Notes for Lecture 6

Today we discuss checking equivalence of branching programs.

1 Branching Programs

We begin by introducing another model of computation, that of branching programs. A branching program accepts or rejects input strings $x_1, \ldots, x_n$ and can be described as a directed acyclic graph with a single root and two final states, 0 and 1. The computational path begins at the root. Each non-final vertex is labeled with some single variable x_i and has two outgoing edges, one labeled 0 and the other 1. The path follows edge b such that $x_i = b$, and the program accepts the input if and only if the path ends at the final state 1.

An illustrative example is the branching program that considers input string $x_1x_2x_3$ and outputs 1 if $x_1 \oplus x_2 \oplus x_3 = 1$. This is shown in figure 1.

Branching programs are a reasonably powerful model of computation; for example we can show that **L** can be computed by branching programs of polynomial size.

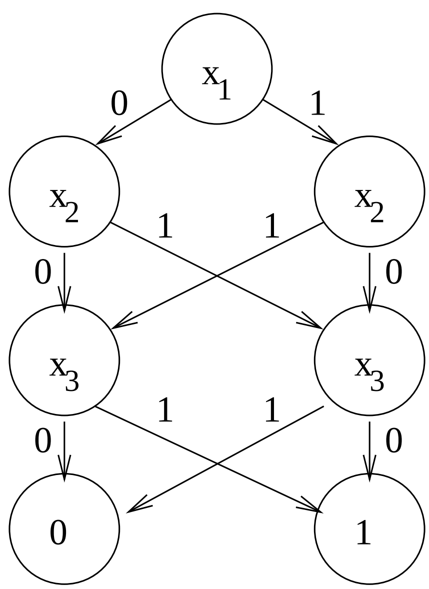

Figure 1: A branching program.

2 Testing Equivalence

We would like to be able to consider two branching programs and determine if they compute the same function. In general, testing equivalence for any model of computation is difficult:

for Turing machines, the problem is undecidable, and for circuits, the problem is **coNP**-complete. A similar result is true for branching programs– here, testing equivalence is also **coNP**-complete, through a reduction from co-3SAT.

Happily, the problem becomes easier once we restrict it to *read-once* branching programs, in which every path from the root to a final state tests each variable at most once. We will show that in this case testing equivalence is in **coRP**. Our strategy will be as follows: Given two read-once branching programs B_1 and B_2, we will define two polynomials p_1 and p_2 of degree n over $x_1, \ldots, x_n$ that are equivalent if and only if B_1 and B_2 are also equivalent. We will then use a **coRP**algorithm to test the equivalence of the two polynomials. The polynomials will be represented in compact form, but will have an exponential number of monomials and thus we cannot simply directly compare their coefficients.

Our polynomials will be determined as follows:

- We assign each vertex v_i in the branching program from the root to the final states a polynomial $p_i(x_1, \ldots, x_n)$, beginning with the root vertex which gets the constant polynomial 1.

- Once a vertex has been assigned a polynomial $p(x_1, \ldots, x_n)$, its outgoing edges are then given polynomials in the following manner: If the vertex performed a test on x_j, the outgoing 0 edge receives the polynomial $(1 - x_j)p(x_1, \ldots, x_n)$ and the outgoing 1 edge receives $(x_j)p(x_1, \ldots, x_n)$.

- Once all of a vertex's incoming edges have been appropriately labeled, that vertex's polynomial becomes the sum of the polynomials of the incoming edges.

- The polynomial associated with the final state 1 is the polynomial of the branching program.

These rules are illustrated in figure 2. The idea is that the polynomial at each vertex is nonzero on a set of inputs only if that set of inputs will lead the branching program to that vertex.

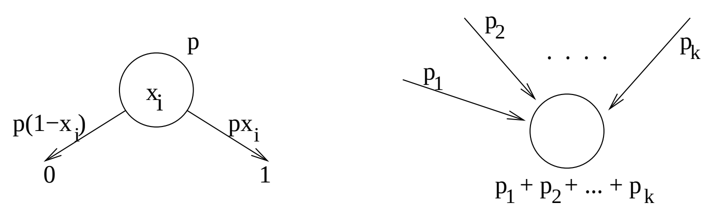

Figure 2: Constructing the polynomial of a branching program.

We can see that when the branching programs are read-once, the degree of a variable in any monomial in the resulting polynomial will have power no larger than 1, and hence the polynomial will be of degree at most n.

We now observe that given read-once branching programs B_1 and B_2, their corresponding polynomials p_1 and p_2 are equivalent if and only if B_1 and B_2 are equivalent. To see this, if any monomial m in p_1 or p_2 does not contain a variable x_i we replace m with $mx_i + m(1 - x_i)$. We can then write p_1 and p_2 as sums of terms of the form $\prod_{i=1}^{n} y_i$, where

y_i is either x_i or $(1 - x_i)$. Each term corresponds to a path in the branching program that ends in final state 1. From this, some extra thought reveals that equivalence of the polynomials is equivalent to equivalence of the branching programs.

Notice also that if the branching programs are polynomial size, we can evaluate the polynomials efficiently, even though they may contain an exponential number of coefficients. All we need now is a method of testing equivalence of polynomials efficiently.

3 The Schwartz-Zippel Lemma

The tool we will use for this is the Schwartz-Zippel lemma, which states:

Lemma 1 *If $p(x_1, \ldots, x_n)$ is an n-variate nonzero polynomial of degree d over a finite field $\mathbb{F}$, then p has at most $d\mathbb{F}^{n-1}$ roots. Equivalently, $\mathbf{Pr}[p(a_1, \ldots, a_n) = 0] \leq d/\mathbb{F}$.*

Notice that we use $\mathbb{F}$ to be both the field and its size.

Proof: The proof proceeds by induction on n. The base case of $n = 1$ is simply that of a univariate polynomial and follows immediately.

Now consider $p(x_1, \ldots, x_n)$ of degree at most d. We can write

$$p(x_1, \ldots, x_n) = p_0(x_2, \ldots, x_n) + x_1 p_1(x_2, \ldots, x_n) + x_1^2 p_2(x_2, \ldots, x_n) + \ldots + x_1^k p_k(x_2, \ldots, x_n)$$

where k is the largest value for which p_k is not identically zero.

We now observe that

$$|\{a_1, \ldots, a_n\} : p(a_1, \ldots, a_n) \neq 0| \geq |\{a_1, \ldots, a_n\} : p(a_1, \ldots, a_n) \neq 0 \text{ and } p_k(a_2, \ldots, a_n) \neq 0|$$

By induction, p_k is nonzero on at least $\mathbb{F}^{n-1} - (d-k)\mathbb{F}^{n-2}$ points. Further, for each such point $a_2, \ldots, a_n$, $p(x_1, a_2, \ldots, a_n)$ is a nonzero univariate polynomial and hence there are at least $\mathbb{F} - k$ values of a_1 such that $p(a_1, \ldots, a_n) \neq 0$.

Putting this together, we can bound the number of nonzero points from below by

$$\mathbb{F}^{n-1}\left(1 - \frac{d-k}{\mathbb{F}}\right)\mathbb{F}\left(1 - \frac{k}{\mathbb{F}}\right) \geq \mathbb{F}^n\left(1 - \frac{d}{\mathbb{F}}\right) = \mathbb{F}^n - d\mathbb{F}^{n-1}$$

This finishes the proof. □

An algorithm for testing equivalence of polynomials and hence read-once branching programs is now immediate. We choose a field of size at least $3d$, and then choose random $a_1, \ldots, a_n$ from $\mathbb{F}^n$. We accept if $p_1(a_1, \ldots, a_n) = p_2(a_1, \ldots, a_n)$.

If the two polynomials are equivalent, then we always accept; otherwise reject with probability at least 2/3, making this a **coRP** algorithm.

Exercises

1. Prove that testing equivalence of general branching programs is co**NP**-complete.
2. Show that every Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a read-once branching program (possibly of exponential size).