gate in $O(s)$ space, given that we know the input graph G_A. This is not hard to do (we basically store one Boolean value at each gate along the path from the output gate to the current position – a similar proof using this technique is shown in chapter 10.) In this way, in $O(s)$ space, we can determine if any given i is inessential.

The basis of the preceding argument is the observation that the transitive closure circuit is quite systematic and hence in space $O(s)$ we can answer basic questions such as connections between gates, etc. Similarly for all the other circuits in this section. The student should carefully work out some other cases. With this, we conclude.

Remark: In Chapter 10, we will study the property stated above, that all the circuits in this section are 'systematically constructed' so that we can essentially determine gates and their interconnections in circuits efficiently. (These are called *uniformity* conditions.) For this reason we are contented with a somewhat sketchy outline here.

Exercise

Exercise 8.7.1:

(i) (Generalized Bezout's theorem) Let

$$F(x) = F_0 x^m + F_1 x^{m-1} + \cdots + F_m \quad (F_0 \neq 0)$$

be a matrix polynomial where each F_i is an $n \times n$ matrix. The *right value* of $F(x)$ at an $n \times n$ matrix A is given by

$$F(A) = F_0 A^m + F_1 A^{m-1} + \cdots + F_m.$$

(The *left value* $\hat{F}(A)$ is similarly obtained except that A is multiplied from the left.) Show that if $F(x)$ is divided by $xI - A$ from the left, the remainder is the right value $F(A)$. Hint: the proof is a direct long division of $F(x)$.

(ii) Let $B(x)$ be the adjoint of $xI - A$. Conclude that $(xI - A)B(x) = P_A(x)I$.

(iii) Infer the Cayley-Hamilton theorem from the Generalized Bezout's theorem. □

Exercise 8.7.2: (Fisher-Ladner) Improve the $2 \log n$ depth for the parallel prefix circuit C_n in the text. **Hint:** consider a recursive construction of a circuit C'_n as illustrated in the following figure where, with the proper wires added, we have

$$DEPTH(C'_n) = 1 + \max\{DEPTH(C_{n/2}), DEPTH(C'_{n/2})\}.$$

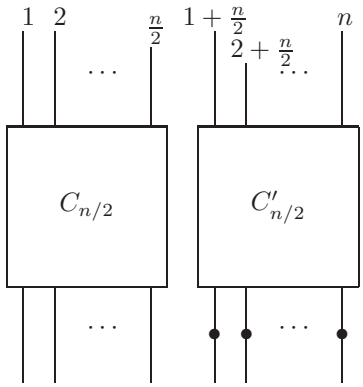


Figure 8.2 Construction of C'_n (the connections between $C'_{n/2}$ and $C_{n/2}$ not shown)

Note that the original C_n is used in this construction.
(a) Give exact expressions for the size and depth of C'_n. (Hint: the exact size of C'_n involves the Fibonacci numbers.)
(b) Compare the fan-out degree of C_n and C'_n. □

Exercise 8.7.3: (Balanced p-ary representation)
(a) Suppose we have a fixed finite state automaton M that reads an input sequence of symbols $a_1 \cdots a_n$ in real time. Let q_i be the state of M after reading a_i, starting from the start state q_0. Show an NC^1 Boolean circuit for the problem of computing $\{q_1, \ldots, q_n\}$ for any input sequence $a_1 \cdots a_n$. **Hint:** Apply parallel prefix.
(b) Apply the above to determine the sign of a balanced p-ary number in NC^1 ($p \geq 3$).
(c) Show how to convert a balanced p-ary number to a p-ary number in NC^1 (assume part (b)). □

Exercise 8.7.4: (General representation)
Let us fix $b \geq 1, p \geq 0, q \geq 1$. We say a word $u \in \{-p, -p+1, \ldots, q-1, q\}\star$ represents the integer

$$\sum_{i=0}^{n} u_i b^i \tag{24}$$

in the (b, p, q)-notation. Note that b-ary notations are $(b, 0, b-1)$-notations; b-adic notations are simply $(b, 1, b)$-notations; balanced p-ary numbers are $(p, -p, p)$-notations. Show that for $(3, 0, 3)$-notations, addition and multiplication of natural numbers (since we cannot represent negative numbers) is in NC^0 and NC^1, respectively. For what values of (b, p, q)-notations are these results true? □

End Exercise

8.8 Complement of Probabilistic Space

This section proves that probabilistic space is closed under complementation. We begin with a useful result:

Lemma 46 *Let B be the fundamental part of the transition matrix of a Markov chain. If B is $m \times m$ and the entries of B are taken from $\{0, \frac{1}{2}, 1\}$, then the stochastic closure B^* has the property that dB^* is an integer matrix for some integer $d < m!2^m$.*

Proof. We know that $B^* = (I - B)^{-1}$. Thus $B^* = \frac{1}{\det(I-B)}\mathtt{adj}(I - B)$. Clearly each entry of $2^{m-1}\mathtt{adj}(I - B)$ is an integer. Also, $\frac{c}{\det(I-B)}$ is an integer for some $c \leq m!$. The result follows. **Q.E.D.**

Theorem 47 *Let* M *be any probabilistic machine that accepts in space $s(n) \geq \log n$. Then there is a $c > 0$ such that every accepted input of length n is accepted with probability more than*

$$\frac{1}{2} + 2^{-c^{s(n)}}.$$

Proof. At the end of section 3, we showed that the probability of accepting any input is given by a suitable entry of D^*, where D is the fundamental part of a reduced transition matrix C. D is $m \times m$ with $m = nO(1)^{s(n)} = O(1)^{s(n)}$. We then apply the previous lemma. **Q.E.D.**

LEMMA 48 *Let $s(n) \geq \log n$. For any probabilistic machine that runs in space $s(n)$, there is a probabilistic machine $\mathbb{N}$ accepting $L(\mathrm{M})$ and runs in space s with error gap $(0, \frac{1}{2}]$. Morever, $\mathbb{N}$ halts with probability 1 and has average time $2^{2^{O(s)}}$.*

Proof. Let $c = \max\{c_1, c_2\}$ where c_1 is chosen so that there are at most c_1^s configurations using space at most s, and c_2 is chosen so that (by the previous theorem) if M accepts an input w then M accepts with probability greater than $\frac{1}{2} + 2^{-c_2^{s(|w|)}}$. Now $\mathbb{N}$ is obtained by modifying the proof of lemma 16 in the last section:

repeat forever
1. Simulate M for another c^s steps. Nondeterministic choices of M become coin-tossing choices of $\mathbb{N}$.
2. If M answers YES then we answer YES.
   If M answers NO, we rewind our tapes to prepare for a restarted simulation.
3. Toss $2c^s$ coins and answer NO if all tosses turn up *heads*.

end

(Notice that unlike in lemma 16, each iteration of the loop continues the simulation from where the last iteration left off, provided the last iteration did not halt.) Now we see that $\mathbb{N}$ halts with probability 1. The average time $\bar{t}$ is seen to satisfy the bound

$$\bar{t} \leq 3c^s + (1 - 2^{-2c^s})\bar{t}$$

which gives us $\bar{t} = 2^{2^{O(s)}}$ as desired.

If M rejects then clearly $\mathbb{N}$ rejects. So assume that M acccepts. Let us call a configuration C of M *live* if there is a computation path starting from C into a YES configuration. Similarly, a configuration C of N is *live* if there is a computation path from C into one in which the simulated M answers YES. If a configuration is not alive, we say it is *dead*.

For $k \geq 1$, define the following events for the probabilistic spaces $\Omega_w^{\mathbb{N}}$ and Ω_w^{M}:

$$\begin{array}{lcl} R_k^{\mathbb{N}} & = & \{\text{N answers NO in the } k\text{th iteration}\} \\ D_k^{\mathbb{N}} & = & \{\text{N became dead during the } k\text{th iteration}\} \\ D_k^{\mathrm{M}} & = & \{\text{M became dead between the } (k-1)c^s\text{th and the } kc^s\text{th step}\} \\ A_k^{\mathbb{N}} & = & \{\text{N is alive at the end of the } k\text{th iteration}\} \\ A_k^{\mathrm{M}} & = & \{\text{M is alive at the end of the } kc^s \text{ step}\} \end{array}$$

Then the rejection event of N corresponds to

$$\begin{array}{rcl} \bigcup_{k\geq 1}\{R_k^{\mathbb{N}}\} & = & \bigcup_{k\geq 1}\left(\{R_k^{\mathbb{N}}, A_k^{\mathbb{N}}\} \cup \bigcup_{j=1}^{k}\{R_k^{\mathbb{N}}, D_j^{\mathbb{N}}\}\right) \\ & = & (\bigcup_{k\geq 1}\{R_k^{\mathbb{N}}, A_k^{\mathbb{N}}\}) \cup (\bigcup_{j\geq 1}\bigcup_{k\geq j}\{R_k^{\mathbb{N}}, D_j^{\mathbb{N}}\}) \end{array}$$

Clearly the probability that M remains alive after c^s steps is at most $1 - e$ where

$$e := 2^{-c^s}.$$

Since the probability of getting $2c^s$ heads in a row is e^2, the probability that N remains alive through one iteration is at most $(1 - e)(1 - e^2)$. We claim that

$$\Pr\{R_k^{\mathbb{N}}, A_k^{\mathbb{N}}\} = (1 - e)^k(1 - e^2)^{k-1}e^2.$$

(In this proof, it is instructive to set up the connection between the probabilistic spaces Ω_w^N and Ω_w^M for input w.) Hence

$$\begin{aligned}\Pr(\bigcup_{k\geq 1}\{R_k^{\mathbb{N}}, A_k^{\mathbb{N}}\}) &= \sum_{k\geq 1}\Pr\{R_k^{\mathbb{N}}, A_k^{\mathbb{N}}\} \\ &\leq e^2\sum_{k\geq 1}(1-e)^k(1-e^2)^{k-1} \\ &< e^2\sum_{k\geq 1}(1-e)^{k-1} \\ &= e.\end{aligned}$$

Next,

$$\begin{aligned}\Pr(\bigcup_{j\geq 1}\bigcup_{k\geq j}\{R_k^{\mathbb{N}}, D_j^{\mathbb{N}}\}) &\leq \Pr(\bigcup_{j\geq 1}\{D_j^{\mathbb{N}}\}) \\ &= \sum_{j\geq 1}\Pr\{D_j^{\mathbb{N}}\} \\ &\leq \sum_{j\geq 1}\Pr\{D_j^{\mathrm{M}}\} \\ &= \Pr\{\text{M rejects}\} \\ &\leq \frac{1}{2} - e\end{aligned}$$

by our choice of the constant $c \geq c_2$. Above we have used the inequality $\Pr\{D_j^{\mathbb{N}}\} \leq \Pr\{D_j^{\mathrm{M}}\}$ relating across two different probability spaces. Hence the probability that N rejects is less than the sum of $e + (\frac{1}{2} - e)$. We conclude that N accepts with probability greater than $\frac{1}{2}$. **Q.E.D.** reject??

The technique in this lemma can be viewed as using $m = c^s$ coin tosses to control a loop so that the expected number of iterations is 2^m. Since we need only $\log m$ space to control this loop, we are able to probabilistically achieve a number of iterations that is double exponential in the space used. This technique, due to Gill, demonstrates one of the fundamental capabilities of coin-tossing that distinguishes space-bounded probabilism from, say, space-bounded alternating computations. The expected number of iterations is achieved in the worst case: if M is a machine that does not halt, then $\mathbb{N}$ has expected time $2^{2^{\Omega(s(n))}}$.

We are ready to show that probabilistic space is closed under complementation.

THEOREM 49 *If $s(n) \geq \log n$ is space-constructible then*

$$PrSPACE(s) = \text{co-}PrSPACE(s).$$

This result was shown by Simon [32]; the proof here is essentially from [28]. This result is almost an immediate consequence of lemma 48.

Proof. Given any probabilistic machine accepting in space $s(n)$, lemma 48 gives us another probabilistic machine accepting the same language in the same space bound with error gap $(0, \frac{1}{2}]$; moreover, M halts with probability 1. Then let N be the complement of M: i.e., N answers YES iff M answers NO. For any input w, the probability that N accepts w plus the probability that M accepts w is equal to the probability of halting, i.e., 1. Hence, N has the error gap $[\frac{1}{2}, 1)$. It follows that N accepts if and only if M rejects. **Q.E.D.**

8.9 Stochastic Time and Space

In this section, we give upper bounds on the complexity of time and space-bounded stochastic computations. Stochastic space is especially interesting in view of the tremendous computational power that seems inherent in it. Also, instead of Markov chains we now turn to the study of discrete time dynamical systems.

THEOREM 50 *For all t, StA-TIME$(t) \subseteq$ ATIME(t^3).*

The basic idea for this result is the bit-counting technique that was used quite effectively for simulating probabilistic alternating machines (chapter 7, section 6). It turns out that several new technical problems arises.

Let M be a stochastic alternating machine that accepts in time $t(n)$. We construct an alternating machine N to simulate M. For the rest of this proof, we fix an input w that is accepted by M and let $t = t(|w|)$. We may assume that N has guessed t correctly and let T_0 be the accepting computation tree of M on w obtained by truncating the complete computation tree $T_{\rm M}(w)$ at levels below t (as usual, root is level 0). To demonstrate the essential ideas, we assume that M has $\oplus$- and $\otimes$-states only. As in chapter 7 we 'normalize' the least fixed point values $Val_{T_0}(C)$ for each configuration in T_0:

$$VAL_0(C) := 2^{2^{t-\ell}}\, Val_{T_0}(C)$$

where $\ell = level(C)$. Thus T_0 is accepting if and only if $VAL_0(C_0) > 2^{2^t-1}$ where C_0 is the root of T_0. It is easy to see that $VAL_0(C)$ is an integer between 0 and 2^{2^t}. We shall think of $VAL_0(C)$ as a 2^t digit number in the balanced 4-ary notation. Although the balanced 4-ary notation is highly redundant, we will want to refer to the 'ith digit of $VAL_0(C)$' in an unambiguous manner. We will show how to uniquely choose a balanced 4-ary representation for each $VAL_0(C)$.

Let us note that in fact $VAL_0(C)$ can be explicitly defined as follows: if $\ell = level(C)$ and $C \vdash (C_L, C_R)$ (provided C is not a leaf) then

$$VAL_0(C) \;=\; \begin{cases} 0 & \text{if } C \text{ is a non-YES leaf} \\ 2^{2^{t-\ell}} & \text{if } C \text{ is a YES leaf} \\ 2^{2^{t-\ell-1}-1}(\,VAL_0(C_L) + VAL_0(C_R)) & \text{if } C \text{ is an } \oplus\text{-configuration} \\ VAL_0(C_L)\cdot VAL_0(C_R) & \text{if } C \text{ is an } \otimes\text{-configuration} \end{cases}$$

It follows that each $VAL_0(C)$ has at most $2^{t-\ell}$ digits of significance.

The alternating simulation of N effectively constructs a tree T_1 that is an 'expansion' of T_0. To describe T_1, we need to define a certain product of trees:

Definition 7 Let T, T' be any two trees. For nodes $i, j \in T$, write $i \to j$ to mean that i is the parent of j. Their *product* $T \times T'$ consists of nodes (i, i') where $i \in T$ and $i' \in T'$ such that $(i, i') \to (j, j')$ if and only if either
(a) $i = j$ and $i' \to j'$, or
(b) $i \to j$, i' is the root of T' and j' is a leaf of T'. ■

Clearly,

$$T \times T' = T' \times T \iff T = T',$$

$$SIZE(T \times T') = SIZE(T' \times T) = SIZE(T) \cdot SIZE(T'),$$

and

$$DEPTH(T \times T') = DEPTH(T' \times T) = DEPTH(T) + DEPTH(T').$$

Define tree T_1 to be $T_0 \times T^t$ where T^t is defined as the binary tree in which every internal node has 2 children and every path from the root to a leaf has length exactly $t+1$. Hence T^t has exactly 2^{t+1} leaves. Let us assume the nodes in T^t are strings $s \in \{L, R\}^*$ such that the root of T^t is the empty string ϵ, and each internal node $s \in T^t$ has two children, sL and sR.

We want an assignment function VAL_1 on T_1 in analogy to VAL_0. Write $VAL_1(C, s)$ (instead of $VAL_1((C, s))$, which is correct but pedantic) for the value assigned to the node $(C, s) \in T_1$, $C \in T_0, s \in T^t$. Instead of integers, $VAL_1(C, s)$ is a balanced 4-ary number.

First, we need one more definition: recall that in section 5, we compute the product of two balanced p-ary numbers $u = u_1 \cdots u_n$, $v = v_1 \cdots v_n$ as the sum of $2n$ balanced p-ary numbers X_i, Y_i $(i = 1, \ldots, n)$. Let us call X_i and Y_i the ith and $(n+i)$th *summand* of the product of u and v. Clearly the summands depend on the particular representation of the numbers $(u)_p$ and $(v)_p$. These $2n$ summands can be regarded as $2n$ digits numbers although they each have at most $2n-1$ digits of significance.

Suppose C is a leaf in T_0, $level(C) = \ell$. Then for any $s \in T^t$, $VAL_1(C, s)$ is defined to be the (ordinary) 4-ary representation of

$$2^{2^{t-\ell}} \text{ or } 0$$

depending on whether C is YES or not. Next assume $C \vdash (C_L, C_R)$. There are two cases:

(1) C is a $\otimes$-configuration.

(1.1) If s is a leaf of T^t. Then s is a string in $\{L,R\}^*$ of length $t+1$. Then $VAL_1(C,s)$ is the sth summand of the product of $VAL_1(C_L,\epsilon)$ and $VAL_1(C_R,\epsilon)$. Here s is interpreted as the 2-adic number (see chapter 1, section 4.2) where the symbols L,R in s are (respectively) interpreted as 1,2. Note that s ranges from 0 to 2^{t+1} and by definition the 0th summand is 0.

(1.2) If s is not a leaf of T^t then $VAL_1(C,s) = VAL_1(C,sL) + VAL_1(C,sR)$.

(2) C is a $\bigoplus$-configuration.

(2.1) If s is a leaf of T^t then let

$$VAL_1(C,s) = 2^{2^{t-\ell-1}-1}[VAL_1(C_L,\epsilon) + VAL_1(C_R,\epsilon)].$$

Note that we are multiplying by a power of 2 and this is relatively trivial in balanced 4-ary notation. Basically we must reexpress each balanced 4-ary digit as a pair of balanced 2-ary digit, shift these to the right by $2^{t-\ell-1}-1$ positions. Then we recombine into balanced 4-ary digits.

(2.2) If s is not a leaf then $VAL_1(C,s) = VAL_1(C,sL)(= VAL_1(C,sR))$.

It is not hard to show that for all $C \in T_0$,

$$VAL_0(C) = VAL_1(C,\epsilon).$$

We note that VAL_1 is uniquely defined since the balanced p-ary representation at the leaves are uniquely specified, and this propagates to all other nodes using the above rules.

Now it should be easy for the reader to use the technique of chapter 7 to provide an alternating machine that guesses the tree T_1 in order to determine the predicate

$$DIGIT(C,s,i,b)$$

that is true if the ith digit of $VAL_1(C,s)$ is equal to b, for $|b| \leq 3$ and $i = 1,\ldots,2^t$. To invoke this procedure, we may assume the work tapes of N contains the following information:

1. i in binary
2. C
3. s
4. $level(C)$ in unary
5. $t - level(C) + |s|$ in unary

Note that each of these uses $O(t)$ space. Moreover, from the above information, we can generate the arguments for the recursive calls in $O(t)$ steps. So the total work spend along any path in T_1 is $O(t^3)$ since T_1 has $O(t^2)$ levels. It is now clear that $DIGIT$ can be solved in $O(t^2)$ alternating time.

The final work to be done is to compare $VAL_1(C_0,\epsilon)$ to $\frac{1}{2}$ where C_0 is the root of T_0. Basically, our goal is to convert the balanced 4-ary number

$$VAL_1(C_0,\epsilon) = u_1u_2\cdots u_m \quad (m = 2^t)$$

to an ordinary 4-ary number

$$v_1v_2\cdots v_m.$$

We begin with a simple remark: since each of the digits v_i must be non-negative, if u_i is negative, we must borrow one unit from the next digit u_{i+1}. Let $b_i = 1$ or 0 depending on whether we need to borrow from u_{i+1} or not in order to make v_i non-negative. Of course, we must also take into account the borrow b_{i-1} from u_i. This gives us the equation

$$b_i = \begin{cases} 1 & \text{if } u_i - b_{i-1} < 0 \\ 0 & \text{if } u_i - b_{i-1} \geq 0 \end{cases}.$$

We set $b_0 = 0$. Note that this method is correct because we know that *apriori* that the number $(u_1\cdots u_m)_4$ is non-negative: so the most significant non-zero digit is positive. It is not hard to reduce the above to: $b_i = 1$ *iff for some* j $(1 \leq j \leq i)$, $u_j < 0$ *and for all* $k = j+1,\ldots,i$, $u_k = 0$. Hence we can in $O(t^2)$ alternating time

check the value of any b_i for $i = 1, \ldots, 2^t$: we simply guess j and then universally check that $u_j < 0$ and $u_k = 0$ for $k = j+1, \ldots, i$. Of course, checking if $u_k = b$ is nothing but the subroutine $DIGIT(C_0, \epsilon, k, b)$ which can be determined in $O(t^2)$ time.

Since we can check for the value of borrow bits b_i, we can check the digits v_i in the 4-ary representation of $VAL_1(C_0, \epsilon)$ via $v_i = u_i - b_{i-1} + 4b_i$. Now it is easy to determine if $VAL_1(C_0, \epsilon)$ is greater than 2^{2^t-1}.

We must address one more detail. The above construction did not consider the other basis functions of a stochastic alternating machine: $\oplus, \wedge, \vee$. However, it should be evident that since we know how to add and to multiply, we can also compute the value

$$VAL_0(C) = 2^{2^{t-\ell-1}-1}(\,VAL_0(C_L) + VAL_0(C_R)) - VAL_0(C_L)\,VAL_0(C_R)$$

where $\ell = level(C)$, C is a $\oplus$-configuration and $C \vdash (C_L, C_R)$. The remaining MIN- and MAX-configurations are also easy. This completes our proof of theorem 50.

Space-bounded Stochastic computation. We consider an $s(n)$ space-bounded stochastic machine M. In analogy with Markov chains, we set up a finite number of *dynamical states*, each corresponding to a configuration of the machine M using space at most s. If the set of states is taken to be $\{1, \ldots, n\}$, a valuation V can be viewed as an n-vector

$$V = (v_1, \ldots, v_n) \in [0,1]^n.$$

Let V_0 denote the vector of zero elements. We have the usual operator $\tau = \tau_\Delta$ where $\Delta = \{1, \ldots, n\}$. Thus

$$\tau(V) = (\tau_1(V), \tau_2(V), \ldots, \tau_n(V))$$

where $\tau_i(V) = 0, 1$ or $v_j \circ v_k$ for some j, k depending on i, $\circ \in \{\oplus\!\!\!\!\bigcirc, \otimes, \oplus\}$. In chapter 7, we showed that the limit of the sequence

$$\tau(V_0), \tau^2(V_0), \tau^3(V_0), \ldots$$

is the least fixed point of our system. Let

$$V^*_\tau = (v^*_1, \ldots, v^*_n)$$

denote this limit. Clearly any fixed point $V = (v_1, \ldots, v_n)$ of τ satisfies the following set of equations: each $i = 1, \ldots, n$,

$$v_i = f_i(v_{j(i)}, v_{k(i)}) \tag{25}$$

where $f_i(x, y)$ is one of the stochastic functions $0, 1, x \oplus\!\!\!\!\bigcirc\, y, x \otimes y, x \oplus y$. Let $\Sigma(V)$ denote the set of equations (25). We can then characterize the least fixed point property V^*_τ as follows:

$$LFP(V^*_\tau) \equiv \Sigma(V^*_\tau) \wedge (\forall V)[\Sigma(V). \Rightarrow .V^*_\tau \leq V].$$

We are now ready to prove the following, by appeal to some results on the complexity of real closed fields. (cf. Renegar [25, 26, 27]).

THEOREM 51 *For all $s(n)$,*

$$StSPACE(s) \subseteq DTIME(2^{2^{O(s)}}).$$

Proof. Suppose M is a stochastic machine that accepts in space $s(n)$. We show how to decide if M accepts any input w in space $s(|w|)$. As usual, we can assume $s = s(|w|)$ is known, and let there be n configurations of M that uses space at most s. Without loss of generality, let these configurations be identified with the integers $1, 2, \ldots, n$ and 1 denotes the initial configuration on input w. Let τ be the operator corresponding of these configurations. Hence we want to accept iff the least fixed point V^*_τ of τ has the property that its first component $[V^*_\tau]_1$ greater than $\frac{1}{2}$. This amounts to checking the validity of the following sentence:

$$(\exists V^*_\tau)(\forall V)[\Sigma(V^*_\tau) \wedge [V^*_\tau]_1 > \frac{1}{2} \wedge (\Sigma(V). \Rightarrow .V^*_\tau \leq V)].$$

This sentence can be decided in time $2^{O(n^4)} = 2^{2^{O(s)}}$, using the above results of Renegar. **Q.E.D.**

Appendix A

Probabilistic Background

The original axiomatic treatment of probability theory due to Kolmogorov [18] is still an excellent rapid introduction to the subject. We refer to [5, 34] for more advance techniques useful for complexity applications. This appendix is a miscellany of quick reviews and useful facts.

Basic Vocabulary. A *Borel field* (or sigma-field) is a set system (Ω, Σ) where Ω is a set and Σ is a collection of subsets of Ω with three properties (i) $\Omega \in \Sigma$, (ii) $E \in \Sigma$ implies $\Omega - E \in \Sigma$ and (iii) $\{E_i\}_{i\geq 0}$ is a countable collection of sets in Σ then the countable union $\cup_{i\geq 0} E_i$ is in Σ. If (iii) is replaced by the weaker condition that $E_1, E_2 \in \Sigma$ imples $E_1 \cup E_2 \in \Sigma$ then we get a *field*. For any collection S of subsets of Ω, there is a unique smallest Borel field that contains S, called the Borel field *generated by* S. The most important example is the *Euclidean Borel field* (R^1, B^1) where $R^1 = \mathbb{R}$ is the real line and B^1 is the Borel field generated by the collection of intervals $(-\infty, c]$ for each real c, $-\infty < c < +\infty$. Members in B^1 are called *Euclidean Borel sets*.

A *probability measure on a Borel field* (Ω, Σ) is a function $\Pr : \Sigma \to [0, 1]$ such that (a) $\Pr(\Omega) = 1$, (b) if $\{E_i\}$ is a countable collection of pairwise disjoint sets in Σ then $\Pr(\cup_{i\geq 0} E_i) = \sum_{i\geq 0} \Pr(E_i)$. A *probability space* is a triple $(\Omega, \Sigma, \Pr)$ where (Ω, Σ) is a Borel field and Pr is a probability measure on (Ω, Σ).

The elements in Ω are often called *elementary events* or *sample points*. Elements of Σ are called *events* or *measurable sets*. Thus Ω and Σ are called (respectively) the *sample space* and *event space*. $\Pr(E)$ is the *probability* or *measure* of the event E. A simple example of probabilistic space is the case $\Omega = \{H, T\}$ with two elements and Σ consists of all subsets of Ω (there are only 4 subsets), and Pr is defined by $\Pr(\{H\}) = p$ for some $0 \leq p \leq 1$.

A *random variable* (abbreviation: r.v.) X of a probability space $(\Omega, \Sigma, \Pr)$ is a real (possibly taking on the values $\pm\infty$) function with domain Ω such that for each real number c, the set

$$X^{-1}((-\infty, c]) = \{\omega \in \Omega \,:\, X(\omega) \leq c\}$$

belongs to Σ. We may simply write

$$\{X \leq c\}$$

for this event. In general, we write[1]

$$\{\ldots X \ldots\}$$

for the event $\{\omega : \ldots X(\omega) \ldots\}$. It is also convenient to write $\Pr\{X \in S\}$ instead of $\Pr(\{X \in S\})$. The intersection of several events is denoted by writing the defining conditions in any order, separated by commas: $\{X_1 \in S_1, X_2 \in S_2, \ldots\}$. If $f(x, y)$ is a real function and X, Y are random variables, then $f(X, Y)$ is a new function on Ω given by $f(X, Y)(\omega) := f(X(\omega), Y(\omega))$. If $f(x, y)$ is "nice", then $f(X, Y)$ will be a new random variable. In particular, the following are random variables:

$$\max(X, Y), \quad \min(X, Y), \quad X + Y, \quad X - Y, \quad XY, \quad X^Y, \quad X/Y.$$

The last case assumes Y is non-vanishing. Similarly, if X_i's are random variables, then so are

$$\inf_i X_i, \quad \sup_i X_i, \quad \liminf_i X_i, \quad \limsup_i X_i.$$

Each X induces a probability measure $\Pr_X$ on the Euclidean Borel field determined uniquely by the condition $\Pr_X((-\infty, c]) = \Pr\{X \leq c\}$. We call $\Pr_X$ the *probability measure* of X. The *distribution function* of X is the real

[1] This '$\{\cdots\}$' notation for events reflects the habit of probabilists to keep the event space implicit. Notice that while probability measures are defined on Σ, random variables are defined on Ω.

function given by $F_X(c) := \Pr_X((-\infty, c])$. Note that $F_X(-\infty) = 0, F_X(\infty) = 1$, F_X is non-decreasing and right continuous. In general, any F with these properties is called a distribution function, and determines a random variable. A set of random variables is *identically distributed* if all members shares a common distribution function F. A finite set of random variables $\{X_i : i = 1, \ldots, n\}$ is *independent* if for any Euclidean Borel sets B_i $(i = 1, \ldots, n)$,

$$\Pr(\cap_{i=1}^n \{X_i \in B_i\}) = \prod_{i=1}^n \Pr\{X_i \in B_i\}.$$

An infinite set of random variables is independent if every finite subset is independent. An important setting for probabilistic studies is a set of independent and identically distributed random variables, abbreviated as *i.i.d.*.

Let (Ω, Σ) be a field, and we are given $m : \Sigma \to [0, 1]$ such that for any countable collection of pairwise disjoint sets $\{E_i \in \Sigma : i \in I\}$,

$$E = \cup_{i \in I} E_i \in \Sigma \text{ implies } m(E) = \sum_{i \in I} m(E_i).$$

Then a standard theorem of Carathéodory says that m can be uniquely extended to a probability measure on (Ω, Σ^*), the Borel field generated Σ.

A standard construction shows that for any countable set of probability measures $\{m_i : i \geq 0\}$ on the Euclidean Borel field, we can construct a probability space $(\Omega, \Sigma, \Pr)$ and a collection of random variables $\{X_i : i \geq 0\}$ such that for each i, m_i is the probability measure of X_i. *Sketch*: We let Ω be the product of countably many copies of the real line $R^1 = \mathbb{R}$, so a sample point is $(w_0, w_1, \ldots)$ where $w_i \in R^1$. A *basic set* of Ω is the product of countably many Euclidean Borel sets $\prod_{i \geq 0} E_i$ where all but a finite number of these E_i are equal to R^1. Let Σ_0 consists of finite unions of basic sets and then our desired Borel field Σ is the smallest Borel field containing Σ_0. It remains to define Pr. For each basic set, define $\Pr(\prod_{i \geq 0} E_i) := \prod_{i \geq 0} \Pr(E_i)$ where only a finite number of the factors $\Pr(E_i)$ are not equal to 1. We then extend this measure to Σ_0 since each member of Σ_0 is a finite union of disjoint basic sets. This measure can be shown to be a probability measure on Σ_0. The said theorem of Carathéodory tells us that it can be uniquely extended to Σ. This concludes our sketch.

A random variable is *discrete* if it takes on a countable set of distinct values. In this case, we may define its *expectation* of X to be $E[X] := \sum_i a_i \Pr\{X = a_i\}$ where i range over all the distinct values a_i assumed by X. Note that $E[X]$ may be infinite. The *variance* of X is defined to be $Var[X] := E[(X - E[X])^2]$. This is seen to give $Var[X] = E[X^2] - (E[X])^2$.

A fundamental fact is that $E[X + Y] = E[X] + E[Y]$ where X, Y are *arbitrary* random variables. Using this simple fact, one often derive surprisingly consequences. In contrast, $Var[X + Y] = Var[X] + Var[Y]$ and $E[XY] = E[X]E[Y]$ are valid provided X, Y are independent random variables.

A random variable X that is 0/1-valued is called a *Bernoulli random variable*. The distribution function of such an X is denoted $B(1, p)$ if $\Pr\{X = 1\}$ is p. If $X_1, \ldots, X_n$ is a set of i.i.d. random variables with common distribution $B(1, p)$ then the random variable $X = X_1 + \cdots + X_n$ has the *binomial distribution* denoted by $B(n, p)$. It is straightforward to calculate that $E[X] = np$ and $Var[X] = np(1 - p)$ if X has distribution $B(n, p)$. Note that Bernoulli random variables is just another way of specifying events, and when used in this manner, we call the random variable the *indicator function* of the event in question. Furthermore, the probability of an event is just the expectation of its indicator function.

Estimations. Estimating probabilities is a fine art. There are some tools and inequalities that the student must become familiar with.

(a) One of these, Stirling's formula in the form due to Robbins (1955), should be committed to memory:

$$n! = \left(\frac{n}{e}\right)^n e^{\alpha_n} \sqrt{2\pi n}$$

where

$$\frac{1}{12n + 1} < \alpha_n < \frac{1}{12n}.$$

Sometimes, the alternative bound $\alpha_n > (12n)^{-1} - (360n^3)^{-1}$ is useful [10]. Using these bounds, it is not hard to show [21] that for $0 < p < 1$ and $q = 1 - p$,

$$G(p, n) e^{-\frac{1}{12pn} - \frac{1}{12qn}} < \binom{n}{pn} < G(p, n) \tag{1}$$

where

$$G(p, n) = \frac{1}{\sqrt{2\pi pqn}} p^{-pn} q^{-qn}.$$

(b) The 'tail of the binomial distribution' is the following sum

$$\sum_{i=\lambda n}^{n} \binom{n}{i} p^i q^{n-i}.$$

We have the following upper bound [10]:

$$\sum_{i=\lambda n}^{n} \binom{n}{i} p^i q^{n-i} < \frac{\lambda q}{\lambda - p} \binom{n}{\lambda n} p^{\lambda n} q^{(1-\lambda)n}$$

where $\lambda > p$ and $q = 1 - p$. This specializes to

$$\sum_{i=\lambda n}^{n} \binom{n}{i} < \frac{\lambda}{2\lambda - 1} \binom{n}{\lambda n} 2^{-n}$$

where $\lambda > p = q = 1/2$.
(c) A useful fact is this: for all real x,

$$e^{-x} \geq 1 - x$$

with equality only if $x = 0$. If $x \geq 1$ then this is trivial. Otherwise, by the usual series for the exponential function, we have that for all real x

$$e^{-x} = \sum_{i=0}^{\infty} \frac{(-x)^i}{i!} = (1 - x) + \frac{x^2}{2!}(1 - \frac{x}{3}) + \frac{x^4}{4!}(1 - \frac{x}{5}) + \cdots.$$

The desired bound follows since $x < 1$. Similarly, we have

$$e^{-x} = (1 - x + x^2/2) - \frac{x^3}{3!}(1 - \frac{x}{4}) - \frac{x^5}{5!}(1 - \frac{x}{6}) - \cdots.$$

Then

$$e^{-x} < 1 - x + x^2/2 = 1 - x(1 - x/2)$$

provided $0 \leq x \geq 4$. If $0 \leq x \leq 1$ then we conclude $e^{-x} < 1 - x/2$. (d) Jensen's inequality. Let $f(x)$ be a convex real function. Convexity of $f(x)$ means $f(\sum_i p_i x_i) \leq \sum_i p f(x_i)$ where $\sum_i p_i = 1, p_i \geq 0$ for all i, and i ranges over a finite set. If X and $f(X)$ are random variables then $f(E[X]) \leq E[f(X)]$. Let us prove this when X has takes on finitely many values x_i with probability p_i: so $E[X] = \sum_i p_i x_i$ and

$$f(E[X]) = f(\sum_i p_i x_i) \leq \sum_i p_i f(x_i) = E[f(X)].$$

For instance, if $r > 1$ then $E[|X|^r] \geq (E[|X|])^r$.

(e) Markov's inequality. Let X be a non-negative random variable, $e > 0$. Then we have the trivial inequality $H(X - e) \leq \frac{X}{e}$ where $H(x)$ (the Heaviside function) is the 0-1 function given by $H(x) = 1$ if and only if $x > 0$. Taking expections on both sides, we get

$$\Pr\{X > e\} \leq \frac{E[X]}{e}.$$

(f) Chebyshev's inequality. Let X be a discrete random variable, $\Pr\{X = a_i\} = p_i$ for all $i \geq 1$, with finite second moment and $e > 0$. Then

$$\Pr\{|X| \geq e\} \leq \frac{E[X^2]}{e^2}.$$

We say this gives an upper bound on tail probability of X. In proof,

$$\begin{aligned} E[X^2] &= \sum_{i \geq 1} p_i a_i^2 \\ &\geq e^2 \sum_{|a_i| \geq e} p_i \\ &= e^2 \Pr\{|X| \geq e\} \end{aligned}$$

Another form of this inequality is

$$\Pr\{|X - E[X]| > e\} \leq \frac{Var(X)}{e^2}$$

where $|X - E[X]|$ measures the deviation from the mean. We could prove this as for Markov's inequality, by taking expectations on both sides of the inequality

$$H(|X - E[X]| - e) \leq \left(\frac{X - E[X]}{e}\right)^2.$$

(g) Chernoff's bound [7] is concerned a set of i.i.d. random variables $X_1, \ldots, X_n$. Let $X = X_1 + \cdots + X_n$ and assume $E[X]$ is finite. Define

$$M(t) := E[e^{tX_1}]$$

and

$$m(a) = \inf_t E[e^{t(X_1 - a)}] = \inf_t e^{-at} M(t).$$

Then Chernoff showed that

$$E[X] \geq a \Rightarrow \Pr\{X \leq na\} \leq [m(a)]^n$$

and

$$E[X] \leq a \Rightarrow \Pr\{X \geq na\} \leq [m(a)]^n.$$

In particular, if X has distribution $B(n, p)$ then it is not hard to compute that

$$m(a) = \left(\frac{p}{a}\right)^a \left(\frac{1-p}{1-a}\right)^{1-a}.$$

Since it is well-known that $E[X] = np$, we obtain for $0 < e < 1$:

$$\Pr\{X \leq (1-e)np\} \leq \left(\frac{1}{1-e}\right)^{(1-e)np} \left(\frac{1-p}{1-(1-e)p}\right)^{n-(1+e)np}.$$

Markov Chains. We continue the discussion of Markov chains from section 3. The *period* of a state i in a chain A is defined in a combinatorial way: it is the largest positive integer d such that every cycle in the underlying graph G_A that contains i has length divisible by d. A state is *periodic* or *aperiodic* depending on whether its period is greater than 1 or not. It is left as an exercise to show that the period is a property of a component.

Recall that state i is recurrent if $f_i^* = 1$. In this case, there is certainty in returning to state i, and under this condition, we may speak of the *mean recurrence time for state* i μ_i, defined as follows:

$$\mu_i = \sum_{n=0}^{\infty} n f_i^{(n)}$$

Using the mean recurrence time, we may introduce new classification of states: state i is *null* if $\mu_i = \infty$, and *non-null* otherwise.

To illustrate the classification of states, we consider the (1-dimensional) random walk with parameter p_0 ($0 < p_0 < 1$): this is the Markov chain whose states are the integers, and the transition probability is given by $p_{i,i+1} = p_0$ and $p_{i,i-1} = 1 - p_0$, for all i. It is clear that every state is essential. It can be shown that each Markov state if recurrent or transient depending on whether $p_0 = \frac{1}{2}$ or not (Exercise). So state 0 is recurrent iff $p = \frac{1}{2}$. Thus $p_0 \neq \frac{1}{2}$ provide examples of essential but transient states. In the recurrent situation, the mean recurrence time is infinite (Exercise). So this illustrates recurrent but null states.

Generating functions. A (real) formal power series is an infinite expression $G(s) = \sum_{n=0}^{\infty} a_n s^n$ in some indeterminate s, where $a_0, a_1, \ldots$ are given real numbers. We say that $G(s)$ is the (ordinary) generating function for the seqence $a_0, a_1, \ldots$. We can manipulate $G(s)$ algebraically: we may add, multiply or (formally) differentiate power series in the obvious way. One should think of $G(s)$ as a convenient way to simultanueously manipulate all the elements of a sequence; hence the terms s^n are just 'place-holders'. These operations reflect various combinatorial operations on the original series. Using well-known identities we can deduce many properties of such series rather transparently. Although we have emphasized that the manipulations are purely formal, we occasionally try to sum the series for actual values of s; then one must be more careful with the analytic properties of these series. Most elementary identities involving infinite series reduces (via the above manipulations) to the following most

fundamental identity $(1-x)^{-1} = \sum_{i\geq 1} x^i$. For example, the student observes that all the results from sections 3 and 4 involving limits is basically an exploitation of this identity.

Rate of Convergence of Substochastic matrices. In section 3, we showed that the entries of the nth power of the fundamental matrix of an absorbing chain approaches 0. We now give a more precise bound on the rate of convergence. For any matrix B, let $\delta_*(B)$ and $\delta^*(B)$ denote the smallest and largest entries in B. Let $\delta(B) = \delta^*(B) - \delta_*(B)$. We have the following simple lemma (cf. [16]):

LEMMA 52 *Let $A = (a_{i,j})$ be a stochastic matrix each of whose entries are at least e for some $0 < e < 1$. For any $n \times m$ non-negative matrix $B = (b_{i,j})$, we have*

$$\delta(AB) \leq (1-2e)\delta(B).$$

Proof. Consider the (i,j)th entry $\sum_{k=1}^n a_{i,k}b_{k,j}$ of AB. Without loss of generality, assume that $a_{i,1} \leq a_{i,2}$.

To obtain a lower bound on the (i,j)th entry, assume wlog that $\delta^*(B) = \max\{b_{1,j}, b_{2,j}\}$. Then

$$\begin{aligned}\sum_{k=1}^n a_{i,k}b_{k,j} &\geq a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + (\sum_{k=3}^n a_{i,k})\delta_*(B)\\ &\geq a_{i,1}\delta^*(B) + a_{i,2}\delta_*(B) + (\sum_{k=3}^n a_{i,k})\delta_*(B)\end{aligned}$$

where the last inequality must be justified in two separate cases (in one case, we use the simple fact that $a \geq b$ and $a' \geq b'$ implies $aa' + bb' \geq b' + a'b$). Thus

$$\begin{aligned}\sum_{k=1}^n a_{i,k}b_{k,j} &\geq a_{i,1}\delta^*(B) + (\sum_{k=2}^n a_{i,k})\delta_*(B)\\ &= e\delta^*(B) + (\sum_{k=2}^n a_{i,k})\delta_*(B) + (a_{i,1}-e)\delta^*(B)\\ &\geq e\delta^*(B) + (1-e)\delta_*(B)\end{aligned}$$

To obtain an upper bound on the (i,j)th entry, Assuming wlog that $\delta_*(B) = \min\{b_{1,j}, b_{2,j}\}$, we have

$$\begin{aligned}\sum_{k=1}^n a_{i,k}b_{k,j} &\leq a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + (\sum_{k=3}^n a_{i,k})\delta^*(B)\\ &\leq a_{i,1}\delta_*(B) + a_{i,2}\delta^*(B) + (\sum_{k=3}^n a_{i,k})\delta^*(B)\\ &\leq a_{i,1}\delta_*(B) + (\sum_{k=2}^n a_{i,k})\delta^*(B)\\ &= e\delta_*(B) + (\sum_{k=2}^n a_{i,k})\delta^*(B) + (a_{i,1}-e)\delta_*(B)\\ &\leq e\delta_*(B) + (1-e)\delta^*(B).\end{aligned}$$

The lemma follows since the difference between the largest and smallest entry of AB is at most

$$(e\delta_*(B) + (1-e)\delta^*(B)) - (e\delta^*(B) + (1-e)\delta_*(B)) \leq (1-2e)\delta(B).$$

Q.E.D.

In the exercises, we show how to extend this to substochastic matrix B, *i.e.*, each row sum in B is at most 1.

EXERCISE

Exercise A.0.1: Show the following inequalities ((i)-(iv) from [Kazarinoff]):

(i) $(1+\frac{1}{n})^n < (1+\frac{1}{n+1})^{n+1}$.

(ii) $(1+\frac{1}{n})^n < \sum_{k=0}^n \frac{1}{k!} < (1+\frac{1}{n})^{n+1}$.

(iii) $n! < (\frac{n+1}{2})^n$ for $n = 2, 3, \ldots$.

(iv) $(\sum_{i=1}^n x_i)(\sum_{i=1}^n \frac{1}{x_i}) \geq n^2$ where x_i's are positive. Moreover equality holds only if the x_i's are all equal.

(v) $n! < \left(\frac{12n}{12n-1}\right)(2\pi n)^{1/2}e^{-n}n^n$. (Use Robbin's form of Stirling's formula.) □

Exercise A.0.2:
(i) (Hölder's Inequality) If X and Y are random variables, and $1 < p < \infty$, $\frac{1}{p} + \frac{1}{q} = 1$ then

$$E[XY] \leq E[|XY|] \leq E[|X|^p]^{1/p} E[|Y|^q]^{1/q}.$$

When $p = 2$, this is the Cauchy-Schwartz inequality. (In case $Y \equiv 1$ we have $E[|X|] \leq E[|X|^p]^{1/p}$, which implies the Liapounov inequality: $E[|X|^r]^{1/r} \leq E[|X|^s]^{1/s}$ for $1 < r < s < \infty$.)
(ii) (Minkowski's Inequality)

$$E[|X+Y|^p]^{1/p} \leq E[|X|^p]^{1/p} + E[|Y|^p]^{1/p}.$$

(iii) (Jensen's inequality) If f is a convex real function, and suppose X and $f(X)$ are integrable random variables. Then $f(E[X]) \leq E[f(X)]$. (Note that convexity means that if $\sum_{i=1}^n c_i = 1$, $c_i > 0$, then $f(\sum i = 1^n c_i x_i) \leq \sum_{i=1}^n c_i f(x_i)$.) □

Exercise A.0.3: Construct the probability space implicitly associated with a Markov chain. □

Exercise A.0.4: For any positive integer k, construct a finite Markov chain with states $0, 1, \ldots, n$ such that states 0 has the value $k \leq p^*_{0,0} < k+1$. Try to minimize $n = n(k)$. □

Exercise A.0.5: In this exercise, we do not assume the Markov chain is finite. Show that the following are properties, though defined for individual states, are characteristics of components:
(i) Period of a Markov state.
(ii) Nullity of a Markov state. □

Exercise A.0.6: Show that $g_{i,j} = f^*_{i,j} g_{j,j}$. (From this, conclude that $g_{i,j} > 0$ if and only if $g_{i,j} = f^*_{i,j}$.) **Hint:** Write $g_{i,j} = \sum_{n=0}^\infty \Pr(A_n B_n C_n | D)$ where D is the event that the state at time 0 is i, A_n is the event that the states at times $1, \ldots, n-1$ are not equal to j, B_n is the event that the state at time n is equal to j, C_n is the event that the state at time s is equal to j for infinitely many $s > n$. Then $\Pr(A_n B_n C_n | D) = \Pr(C_n | A_n B_n D) \Pr(A_n B_n | D)$ But the Markov property implies $\Pr(C_n | A_n B_n D) = \Pr(C_n | D)$. □

Exercise A.0.7: Above, we proved a bound on the rate of convergence of stochastic matrices. Extend it to substochastic matrices. □

Exercise A.0.8:
(a) Prove equation (1) in the appendix.
(b) Say $f(n) \sim g(n)$ if $f(n)/g(n)$ approaches 1 as $n \to \infty$. Conclude that for $k = 1, \ldots, n/2$, $p = k/n$ and $q = 1 - p$, then as $k \to \infty$ and $n - k \to \infty$:

$$\left(\begin{array}{c} n \\ k \end{array} \right) \sim \frac{1}{\sqrt{2\pi pqn}(p^p q^q)^n}$$

(c) Let $0 < p < 1$ and $q = 1 - p$. Show that the probability that a Bernoulli random variable with mean p attains k successes in n trials is

$$\left(\begin{array}{c} n \\ k \end{array} \right) p^k q^{n-k} \sim \frac{1}{\sqrt{2\pi pqn}}$$

□

Exercise A.0.9: Show
(a)

$$\left(\frac{1-p}{1-\delta p} \right)^{1-\delta p} \leq e^{\delta - 1}$$

for $0 < \delta < 2$.
(b)

$$\left(\frac{1}{1+e} \right)^{1+e} \leq e^{-e-(e^2/3)}$$

for $0 < e \leq 1$.
(c)

$$\left(\frac{1}{1-e} \right)^{1-e} \leq e^{e-(e^2/2)}$$

for $0 < e \leq 1$.
(d) Conclude that in the binomial case of Chernoff's inequality (see appendix),

$$\Pr\{X \geq (1+e)np\} \leq e^{-(e^2/3)np}$$

and

$$\Pr\{X \leq (1-e)np\} \leq e^{-(e^2/2)np}.$$

□

Exercise A.0.10: Deduce from Chernoff's bound the following estimate on the tail of binomial distribution:

$$\sum_{i=\lfloor t/2 \rfloor}^{t} \binom{t}{i} p^i q^{t-i} \leq (4pq)^{t/2}.$$

□

End Exercise

Bibliography

[1] A. Avizienis. Signed-digit number representation for fast parallel arithmetic. *Inst. Radio Engr. Trans. Electron. Comput.*, 10:389–400, 1961.

[2] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computers and Systems Science*, 36:254–276, 1988.

[3] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. *J. of Computer and System Sciences*, 50(2):191–202, 1995.

[4] S. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984.

[5] B. Bollobás. *Random Graphs*. Academic Press, 1985.

[6] A. Borodin, S. Cook, and N. Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Computation*, 58:113–136, 1983.

[7] H. Chernoff. A measure of asymptotic efficiency for tests of hypothesis based on sum of observations. *Ann. of Math. Stat.*, 23:493–507, 1952.

[8] K. L. Chung. *Markov Chains with stationary transition probabilities*. Springer-Verlag, Berlin, 1960.

[9] K. de Leeuw, E. F. Moore, C. E. Shannon, and N. Shapiro. *Computability by probabilistic machines*. Automata Studies. Princeton, New Jersey, 1956.

[10] W. Feller. *An introduction to Probability Theory and its Applications*. Wiley, New York, 2nd edition edition, 1957. (Volumes 1 and 2).

[11] F. R. Gantmacher. *The Theory of Matrices*. Chelsea Pub. Co., New York, 1959. Volumes 1 and 2.

[12] J. T. Gill. Computational complexity of probabilistic Turing machines. *SIAM J. Comp.*, 6(4):675–695, 1977.

[13] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proofs. *17th ACM Symposium STOC*, pages 291–304, 1985.

[14] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. *18th ACM Symposium STOC*, pages 59–68, 1986.

[15] H. Jung. Relationships between probabilistic and deterministic tape complexity. *10th Sympos. om Mathematical Foundations of Comp. Sci.*, pages 339–346, 1981.

[16] J. G. Kemeny and J. L. Snell. *Finite Markov chains*. D. Van Nostrand, Princeton, N.J., 1960.

[17] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhauser, 1993.

[18] A. N. Kolmogorov. *Foundations of the theory of probability*. Chelsea Publishing Co., New York, 1956. Second English Edition.

[19] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, 1980.

[20] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[21] W. W. Peterson and J. E. J. Weldon. *Error-Correcting Codes*. MIT Press, 1975. 2nd Edition.

[22] N. Pippenger. Developments in "The synthesis of reliable organisms from unreliable components". manuscript, University of British Columbia, 1988.

[23] M. O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, New York, 1976.

[24] J. Radhakrishnan and S. Saluja. Lecture notes: Interactive proof systems. Research Report MPI-I-95-1-007, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, March 1995. This is a full TECHREPORT entry.

[25] J. Renegar. On the Computational Complexity and Geometry of the First-Order Theory of the Reals, Part I: Introduction. Preliminaries. The Geometry of Semi-Algebraic Sets. The Decision Problem for the Existential Theory of the Reals. *J. of Symbolic Computation*, 13(3):255–300, March 1992.

[26] J. Renegar. On the Computational Complexity and Geometry of the First-Order Theory of the Reals, Part II: The General Decision Problem. Preliminaries for Quantifier Elimination. *J. of Symbolic Computation*, 13(3):301–328, March 1992.

[27] J. Renegar. On the Computational Complexity and Geometry of the First-Order Theory of the Reals, Part III: Quantifier Eliminiation. *J. of Symbolic Computation*, 13(3):329–352, March 1992.

[28] W. L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computers and Systems Science*, 28:216–230, 1984.

[29] U. Schöning. *Complexity and Structure*, volume 211 of *Lecture Notes in Computer Science*. Springer-Verlag, 1986.

[30] A. Shamir. IP=PSPACE. *J. of the ACM*, 39(4):869–877, 1992.

[31] J. Simon. On tape bounded probabilistic Turing machine acceptors. *Theor. Computer Science*, 16:75–91, 1981.

[32] J. Simon. Space-bounded probabilistic turing machine complexity are closed under complement. *13th Proc. ACM Symp. Theory of Comp. Sci.*, pages 158–167, 1981.

[33] R. Solovay and V. Strassen. A fast monte-carlo test for primality. *SIAM J. Computing*, 6:84–85, 1977.

[34] J. Spencer. *Ten Lectures on the Probalistic Method*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, 1987.