PRINCETON UNIVERSITY COS 522: COMPUTATIONAL COMPLEXITY

Lecture 21: Algebraic Computation Models

Lecturer: *Sanjeev Arora* Scribe:*Loukas Georgiadis*

We think of numerical algorithms –root-finding, gaussian elimination etc. —as operating over **R** or **C**, even though the underlying representation of the real or complex numbers involves finite precision. Therefore, it is natural to ask about the complexity of computations over the real numbers or even computations over arbitrary fields. Such an idealized model may not be implementable, but it provides a useful approximation to the asymptotic behavior as computers are allowed to use more and more precision in their computations. However, coming up with a meaningful, well-behaved model is not an easy task, as the following example suggests.

EXAMPLE 1 (PITFALLS AWAITING DESIGNERS OF SUCH MODELS) A real number can encode infinite amount of information. For example, a single real number is enough to encode the answer to every instance of SAT (of any language, in general). Thus, a model that can store any real number with infinite precision may not be realistic. Shamir has shown how to factor any integer n in poly$(\log n)$ time on a computer that can do real arithmetic with arbitrary precision.

The usual way to avoid this pitfall is to restrict the algorithms' ability to access individual bits (e.g., the machine may require more than polynomial time to extract a particular digit from a real number).

In this lecture we define two algebraic computation models: algebraic trees and algebraic Turing Machines. The latter is closely related to the standard Turing Machine model and allows us to define similar complexity classes. Then, we refer to the relation of algebraic and boolean classes. We conclude with a discussion on decidability for the algebraic Turing Machine.

1 Algebraic Computation Trees

Recall the comparison based sorting algorithms; they only allow questions of the type $x_i > x_j$, which is the same as asking whether $x_i - x_j > 0$. The left hand side term of this last inequality is a linear function. We examine the effect of allowing i) the use of any polynomial function and ii) the introduction of new variables together with the ability to ask questions about them. These lead to the following definition:

DEFINITION 1 *An Algebraic Computation Tree is a binary tree where each of the nodes has one of the following types:*

- *Leaf labelled "Accept" or "Reject".*
- *Computation node v labelled with y_v, where $y_v = y_u \circ y_w$ and u, w are either inputs or the labels of ancestor nodes and the operator $\circ$ is in $\{+, -, \times, \div, \sqrt{}\}$ (Note: this is*

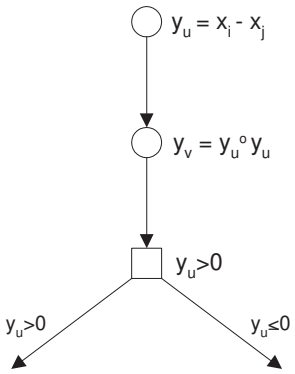

Figure 1: An Algebraic Computation Tree

the standard definition for algebraic trees over fields. We don't allow division in the case of rings).

- *Branch node with out-degree 2. The branch that is taken depends on the evaluation of some condition, such as* $y_u = 0$ *or* $y_u \geq (\leq)0$*, and* y_u *is some designated ancestor.*

The path from the root to an accepting or rejecting leaf depends on the input $x_1, \ldots, x_n \in R$*. The complexity of the computations is measured with respect to the following cost model:*

- $+, -$ *are free.*
- $\times, \div, \sqrt{}$ *are charged unit cost.*

A fragment of an algebraic decision tree is shown in figure 1. The following examples illustrate some of the languages (over real numbers) whose complexity we want to study.

EXAMPLE 2 [Element Distinctness] Given n numbers $x_1, x_2, \ldots, x_n$ we ask whether they are all distinct. This is equivalent to the question whether $\prod_{i \neq j}(x_i - x_j) \neq 0$.

EXAMPLE 3 [Real number version of subset sum] Given a set of n real numbers $X = \{x_1, x_2, \ldots, x_n\}$ we ask whether there is a subset $S \subseteq X$ such that $\sum_{i \in S} x_i = 1$.

REMARK 1 If the depth of a path is d and $y_1, \ldots, y_d$ are variables denoting values of nodes along this path, then the set of inputs $(x_1, \ldots, x_n)$ that follow this path satisfy a set $\mathcal{C}$ of constraints of the type

$$p_i(y_1, \ldots, y_d, x_1, \ldots, x_n) \bowtie 0,$$

where p_i is a degree 2 polynomial (this is an effect of introducing a new variable at each node) and $\bowtie$ is in $\{\leq, \geq, =, \neq\}$. For example, $y_v = y_u \div y_w \leftrightarrow y_v y_w - y_u = 0$. The set S of points that end up in the same path are the solution to an algebraic system (see figure 2).

By applying the following trick we can replace the "$\neq$" constraints from $\mathcal{C}$. Then, S is defined as a semi-algebraic set and we can apply some known results to bound its number of connected components (see Definition 3).

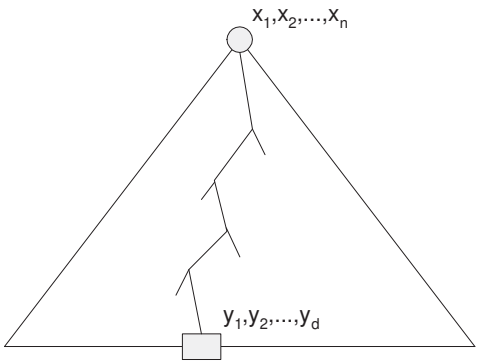

Figure 2: A computation path p of length d defines a set of constraints over the n input variables x_i and d additional variables y_j, which correspond to the nodes on p.

Remark 2 (Rabinovitch's Trick) We can change the "$\neq$" to "$=$" in some constraint

$$p_i(y_1, \ldots, y_m) \neq 0,$$

by introducing a new variable z and asking whether there is a value of z such that

$$q_i(y_1, \ldots, y_m, z) \equiv 1 - zp_i(y_1, \ldots, y_m) = 0$$

(this transformation holds for all fields). Note that we have increased the degree by 1. Alternatively, one can ask whether

$$p_i^2(y_1, \ldots, y_m) > 0,$$

which doubles the degree and does not hold for all fields (e.g., the complex numbers).

Note that after this conversion the maximum degree of the constraints in $\mathcal{C}$ remains 2, because the trick is used only for the branch $y_u \neq 0$ which is converted to $1 - z_v y_u = 0$. (We find Rabinovitch's trick useful also in section 2.2 where we prove a completeness result for Hilbert's Nullstellensatz.)

Definition 2 *A set $S \subseteq \mathbf{R}^n$ is* connected *if for all $x, y \in S$ there is path p that connects x and y and lies entirely in S.*

Definition 3 *For $S \subseteq \mathbf{R}^n$ we define $\#(S)$ to be the number of connected components of S.*

The main idea for proving lowerbounds for the Algebraic Computation Tree model is to show that $\#(S)$ is large. To that end, we will find the following result useful.

Theorem 1 (Simple consequence of Milnor-Thom)
If $S \subseteq \mathbf{R}^n$ is defined by degree d constraints with m equalities and h inequalities then

$$\#(S) \leq d(2d-1)^{n+h-1}$$

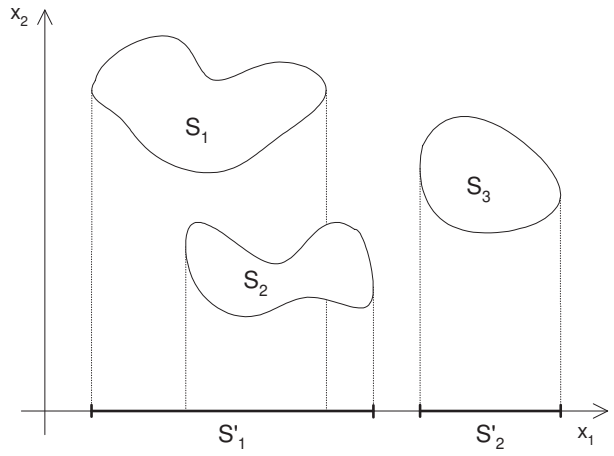

Figure 3: Projection can merge but not add connected components

REMARK 3 Note that the above upperbound is independent of m.

DEFINITION 4 *Let $W \subseteq \mathbf{R}^n$. We define the algebraic decision tree complexity of W as*

$$C(W) = \min_{\substack{\textit{computation} \\ \textit{trees } C \textit{ for } W}} \{\textit{depth of } C\}$$

THEOREM 2

$$C(W) = \Omega\Big(\log\left(\max\{\#(W), \#(\mathbf{R}^n - W)\}\right) - n \Big)$$

PROOF: Suppose that the depth of a computation tree for W is d, so that there are at most 2^d leaves. We will use the fact that if $S \subseteq \mathbf{R}^n$ and $S|_k$ is the set of points in S with their $n-k$ coordinates removed (projection on the first k coordinates) then $\#(S|_k) \leq \#(S)$ (figure 3).

For every leaf there is a set of degree 2 constraints. So, consider a leaf ℓ and the corresponding constraints $\mathcal{C}_\ell$, which are in variables $x_1, \ldots, x_n, y_1, \ldots, y_d$. Let $W_\ell \subseteq \mathbf{R}^n$ be the subset of inputs that reach ℓ and $S_\ell \subseteq \mathbf{R}^{n+d}$ the set of points that satisfy the constraints $\mathcal{C}_\ell$. Note that $W_\ell = \mathcal{C}_\ell|_n$ i.e., W_ℓ is the projection of $\mathcal{C}_\ell$ onto the first n coordinates. So, the number of connected components in W_ℓ is upperbounded by $\#(\mathcal{C}_\ell)$. By Theorem 1 $\#(\mathcal{C}_\ell) \leq 2 \cdot 3^{n+d-1} \leq 3^{n+d}$. Therefore the total number of connected components is at most $2^d 3^{n+d}$, so $d \geq \log(\#(W)) - O(n)$. By repeating the same argument for $\mathbf{R}^n - W$ we have that $d \geq \log(\#(\mathbf{R}^n - W)) - O(n)$. □

Now we can apply the previous result to get a $\Omega(n \log n)$ lowerbound for ELEMENT DISTINCTNESS.

THEOREM 3 (LOWERBOUND FOR ELEMENT DISTINCTNESS)
Let $W = \{(x_1, \ldots, x_n) | \prod_{i \neq j} (x_i - x_j) \neq 0\}$. *Then,*

$$\#(W) \geq n!$$

PROOF: For each permutation σ let

$$W_\sigma = \{(x_1, \ldots, x_n) \mid x_{\sigma(1)} < x_{\sigma(2)} < \ldots < x_{\sigma(n)}\}.$$

That is, let W_σ be the set of n-tuples $(x_1, \ldots, x_n)$ to which σ gives order. We claim that for all $\sigma' \neq \sigma$ the sets W_σ and $W_{\sigma'}$ are not connected. The theorem follows immediately from this claim.

Now consider two distinct permutations σ and σ'. There exist two distinct i, j with $1 \leq i, j \leq n$, such that $\sigma^{-1}(i) < \sigma^{-1}(j)$ but $\sigma^{-1}(i) > \sigma^{-1}(j)$. Thus, in W_σ we have $X_j - X_i > 0$ while in $W_{\sigma'}$ we have $X_i - X_j > 0$. But as we move from the first condition to the second, at some point $X_j - X_i$ must become 0 (by the intermediate value theorem of calculus). Definition 2 then implies that W_σ and $W_{\sigma'}$ cannot be connected. □

2 The Blum-Shub-Smale Model

Here we consider a Turing Machine that computes over some arbitrary field K (e.g. $K = \mathbf{R}, \mathbf{C}, \mathbf{Z}_2$). This is a generalization of the standard Turing Machine model which operates over the ring $\mathbf{Z}_2$. Each cell can hold an element of K with a finite number of cells initially not blank. In the standard model the computation and branch operations can be executed in the same step. Here we perform these operations separately. So we divide the set of states into the following three categories:

- Shift state: move the head to the left or to the right of the current position.
- Branch state: if the content of the current cell is a then goto state q_1 else goto state q_2.
- Computation state: replace the contents of the current cell with a new value. The machine has a hardwired function f and the new contents of the cell become $a \leftarrow f(a)$. In the standard model for rings, f is a polynomial over K, while for fields f is a rational function p/q where p, q are polynomials in $K[x]$ and $q \neq 0$. In either case, f can be represented using a constant number of elements of K.
- The machine has a "register" onto which it can copy the contents of the cell currently under the head. This register's contents can be used in the computation.

In the next section we define some complexity classes related to the BSS model. As usual, the time and space complexity of these Turing Machines is defined with respect to the input size, which is the number of cells occupied by the input.

REMARK 4 The following examples show that some modifications of the BSS model can increase significantly the power of an algebraic Turing Machine.

- If we allow the branch states to check if $a > 0$, for real a then, the model becomes unrealistic because it can decide problems that are undecidable on the normal Turing machine. In particular, such a machine can compute $\mathbf{P}/poly$ in polynomial time. (Recall that we showed that $\mathbf{P}/poly$ contains undecidable languages.) If a language is in $\mathbf{P}/poly$ we can represent its circuit family by a single real number hardwired into the Turing machine (specifically, as the coefficient of of some polynomial $p(x)$ belonging to a state). The individual bits of this coefficient can be accessed by dividing by 2, so the machine can extract the polynomial length encoding of each circuit. Without this ability we can prove that the individual bits cannot be accessed.
- If we allow rounding (computation of $\lfloor x \rfloor$) then it is possible to factor in polynomial time, a result due to Shamir.

Even without these modifications, the BSS model is in some sense more powerful than real-world computers: Consider the execution of the operation $x \leftarrow x^2$ for n times. Since we allow each cell to store a real number, the Turing machine can compute and store in one cell (without overflow) the number x^{2^n} in n steps.

2.1 Complexity Classes over the Complex Numbers

Now we define the corresponding to $\mathbf{P}$ and $\mathbf{NP}$ complexity classes over $\mathbf{C}$:

DEFINITION 5 ($\mathbf{P_C}$,$\mathbf{NP_C}$) *$\mathbf{P_C}$ is the set of languages that can be decided by a Turing Machine over $\mathbf{C}$ in polynomial time. $\mathbf{NP_C}$ is the set of languages L for which there exists a language L_0 in $\mathbf{P_C}$, such that an input x is in L iff there exists a string $(y_1, \ldots, y_{n^c})$ in $\mathbf{C}^{n^c}$ such that (x, y) is in L_0.*

The following definition is a restriction on the inputs of a TM over $\mathbf{C}$. These classes are useful because they help us understand the relation between algebraic and binary complexity classes.

DEFINITION 6 (0-1-$\mathbf{NP_C}$)

$$\text{0-1-}\mathbf{NP_C} = \{L \cap \{0,1\}^* \mid L \in \mathbf{NP_C}\}$$

Note that the input for an $\mathbf{NP_C}$ machine is binary but the nondeterministic "guess" may consist of complex numbers. Trivially, 3SAT is in 0-1-$\mathbf{NP}_C$: even though the "guess" consists of a string of complex numbers, the machine first checks if they are all 0 or 1 using equality checks. Having verified that the guess represents a boolean assignment to the variables, the machine continues as a normal Turing Machine to verify that the assignment satisfies the formula.

It is known that 0-1-$\mathbf{NP_C} \subseteq \mathbf{PSPACE}$. In 1997 Koiran proved that if one assumes the Riemann hypothesis, then 0-1-$\mathbf{NP_C} \subseteq \mathbf{AM}[2]$. Recall that $\mathbf{AM}[2]$ is $\mathbf{BP} \cdot \mathbf{NP}$ so Koiran's result suggests that 0-1-$\mathbf{NP_C}$ may not be much bigger than $\mathbf{NP}$.

2.2 Hilbert's Nullstellensatz

The language $\mathbf{HN_C}$ is defined as the decision version of Hilbert's Nullstellensatz over $\mathbf{C}$. The input consists of m polynomials p_i of degree d over $x_1, \ldots, x_n$. The output is "yes" iff the polynomials have a common root $a_1, \ldots, a_n$. Note that this problem is general enough to include SAT. We illustrate that by the following example:

$$x \vee y \vee z \leftrightarrow (1-x)(1-y)(1-z) = 0.$$

Next we use this fact to prove that the language 0-1-$\mathbf{HN_C}$ (where the polynomials have 0-1 coefficients) is complete for 0-1-$\mathbf{NP_C}$.

THEOREM 4 (BSS)
0-1-$\mathbf{HN_C}$ *is complete for* 0-1-$\mathbf{NP_C}$.

PROOF: (Sketch) It is straightforward to verify that 0-1-$\mathbf{HN_C}$ is in 0-1-$\mathbf{NP_C}$. To prove the hardness part we imitate the proof of the Cook-Levin theorem; we create a computation tableau and show that the verification is in 0-1-$\mathbf{HN_C}$.

To that end, consider the usual computation tableau of a Turing Machine over $\mathbf{C}$ and as in the case of the standard Turing Machines express the fact that the tableau is valid by verifying all the 2×3 windows, i.e., it is sufficient to perform local checks (Figure 4). Reasoning as in the case of algebraic computation trees, we can express these local checks with polynomial constraints of bounded degree. The computation states $c \leftarrow q(a,b)/r(a,b)$ are easily handled by setting $p(c) \equiv q(a,b) - cr(a,b)$. For the branch states $p(a,b) \neq 0$ we can use Rabinovitch's trick to convert them to equality checks $q(a,b,z) = 0$. Thus the degree of our constraints depends upon the degree of the polynomials hardwired into the machine. Also, the polynomial constraints use real coefficients (involving real numbers hardwired into the machine). Converting these polynomial constraints to use only 0 and 1 as coefficients requires work. The idea is to show that the real numbers hardwired into the machine have no effect since the input is a binary string. We omit this mathematical argument here. □

3 Decidability Questions

Since the Blum-Shub-Smale model is more powerful than the ordinary Turing Machine, it is plausible to reconsider decidability questions. In this section we show that some problems do indeed remain undecidable. In particular we study the decidability of the Mandelbrot set with respect to Turing Machines over $\mathbf{C}$. Roger Penrose had raised this question in his meditation regarding artificial intelligence.

3.1 The Mandelbrot Set

DEFINITION 7 (MANDELBROT SET DECISION PROBLEM) *Let* $P_C(Z) = Z^2 + C$. *Then, the Mandelbrot set is defined as*

$$\mathcal{M} = \{C \mid \textit{the sequence } P_C(0), P_C(P_C(0)), P_C(P_C(P_C(0))) \ldots \textit{ is bounded } \}.$$

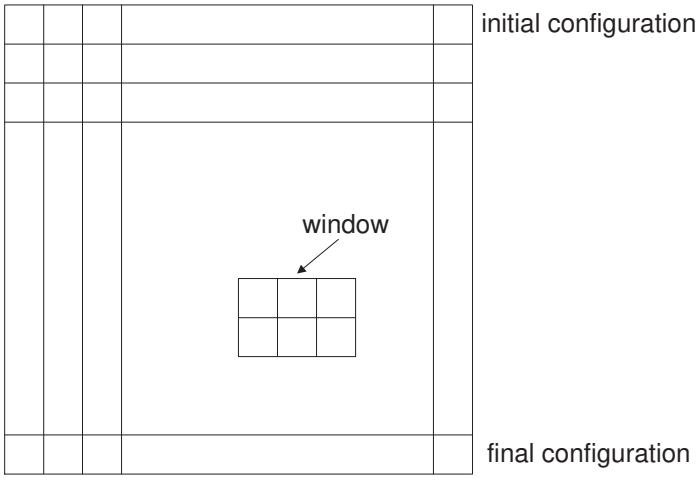

Figure 4: Tableau of Turing Machine configurations

Note that the complement of $\mathcal{M}$ is recognizable if we allow inequality constraints. This is because the sequence is unbounded iff some number $P_C^k(0)$ has complex magnitude greater than 2 for some k (exercise!) and this can be detected in finite time. However, detecting that $P_C^k(0)$ is bounded for every k seems harder. Indeed, we have:

THEOREM 5
$\mathcal{M}$ is undecidable by a machine over $\mathbf{C}$.

PROOF: (Sketch) The proof uses the topology of the Mandelbrot set. Let $\mathcal{M}$ be any TM over the complex numbers that supposedly decides this set. Consider T steps of the computation of this TM. Reasoning as in Theorem 4 and in our theorems about algebraic computation trees, we conclude that the sets of inputs accepted in T steps is is a finite union of semialgebraic sets (i.e., sets defined using solutions to a system of polynomial equations). Hence the language accepted by $\mathcal{M}$ is a countable union of semi-algebraic sets, which implies that its Hausdorft dimension is 1. But it is known Mandelbrot set has Hausdorft dimension 2, hence $\mathcal{M}$ cannot decide it. □