6.S897 Algebra and Computation March 30, 2015

Lecture 14

Lecturer: Madhu Sudan *Scribe: Govind Ramnarayan*

1 Today: Arithmetic Circuits

- Models of computation
- Basic results

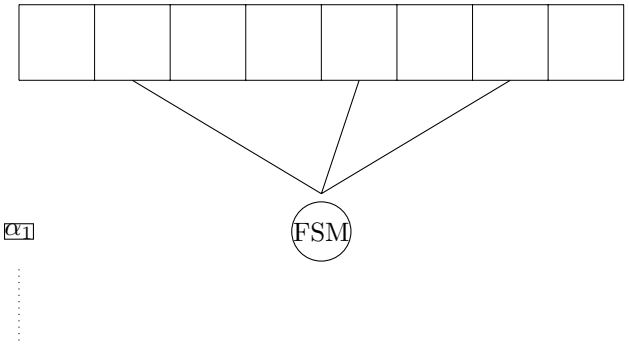
In order to begin to talk about arithmetic complexity, we'll need to talk about some arithmetic problems.

2 Problems

We'll be interested in two problems in particular:

1. Computing some function $\phi : \mathbb{F}^n \to \mathbb{F}^m$ of the form $\phi = (\phi_1, \ldots, \phi_m)$, where each $\phi_i \in \mathbb{F}[x_1, \ldots, x_n]$ is a polynomial.
   Examples include computing the determinant or permanent of an $n \times n$ matrix, which are both functions from $\mathbb{F}^{n \times n} \to \mathbb{F}$. This problem can also be phrased as the following: Given some $x \in \mathbb{F}^n$, compute $\phi(x)$.
   Note that it's not too easy to see how to express various natural problems, like gcd or division, as problems of this form. This class is fairly restricted. The next class is a bit more general.
2. Solving Polynomial Relations:
   $\phi : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}$
   Given: $x \in \mathbb{F}^n$ find y s.t. $\phi(x, y) = 0$.
   - This class covers root finding.
   - A complete problem for this class: Given $p_1, \ldots, p_m \in \mathbb{F}[x_1, \ldots, x_n]$, find a common zero.

3 Blum-Shub-Smale Model of Computation



The Blum-Shub-Smale model of computation is quite similar to that of a Turing Machine. We have a finite state machine that gets to use a tape, where the tape contents are elements of a field rather than bits. The machine can add, subtract, multiply, and divide, and it can branch on 0 (i.e. it can make a query "Is this 0?" and branch based on the result). Furthermore, the machine can have a constant number of internal constants $\alpha_1, \ldots, \alpha_c$. Note that for a finite field, this is comparable to a Turing Machine. However, for an

infinite field, this model may have more power.

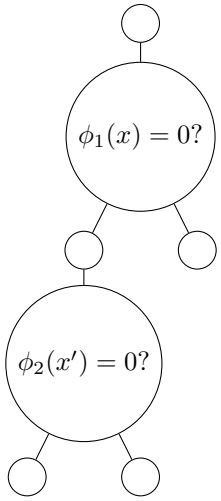
- Example: Complex Numbers
  - Model of computation has infinite precision.
  - But at the end, can only check if something is 0 or not.
  - A related aside: over the reals, we cannot even check inequalities in this model! Doing so could give us a lot more power.

Given such a machine M, let $L(M)$ denote the language of strings from our finite field $\mathbb{F}$ that are accepted. Formally:

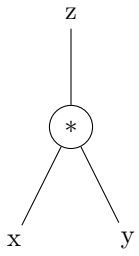
$L(M) \subseteq \{\mathbb{F}^n\}_{n \geq 0}$
$= \{x | M \text{ accepts } x\}$

What about $L(M) \cap \{0,1\}^*$? Can we say anything about that? In fact, we think that anything taking poly-time in this model is contained in BPP.

Below we give a visualization of the kind of computation done by the BSS model. Arithmetic operations do not branch, and we branch when we check if something is equal to 0.



The depth of the computation tree above gives the number of operations you perform. Note that the fan out can be exponential in the depth, as we could branch on every step. Furthermore, note that, if you chose to compute a polynomial, what you get is a circuit computing that polynomial. Finally, note that each step of computation can be represented by a low degree polynomial equation. For example, take the following gate:



This can be represented as the polynomial equation $z - xy = 0$. This will be notable in the later section on Hilbert-Nullstellensatz.

3.1 P and NP in the BSS Model

- BSS-P consists of all boolean functions $\phi_n : \mathbb{F} \to \{0,1\}$, $\{\phi_n\}_{n \geq 0}$ that are computable in polynomial time.

- BSS-NP consists of ψ_n such that $\psi_n = \{x | \exists y \text{ s.t. } \phi_{n,m}(x,y) = 0\}$, where $\phi_{n,m} : \mathbb{F}^n \times \mathbb{F}^m \to \{0,1\}$ is computable in polynomial time in the BSS model.

4 Hilbert-Nulstellensatz (HN)

- Input: Polynomials $p_1, \ldots, p_m \in \mathbb{F}[x_1, \ldots, x_n]$. Wlog we can take the p_i's to have degree 2.
- Accept if $\exists \alpha_1, \ldots, \alpha_n \in \mathbb{F}$ such that $p_i(\alpha_1, \ldots, \alpha_n) = 0$ for all i.

We assert that this problem is $\text{NP}_{\mathbb{F}}$-complete. Furthermore, note that in a large field, the α_i's may be very complicated to represent in bits! In fact, it is open if $NP_{\mathbb{C}} \cap \{0,1\}^* \subseteq NP$, though this is expected to be false. However, we do know that $\text{NP}_{\mathbb{C}} \cap \{0,1\}^* \subseteq$ PSPACE, and that, under the general Riemann hypothesis, $\text{NP}_{\mathbb{C}} \cap \{0,1\}^* \subseteq AM$ (in other words, $\text{NP}_{\mathbb{C}}$ has a 2-round interactive proof).

4.1 A Notable Tangent: Expressing Problems in HN Form

Consider the following problem: Given $P_1, \ldots, P_n, Q_1, \ldots, Q_n$ polynomials from $\mathbb{F}^m \to \mathbb{F}$, does there exists $x \in \mathbb{F}^m$ such that $P_i(x) = 0, Q_i(x) \neq 0$?
Note that we want the condition $Q_i(x) \neq 0$. However, we can still express this in HN form. Let $Q = \prod_i Q_i$. We can now rephrase our problem as: does there exist $x \in \mathbb{F}^m$ and $y \in \mathbb{F}$ such that $P_i(x) = 0$ and $1 - y \cdot Q(x) = 0$?

5 Arithmetic Circuits and Valiant's Classes

Here, we will just look at the circuits that compute polynomials. Arithmetic circuits are also known as *straight-line programs.*

Definition 1 (Informal) *An arithmetic circuit C over a field $\mathbb{F}$ consists of:*

- ***Input Variables**: $x_1, \ldots, x_n$*
- ***Gates**: Gates of the form $U \leftarrow V \diamond W$, where $\diamond \in \{+, -, *, \div\}$, and V and W are outputs of previous gates, and U is the output of this gate.*
- ***Output Variables:** Some outputs of gates $y_1, \ldots, y_m$.*

Definition 2 *The class "Valiant P", or VP, is as follows. VP = $\{\phi_n : \mathbb{F}^n \to \mathbb{F}^{m(n)}$:deg(ϕ_n) $\leq$poly(n), ϕ_n is computed by circuits of size poly(n)$\}$*

Some notes about VP:

- The restriction of low degree on the polynomial ϕ_n does not explicitly prevent the circuit that computes it from computing high degree polynomials in intermediate steps, but it turns out we can assume we have low degree polynomials in intermediate steps as well.

- This definition excludes the high degree polynomials we can compute in polynomial time, e.g. by repeated squaring.
- Turns out we can exclude division!
  - It's easy to see that we need at most 1 division in such a computation – we can keep the numerator and denominator separate, then divide at the end. Intuitively, the reason we can exclude division completely is that we are okay with doing a number of calculations proportional to the degree of the output, so we can substitute a division with the other operations using this computational power.
- Depth does not help. Every function in VP can be computed with a $\log^2 n$ depth circuit.
- Consider $\phi_i : \mathbb{F}^n \to \mathbb{F}^m$, such that $\phi_1, \ldots, \phi_m \in \mathbb{F}[x_1, \ldots, x_n]$.
  The complexity of computing these m polynomials is linearly related to the complexity of computing $\hat{\phi} : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}$, where

$$\hat{\phi}(x_1, \ldots, x_n, y_1, \ldots, y_m) = \sum_{j=1}^{m} y_j \phi_j(x_1, \ldots, x_n)$$

Now, we will define the equivalent of NP in the arithmetic complexity world. As intuition for this definition, note that the natural analog to the existential quantifier in the arithmetic world is the sum operation.

Definition 3 *The class "Valiant NP", or VNP, is as follows.* $VNP = \{\phi_n : \mathbb{F}^n \to \mathbb{F} s.t. \exists \psi_{n,m} : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F} \in VP s.t. \phi_n(x) = \sum_{y \in \{0,1\}^m} \psi_{n,m}(x, y)\}$.

This class is motivated by the Permanent function:

$$Perm(M) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} M_{i\sigma(i)}$$

A priori it is not clear that VNP includes the Permanent, as we are summing over $n!$ terms, while the definition of VNP only has us summing over 2^n terms. However, note that $\prod_{i=1}^{n} \sum_{j=1}^{m} M_{ij}$ includes all the terms of the permanent, and more! This motivates a smaller, inclusion-exclusion type formula for the permanent. Indeed, such a formula exists, with each of the terms in the sum being in VP, proving that the permanent is in VNP:

$$Perm(M) = \sum_{T \subseteq [n]} (-1)^{n-|T|} \prod_{i=1}^{n} \sum_{j \in T} M_{ij}$$

Next lecture, we will prove some of the claims listed in this class, and perhaps do some super-linear lower bounds for arithmetic circuits.