1 Today

1. Depth reduction in arithmetic circuits
2. Lower bounds on circuit size

2 Depth Reduction in Arithmetic Formulae

Recall the theorem of Ben-Or and Cleve [BOC92] from last time.

**Theorem 1** If $f$ is computed by a polynomial size formula, then $f$ can also be computed by a polynomial size register machine using only 3 registers.

Let us write $\text{BP-size}(f)$ (for branching process size) to denote the size of the smallest such 3 register machine computing $f$.

We essentially proved Theorem 1 by noting that

$$f = f_1 + f_2 \Rightarrow \text{BP-size}(f) \leq \text{BP-size}(f_1) + \text{BP-size}(f_2)$$

and

$$f = f_1 \times f_2 \Rightarrow \text{BP-size}(f) \leq 2[\text{BP-size}(f_1) + \text{BP-size}(f_2)].$$

This is not quite enough to prove the theorem, however, since it is not clear at this point how many times we will incur this factor 2 blow-up in the BP-size. In particular, this will only imply the result if we can argue that $f$ is in fact computed by a low depth formula.

The idea for depth reduction in arithmetic formulae is essentially the same as in the boolean case, which goes as follows. First, find an interior gate $v$ such that $(2/3)\text{size}(f) \geq \text{size}(v) \geq (1/3)\text{size}(f)$. This is always possible due to the tree structure of the circuit. Intuitively, the size conditions ensure that the sub-formula computing $v$ and the sub-formula computing $f$ conditioned on the value of $v$ are of comparable size, and so we should try to compute these in parallel.

To be precise, we create two copies of the sub-circuit computing $f$ conditioned on the value of $v$: one hard-wired with the value $v = 0$, and the other with $v = 1$. To compute $f$, then, we compute $v$ along with the conditional values of $f$ in parallel and output the correct value once $v$ is observed.

We cannot apply this program directly in the arithmetic setting, since the gate $v$ may take an infinite set of values, but the idea is essentially the same.
If we condition on knowing the value of \( v \), then \( f \) is a linear function of \( v \), i.e., \( f = Av + B \) for polynomials \( A \) and \( B \). Now we can compute \( A \), \( v \), and \( B \) in parallel as before to achieve the depth reduction. Thus, in the setting of Theorem 1, we may assume that the formula computing \( f \) has logarithmic depth, and hence we incur only a polynomial blow-up in the BP-size.

Things will be much more difficult with arithmetic circuits (as opposed to formulae) because in this case \( f \) may depend on a gate \( v \) in a significantly non-linear fashion.

### 3 Depth Reduction in Arithmetic Circuits

The main result here is the following theorem from [VSBR83].

**Theorem 2** If \( f \) is a polynomial of degree \( \leq d \) and is computed by a circuit of size \( \leq s \), then \( f \) can also be computed by a circuit of size \( \text{poly}(s, d) \) with depth \( \leq (\log s)(\log d) \).

We would like to proceed analogously to the case of arithmetic formulae, but in order to do this, we need a way to quantify how useful the partial functions \( f_v \) are for computing \( f \). The key here is to introduce a notion of partial derivative.

**Definition 3** Given an arithmetic circuit computing a function \( f \), and two gates \( v \) and \( w \), we write

\[
\partial_w(v) = \left. \frac{\partial f_{v,w}}{\partial w} \right|_{w=f_w},
\]

where \( f_{v,w}(x_1, \ldots, x_n, w) \) denotes the partial function \( f_v \) as a function of the value of the gate \( w \).

It is instructive to think of \( \partial_w(v) \) as a measure of the number of paths from \( w \) to \( v \) in the circuit. We will not make this statement entirely precise, but one useful fact is that if there are no paths from \( w \) to \( v \), then we do indeed have \( \partial_w(v) = 0 \).

Now, to compute \( f \), we will compute all the \( f_w \) and all the \( \partial_w(v) \) in some order. In particular, at stage \( i \), we will compute:

1. all the \( f_w \) with \( 2^i \leq \deg(f_w) \leq 2^{i+1} \)
2. all the \( \partial_w(v) \) with \( v \) and \( w \) satisfying \( 2^i \leq \deg(f_v) - \deg(f_w) \leq 2^{i+1} \) and \( \deg(f_w) \leq \deg(v) \leq 2 \deg(w) \).

If we can manage to do this, it is clear we will have computed \( f \) by stage \( \log d \) since \( f \) has degree \( \leq d \). Before we can specify the details of this computation, we need one more definition.

**Definition 4** We write

\[
\mathcal{G}_m = \{ t \mid \deg(f_t) > m, \ f_t = f_{t_1} \times f_{t_2}, \ \deg(f_{t_i}) \leq m \ \forall i \}.
\]

Now, Theorem 2 will follow from the following claim.

**Claim 5** For all \( v, w \) such that \( m < \deg(f_v) \leq 2m \) and \( \deg(f_v) \leq m < \deg(f_w) \leq 2 \deg(f_w) \), we have
1. \( f_v = \sum_{t \in G_m} f_t \partial_t(v) \)

2. \( \partial_w(v) = \sum_{t \in G_m} \partial_w(t) \partial_t(v) \)

Notice that, if we have computed all the values \( f_w \) and \( \partial_w(v) \) from stage \( i \) above, then it follows from Claim 5 that we can compute all the values in stage \( i + 1 \) using a circuit of depth at most \( \log s \) (this is to compute a sum of at most \( s \) values). Hence, since there can be at most \( \log d \) stages, the theorem follows.

**Proof** The proof will be by a fairly simple induction on the depth of the circuit. Also, note that it will suffice to prove (1), since (2) follows by applying \( \partial_w \) to both sides.

An easy case to pick off is when \( v \) itself lies in \( G_m \). In this case, we can write (1) as

\[
    f_v = f_v \partial_v(v) + \sum_{t \in G_m \setminus \{v\}} f_t \partial_t(v).
\]

But since there are no paths among elements of \( G_m \), \( \partial_t(v) = 0 \) for every \( t \), and further, \( \partial_v(v) = 1 \), and so (1) reduces simply to \( f_v = f_v \).

Now, there are two distinct cases to check for the induction, depending on whether gate \( v \) is an addition or a multiplication gate.

First, if \( f_v = f_{v_1} + f_{v_2} \), then by induction on \( v_1 \) and \( v_2 \) (here we assume that the circuit is suitably homogenized so that \( f_{v_1} \) and \( f_{v_2} \) both satisfy the degree conditions)

\[
    f_v = f_{v_1} + f_{v_2} = \sum_{t \in G_m} f_t \partial_t(v_1) + \sum_{t \in G_m} f_t \partial_t(v_2) = \sum_{t \in G_m} f_t \partial_t(v),
\]

since \( \partial_t \) is a linear operator when we conflate \( v \) with \( f_v \).

Next, if \( f_v = f_{v_1} \times f_{v_2} \), then either \( v_1 \) and \( v_2 \) both have degree \( \leq m \), in which case \( v \in G_m \) and we are done (see above), or one of the two gates (say, \( v_1 \)) has degree \( > m \) which in turn implies that \( v_2 \) has degree \( \leq m \). Thus, by induction we have

\[
    f_{v_1} = \sum_{t \in G_m} f_t \partial_t(v_1).
\]

However, by the product rule for partial derivatives, for any \( t \in G_m \) we have

\[
    \partial_t(v) = f_{v_1} \partial_t(v_2) + \partial_t(v_1) f_{v_2} = \partial_t(v_1) f_{v_2},
\]

since there are certainly no paths from \( t \) to \( v_2 \) (by considering degrees). Hence, (1) follows from \( f_{v_1} = \sum_{t \in G_m} f_t \partial_t(v_1) \) by multiplying through by \( f_{v_2} \). \( \blacksquare \)

### 4 Lower Bounds

At this point, we have seen some upper bounds on circuit size and depth by explicitly constructing or manipulating a circuit. On the other hand, it is generally much more difficult to give super-linear lower bounds for any reasonably expressive model of computation. For arithmetic circuits, at least, we will see that it is not so hard.

Consider a circuit computing the function \( f : \mathbb{F}^n \to \mathbb{F}^n \) that maps \((x_1, \ldots, x_n)\) to \((x_1', \ldots, x_n')\) for some fixed integer \( r \). One of the main results of Strassen
[Str75] is that this function has no arithmetic circuit of size smaller than $n \log r$. From this result, one can also show (see [BS83]) that the function sending $(x_1, \ldots, x_n, y_1, \ldots, y_r)$ to $\sum_{i=1}^r y_i x_i^*$ has no size $n \log r$ circuit.

The proof of Strassen is almost trivial, although it does rely on some (still reasonably easy) results from algebraic geometry. The idea is to write the gates of a circuit computing $f$ as polynomial constraints that equate their input to their output. Precisely, for an addition gate $v = u + w$ we add the constraint

$$y_v - (y_u + y_w) = 0,$$

and for a multiplication gate $v' = u' \times w'$ we add the constraint

$$y_{v'} - y_{u'} y_{w'} = 0.$$

Finally we add constraints enforcing the values of the output gates (to some fixed values).

If a circuit of size $s$ can compute $f$, then in this manner we obtain a system of $s$ polynomial equations that are either linear or quadratic. Now consider a primitive $r$-th root of unity $\omega$ in the field (this is essentially the only assumption that needs to be made on the field). The powers $1, \omega, \omega^2, \ldots, \omega^{r-1}$ are all distinct, and taking any $n$ of these (with replacement) as inputs to $f$ will evaluate to $(1, \ldots, 1)$. Hence, if we set all of the output gates to 1 in our polynomial constraints, the resulting system of polynomial equations has at least $r^n$ common zeros.

However, it follows from a result in algebraic geometry known as Bézout’s Theorem that a system of $s$ polynomial equations of degree $\leq 2$ has either $\leq 2^s$ common zeros, or else has an infinite number. Since all the solutions to $f(x_1, \ldots, x_n) = (1, \ldots, 1)$ are roots of unity, of which there are a finite number, the former case of Bézout’s Theorem applies and we obtain

$$r^n \leq 2^s \Rightarrow s \geq n \log r.$$

References


