DD3501 Current Research in Proof Complexity Dec 12, 2011

Lecture 9

Lecturer: Jakob Nordström *Scribe: Carl Björkman*

Last lecture, we proved—or at least sketched the proof—that refuting unsatisfiable random k-CNF formulas requires exponential size in polynomial calculus (the proof was via a degree lower bound, and so holds simultaneously for both PC and PCR).

Today we are going to talk about space in polynomial calculus, measured as the number of monomials in the configurations. We will focus on PCR since (as explained before) this eliminates exponential blow-ups in space caused by the encoding of disjunctive clauses and CNF formulas, and since any lower bound on space in PCR also provides a lower bound for PC.

1 State of the Art on Space in PC/PCR and Some Open Questions

The study of space in other proof systems than resolution was initiated by Aleknovich et al. [ABRW02], and they showed (among other things) that the space lower bound

$$Sp_{\mathcal{PCR}}\big(PHP_n^m \vdash \bot\big) \geq n/4 \tag{1.1}$$

holds for pigeonhole principle formulas in PCR. The paper [ABRW02] also raised a number of open problems, but on most of them little or no progress has been made. In particular, [ABRW02] has remained state of the art regarding space in PCR.

Let us therefore begin by stating some of the interesting problems that remain open. Given what we did last time, it is natural to start by asking about the space complexity of random k-CNF formulas.

Open Question 1. *Are random k-CNF formulas hard with respect to space for PCR?*

Any other answer than "yes" for this question would be extremely surprising,[1] but given what is known today we are unable to rule out even the possibility that PC could refute any random k-CNF formula in constant space. Therefore, it would be great to prove any nonconstant lower bound even for PC.

If we look at the pigeonhole principle formulas in (1.1) and set $m = n + 1$ then the size of PHP_n^{n+1} is $\Theta\big(n^3\big)$. Thus, expressed in the formula size N the lower bound in (1.1) is $\Omega(\sqrt[3]{N})$, and for PHP formulas this is tight. The best general *upper* bound that we can prove on PCR space is $\mathrm{O}(N)$, however, so there is a gap between the best known upper and lower bounds. This leads to the following question.

Open Question 2. *Are there (k-)CNF formulas F_n such that $Sp_{\mathcal{PCR}}(F \vdash \bot) = \Omega(S(F_n))$?*

Again, we would expect the answer to this question to be "yes," and an obvious candidate formula family for proving this would be the random k-CNF formulas in Open Question 1.

But there is an even more basic question here. As we have discussed before, we prefer to prove results for k-CNF formulas (where as always it is assumed that $k = \mathrm{O}(1)$). The motivation for this is that any formula can be converted into an equivalent 3-CNF formula of essentially the same size, and we would like to have lower bounds that hold even after such a conversion (otherwise, if turning a formula into 3-CNF would always make it easy, in

[1] Perhaps it should be pointed out in this context that it is known random k-CNF formulas are hard for resolution clause space. In the context of this course, the easiest way to prove this is to appeal to width lower bounds proven in [BW01] (similar to the lower bounds we proved in earlier lectures) plus the lower bound on space in terms of width in [AD08]. The correct citation for the result is the earlier paper [BG03], however, where the lower bound was proven by a direct argument.

practice one could always perform such a preprocessing step). Also, when one considers formula families of unbounded width, one sometimes gets strange results. For example, we have seen for polynomial calculus that CNF formulas with wide clauses containing many negated literals will require exponential monomial space, but converting these formulas to k-CNFs brings the space requirements down to linear. There are other examples of this as well, as discussed, for instance, in [Nor09, Section 5].

However, we have not really known anything regarding lower bounds on space complexity in PC or PCR for k-CNF formulas. Recall that $W(PHP_n^m) = n$, but the space lower bound (1.1) in [ABRW02] is $n/4$ which is much smaller than n. And the proof of (1.1) depends heavily on that we have these wide clauses. As to formulas of bounded width, up to this point nothing has been known, not even for PC (which would seem to be much weaker than PCR with respect to space).

An obvious approach for proving space lower bounds for k-CNF formulas would be to convert the PHP formulas to 3-CNF and try to do some version of the proof in [ABRW02]. There are a number of results in resolution where one gets the same bounds for 3-CNF versions of formulas in this way, although the required modifications in the proofs are sometimes technical and tedious. But this approach just does not seem to work here. The proof presented in [ABRW02] breaks down for $\widetilde{PHP}_n^m$ and it is not clear how to fix it (and of course, the fact that the proof breaks does not proof the opposite statement—it still seems very likely that the same lower bound should hold, but we just do not know how to prove it).

In view of this discussion, we can ask the following somewhat provocative question.

Question 3. *Is it true that PCR (or even PC) can refute* any *unsatisfiable k-CNF formula in constant monomial space?*

For a long time this was an open question and the answer "yes"—although seemingly absurd—was consistent with the state of the art. However, very recently a lower bound was proven for k-CNF formulas by the lecturer in joint work with Filmus, Lauria, Thapen and Zewi [FLN+12].

When this course was being planned, the plan for todays lecture was to talk about the [ABRW02] result. However, given the recent developments we will instead cover the [FLN+12] result instead. For fairness' sake, it should be said that the two results are very similar in spirit. The paper [FLN+12] is very much inspired by [ABRW02] and uses a lot of the same proof techniques. The formulas studied in [FLN+12] are also encodings of the pigeonhole principle, but not the ordinary PHP formula but instead a different variation of it.

The main result in [FLN+12] is as follows.

Theorem 1.1 ([FLN+12]). *There are unsatisfiable 4-CNF formulas of size $\Theta(n)$ which require monomial space $Sp_{\mathcal{PCR}}(F_n \vdash \bot) = \Omega(\sqrt[3]{n})$ in polynomial calculus resolution.*

However, this is not exactly what we will prove today. Instead, we will prove a slightly weaker result, in the hope that this will be somewhat easier to follow. But all the ingredients needed to establish Theorem 1.1 will be there in the result we will prove, which is the following.

Theorem 1.2 ([FLN+12]). *There are unsatisfiable CNF formulas F_n of size $\Theta(n)$ and width $\Theta(\log n)$ such that $Sp_{\mathcal{PCR}}(F_n \vdash \bot) = \Omega(\sqrt[3]{n/\log n})$.*

So the width is not quite constant (that is, these are not k-CNF formulas) but it is a clear improvement over [ABRW02]. Note that in Theorem 1.2, the space lower bound is polynomial while the formula width is logarithmic. Thus, it is no longer the case that we have very large clauses and get a smaller lower bound on space, but rather the space lower bound is exponential in the formula width.

The proofs of Theorems 1.1 and 1.2 are both very much inspired by [ABRW02] and use very similar techniques. However, there are also some extra "twists" needed (as is to be expected—otherwise the problem probably would not have stayed open for so long).

The formulas used in both theorems are encodings of the pigeonhole principle, but not the versions that we are used to seeing in the proof complexity literature. In this lecture, we will focus on the formulas in Theorem 1.2.

2 Different CNF Encodings of the Pigeonhole Principle

Before starting to discuss Theorem 1.2, however, this seems like a natural place to make a short detour and discuss the different "standard encodings" of pigeonhole principles encountered in proof complexity. This is not strictly necessary for what will follow, but is useful background knowledge. There are four such "standard encodings" as described below (and in proving Theorem 1.2 we will use a different, fifth encoding). We begin by giving some informal descriptions of these different encodings.

The formulas that we will refer to as *"basic" PHP*, which are the formulas we have seen before in this course, just encode that every pigeon has at least one hole. Note that in this version a pigeon can occupy several holes, i.e., there can be "fat pigeons". The *functional PHP* formulas specify that every pigeon gets *exactly* one hole. That is, here we can think of functions mapping pigeons to holes. *Onto PHP* formulas are "basic PHP" formulas with the added constrain that all holes should get a pigeon. Finally, *onto functional PHP* formulas are, as the name suggests, simply a combination of functional and onto PHP. These formulas specify that there is a bijective matching between pigeons and holes. This variant is also known as "bijective PHP" or "perfect matching principle."

Let us next describe how to encode these various restrictions. Note that in what follows, we are always assuming $m > n$ (so that the formulas are unsatisfiable). The standard setting would be to have $m = n + 1$, but as mentioned before larger values of m can also be of interest and have been studied in the literature.

Pigeon axioms	$P^i = \bigvee_{j=1}^{n} x_{ij}$	$i \in [m]$	(2.1a)
Hole axioms	$H_j^{ii'} = \overline{x}_{ij} \vee \overline{x}_{i'j}$	$i, i' \in [m]$, $j \in [n]$, $i \neq i'$	(2.1b)
Functional axioms	$F_{jj'}^{i} = \overline{x}_{ij} \vee \overline{x}_{ij'}$	$i \in [m]$, $j, j' \in [n]$, $j \neq j'$	(2.1c)
Onto or surjective axioms	$S_j = \bigvee_{i=1}^{m} x_{ij}$	$j \in [n]$	(2.1d)

With this notation, we can now define the formula families as follows, where the names of the formulas should hopefully be self-explanatory. All indices i, i' run from 1 to m and indices j, j' run from 1 to n.

$$PHP_n^m = \bigwedge_i P^i \wedge \bigwedge_j \bigwedge_{i \neq i'} H_j^{ii'} \tag{2.2a}$$

$$FPHP_n^m = PHP_n^m \wedge \bigwedge_i \bigwedge_{j \neq j'} F_{jj'}^{i} \tag{2.2b}$$

$$\textit{Onto-}PHP_n^m = PHP_n^m \wedge \bigwedge_j S_j \tag{2.2c}$$

$$\textit{Onto-}FPHP_n^m = FPHP_n^m \wedge \bigwedge_j S_j \tag{2.2d}$$

It should be noted that the distinction between the different flavours of the pigeonhole principle formulas is not always made clear in the literature. Some papers just speak of PHP formulas, and one has to look carefully in the definitions to see which variant is being referred to. Also, the reader should be aware that notation for the different versions of PHP is not standardized, although it is probably fair to say that the notation used here is fairly common.

When one want to prove strong lower bounds, considering onto functional PHP formulas gives the strongest results since the other formulas are subsets of these formulas. In the other

direction if one wants to prove a lower bound then showing such a bound for the "basic" PHP formulas gives the strongest results. A very nice survey of what is known about these formulas in different proof systems (and for different values of $m > n$) is [Raz01]. One particularly nice feature of this survey paper is that it makes clear, for every result stated, for which version of PHP it holds, even if the authors of the original paper did not specify this.

3 A Lower Bound on PCR Space for Formulas of Logarithmic Width

To prove Theorem 1.2, we will study the functional pigeonhole principle but in a different encoding than the formulas $FPHP_n^m$ above. After all, if proving lower bounds for the standard encodings of the pigeonhole principle seems hard, why not try a different encoding? The formula family that we will describe next comes from bounded arithmetic (the third reason for studying proof complexity mentioned in lecture 1, and an area that we unfortunately will not be able to cover at all in this course), where such an encoding of the pigeonhole principle occurs naturally.

3.1 Bitwise PHP Formulas

Before we state the definition of the PHP encoding we will use, we set up some notation. The first piece of notation is to encode positive and negative literals over a variable x by exponentiation as follows:

$$x^b = \begin{cases} x & \text{if } b = 0, \\ \overline{x} & \text{if } b = 1. \end{cases} \tag{3.1}$$

We add a word of caution that this notation is not quite standard. In particular, it has been used in some papers on resolution (for instance, [BW01]) defined exactly the other way around with respect to the value of b. In the current context of PCR, however, when we identify 0 with truth, the definition in (3.1) is clearly the right one. The important thing to note here is that for $b \in \{0, 1\}$, we get that the equation $x^b = 0$ is satisfied, i.e., that x^b is true, if and only if $x = b$.

As usual we have the standard notation $[n] = \{1, 2, \ldots, n\}$, but we also define

$$[n, m) = \{n, n + 1, \ldots, m - 1\} \ . \tag{3.2}$$

We can now describe the PHP encoding we will use.

Definition 3.1 (Bitwise PHP formula). Let $n = 2^\ell$. Then the *bitwise pigeonhole principle formula* $BPHP_n^m$ has propositional variables $x[p, i]$ for each $p \in [0, m)$ and $i \in [0, \ell)$. We think of $[0, m)$ as a set of pigeons and $[0, n)$ as a set of holes. Each pigeon p is sent to a hole whose binary encoding is given by the string $x[p, \ell - 1] \cdots x[p, 1]x[p, 0]$. The variables $x[p, i]$ are said to be *associated* with the pigeon p.

The formula $BPHP_n^m$ states that no two pigeons map to the same hole. For every two pigeons $p_1 \neq p_2 \in [0, m)$ and every hole $h \in [0, n)$ we have a *hole axiom*

$$\bigvee_{i=0}^{\ell-1} x[p_1, i]^{1-h_i} \vee \bigvee_{i=0}^{\ell-1} x[p_2, i]^{1-h_i} \tag{3.3}$$

stating that either p_1 is not mapped to h or p_2 is not mapped to h, where $h_{\ell-1} \cdots h_0$ is the binary encoding of h.

Note that the pigeons are indexed from 0 to $n-1$ and the holes are indexed from 0 to $m - 1$. What the clause in (3.3) says is that for at least one of the pigeons p_1 or p_2, the binary expansion of the hole this pigeon is sent to is different from the binary expansion of h, in the sense that there is some bit that has the opposite sign of the hole for either the first or the second pigeon. Note that the width of the clauses is $2\ell = \mathrm{O}(\log n)$.

3.2 Well-Behaved Assignments

Using the formulas in Definition 3.1, we can now restate Theorem 1.2 as follows (where we set $m = n + 1$ to get the size and width as stated in Theorem 1.2).

Theorem 3.2. $Sp_{\mathcal{PCR}}(BPHP_n^m \vdash \bot) > n/8$.

Note that $BPHP_n^m$ is indeed an encoding of the functional pigeonhole principle in that every pigeon is sent to exactly one hole. Also, we observe that any (total) truth value assignment α to $Vars\big(BPHP_n^m\big)$ defines a unique function $f_\alpha : [0, m) \to [0, n)$. All truth value assignments discussed in what follows will be total, and we will use this dual view below and identify the assignment α and the function f_α. Let us continue with some important definitions for the proof.

Definition 3.3 (Well-behaved truth value assignment). A total truth value assignment α over $Vars\,(BPHP_n^m)$ is *well-behaved* over a set of pigeons $S \subseteq [0, m)$ if it sends these pigeons to distinct holes.

Note that there are no well-behaved assignment over the full set of pigeons $[0, m)$, since this would mean that each of the m pigeons gets its own private hole among the $n < m$ holes to choose from. However, for any $S \subsetneqq [0, m)$ of size $|S| \leq n$ there are well-behaved assignments.

3.3 The General Proof Strategy

Now let us outline the proof strategy which is actually a sort of generic strategy for proving lower bounds. Given a formula F and a space bound s we want to prove $Sp_{\mathcal{PC}}(F \vdash \bot) \geq s$. Equivalently, we can study any derivation $\pi = \{\mathbb{P}_0 = \emptyset, \ldots, \mathbb{P}_\tau\}$ such that $Sp(\pi) < s$ and show that π does not derive contradiction (i.e., that $1 \notin \mathbb{P}_\tau$).

A natural way to go about this is to consider any derivation in space less than s and prove for each $\mathbb{P}_i$ in this derivation that it is satisfiable. But how do we do this? We do not know much about $\mathbb{P}_i$, other than that the configuration has been derived in small space. But it seems very hard to say something about the structure of the configuration. This is true already for resolution, where we only have disjunctive clauses as lines in the derivations and only use the simple resolution rule. The task seems even harder for PC or PCR, where a priori any (multilinear) polynomial can appear as a proof line and polynomials can be added and multiplied, creating a rich and complex structure. There is just no way we can get a handle on such a configuration directly (at least this lecturer has no idea how to do it). Instead, we will use a different approach.

The idea we will use is that we study a configuration $\mathbb{P}_i$ by constructing an *auxiliary configuration* $\mathcal{A}_i$ that is easier to understand than $\mathbb{P}_i$ but at the same time gives us information about $\mathbb{P}_i$. We want the following properties must hold for our auxiliary configuration $\mathcal{A}_i$:

1. $\mathcal{A}_i$ implies $\mathbb{P}_i$ (i.e., $\mathcal{A}_i$ is "stronger" than $\mathbb{P}_i$).
2. $\mathcal{A}_i$ is satisfiable. (Which, in particular, implies that $\mathbb{P}_i$ will also be satisfiable, which is exactly what we want to prove.)
3. Whenever we do a derivation step $\mathbb{P}_i \rightsquigarrow \mathbb{P}_{i+1}$, we can do a *local update* of $\mathcal{A}_i$ to $\mathcal{A}_{i+1}$ if $Sp(\mathbb{P}_i)$ is small enough (that is, less than s).

It should be clear that if we can do this, we also immediately get a lower bound on space.

If we describe this setup a little bit more intuitively and informally, we can think of it as a game between us and an adversary, the prover. The prover performs a PCR derivation, and strange and wonderful things are going on there that we do not quite understand. However, for every configuration $\mathbb{P}_i$ in this PCR derivation we come up with a simpler configuration $\mathcal{A}_i$ that

we do understand. It is important to understand that $\mathcal{A}_i$ may (and will) look very different from $\mathbb{P}_i$, but the crucial aspect here is that we stay "on the right side" of $\mathbb{P}_i$ by making sure that our configuration $\mathcal{A}_i$ is always stronger. And if we can make sure that all configurations $\mathcal{A}_i$ are satisfiable as long as the space does not exceed s, this means that the adversary can never derive contradiction in space less than s.

This is the method used to prove space lower bounds in [ABRW02], where it was used for both resolution and PCR.

3.4 Commitments

The challenge here will clearly be to get the concrete details right for the auxiliary configurations so that it works and all properties hold. In our implementation of this program, we will consider implications with respect to well-behaved assignments (Definition 3.3). We will think about the auxiliary configurations as "commitments" to put certain pigeons in certain subsets (in particular: halves) of the holes by fixing one bit of the pigeon assignment. Let us state the formal definition of this.

Definition 3.4 (Commitment). A *disjunctive commitment*, or just *commitment* for brevity, is a 2-clause of the form $x[p_1, i_1]^{b_1} \vee x[p_2, i_2]^{b_2}$, where $p_1 \neq p_2$. The *domain* of a commitment C, written $\mathrm{dom}(C)$, is the set of pigeons mentioned in C.

A *commitment set* $\mathcal{A}$ is a set of commitments $\{C_1, C_2, \ldots, C_s\}$ such that for all $i \neq j$ it holds that $\mathrm{dom}(C_i) \cap \mathrm{dom}(C_j) = \emptyset$ (that is, all the pigeons mentioned in the set are distinct). The domain of a commitment set $\mathcal{A}$, written $\mathrm{dom}(\mathcal{A})$, is $\mathrm{dom}(\mathcal{A}) = \bigcup_{C \in \mathcal{A}} \mathrm{dom}(C)$. The *size* of a commitment set $\mathcal{A}$, denoted $|\mathcal{A}|$, is the number of commitments in $\mathcal{A}$.

An assignment[2] α is *well-behaved on and satisfies* a commitment set $\mathcal{A}$ if α is well-behaved on $\mathrm{dom}(\mathcal{A})$ and satisfies $\mathcal{A}$.

We want to look at any PCR derivation π in small space and for any $\mathbb{P}_i$ in π we want to construct a commitment set $\mathcal{A}_i$ such that any α that is well-behaved on $\mathcal{A}_i$ and satisfies $\mathcal{A}_i$ must also satisfy $\mathbb{P}_i$. If we can do this we are done.

3.5 Entailment and the Locality Lemma

We will not be able to carry out the whole proof of Theorem 1.2 today, but we conclude this lecture by stating the key lemmas used in the proof, and proving all of them except the most important one, and then sketching how the proof of the theorem goes.

Lemma 3.5. *Given any set $S \subseteq [0, m), |S| < n/2$, any assignment α that is well-behaved on S, and any literal $x[p, i]^b$ associated with a pigeon $p \notin S$, we can modify α to β by reassigning variables associated to the pigeon p so that β is well-behaved on $S \cup \{p\}$ and satisfies $x[p, i]^b$.*

Proof. Exactly half of the n holes has a binary expansion where the ith bit is b. The pigeons in S use less than $n/2$ holes. Hence there must exist a hole h that is not assigned to any pigeon in S and has the right value of the ith bit, so that sending p to h satisfies $x[p, i]^b$. □

From Lemma 3.5 we immediately get the following important corollary.

Corollary 3.6. *Let $S, T \subset [0, m)$ be sets of pigeons such that $S \cap T = \emptyset$ and $|S \cup T| \leq n/2$, and let X be a set containing exactly one literal $x[p, i_p]^{b_p}$ for every $p \in T$. Then any assigned α that is well-behaved on S can be modified to an assignment β by reassigning pigeons in T so that β is well-behaved on $S \cup T$ and satisfies all literals in X.*

Proof. Consider pigeons in T one by one and apply Lemma 3.5. □

[2] Recall that all truth value assignments discussed here are total.

Let us now define what it means that a commitment set implies, or *entails* a set of multilinear polynomials.

Definition 3.7 (Entailment). Let $\mathcal{A}$ be a commitment set and $\mathbb{P}$ be a PCR-configuration. Then $\mathcal{A}$ *entails* $\mathbb{P}$ *over well-behaved assignments* if every assignment α which is well-behaved on and satisfies $\mathcal{A}$ must also satisfy $\mathbb{P}$. That is, for every polynomial $P \in \mathbb{P}$ we have that $P(\alpha) = 0$.

Now we are ready to state the main technical lemma.

Lemma 3.8 (Locality lemma[3]). *Suppose that $\mathcal{A}$ is a commitment set, $|\mathcal{A}| \leq n/4$, and that $\mathbb{P}$ is a PCR-configuration such that $\mathcal{A}$ entails $\mathbb{P}$ over well-behaved assignments. Then there is a commitment set $\mathcal{B}$ of size $|\mathcal{B}| \leq 2 \cdot Sp(\mathbb{P})$ such that $\mathcal{B}$ entails $\mathbb{P}$ over well-behaved assignments.*

As it turns out, Lemma 3.8 is exactly what we need to prove Theorem 1.2. This lemma is an absolutely crucial component in the proof, and we remark that it is very similar to a lemma in [ABRW02] (also called a "locality lemma" in that paper, so that is from where the name comes). Also, our proof of Lemma 3.8, which we will present in the next lecture,[4] is very similar to that in [ABRW02] However we need to do things a bit differently to make the proof work for the *BPHP* formulas (and for the 4-CNF formulas in Theorem 1.1 which we do not discuss here).

3.6 Brief Proof Sketch for Theorem 1.2

Let us now finally try to outline a proof of Theorem 1.2. Given any PCR-derivation $\pi = \{\mathbb{P}_0 = \emptyset, \mathbb{P}_1, \ldots, \mathbb{P}_\tau\}$ from $BPHP_n^m$ in space at most $n/8$, we want to construct a sequence of commitment sets $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_\tau$ such that for each time step t it holds that $|\mathcal{A}_t| \leq 2 \cdot Sp(\mathbb{P}_t)$ and $\mathcal{A}_t$ entails $\mathbb{P}_t$ over well-behaved assignments.

Clearly we may define $\mathcal{A}_0$ to be the empty commitment. Supposing that we have already defined $\mathcal{A}_t$, we want to construct $\mathcal{A}_{t+1}$ by induction. We get three cases, depending on which step is used to obtain $\mathbb{P}_{t+1}$ from $\mathbb{P}_t$. We sketch how the proof goes, and will return to the details in next lecture.

Download There are two kinds of axioms: logical axioms ($x^2 - x = 0$ and $x + \overline{x} - 1 = 0$) and hole axioms. Logical axioms can be ignored since they are always satisfied by definition by any truth value assignment. For a hole axiom $\bigvee_{i=0}^{\ell-1} x[p_1, i]^{1-h_i} \vee \bigvee_{i=0}^{\ell-1} x[p_2, i]^{1-h_i}$, we can essentially just pick any one literal for p_1 and any one literal for p_2 and let the disjunction of these two literals be the new commitment which we add to $\mathcal{A}_{t+1}$. (Note that this illustrates the fact that we are "on the right side" of our adversary the prover, since this commitment subclause of size 2 is certainly stronger than, and hence implies, the full hole axiom.)

Actually, the approach described above does not quite work as statedy (exercise for the reader: why?), but we will return to the details and get them right next time.

Inference For inference steps we do not need to do anything. Since the derivation rules in PCR are sound, any polynomial derived from $\mathbb{P}_t$ is also implied by $\mathbb{P}_t$, which means that $\mathcal{A}_t$ works fine as a commitment set also for $\mathbb{P}_{t+1}$.

Erasure This is the tricky case. Suppose $\mathbb{P}_{t+1} \subsetneqq \mathbb{P}_t$. The commitment set $\mathcal{A}_t$ implies $\mathbb{P}_{t+1}$ but might be far too large. Maybe the prover managed to sum two polynomials P and Q in such a way that a lot of monomials cancelled and then erased both P and Q, which means that $\mathbb{P}_{t+1}$ contains a set of very few monomials having very strong implications!

[3] Also known as the "Magic lemma" among the authors of [FLN+12].

[4] That is, the lecture after the guest lecture on SAT solving.

For instance, if the prover has derived a polynomial of the form $uvwxyz - 1$, then this fixes the truth value of 6 variables although the monomial space is just 2 (and it is not hard to see that 6 can be increased to any arbitrarily large number by just letting the first monomial have degree going off to infinity).

However, the fact that Lemma 3.8 holds tells us, as if by magic, that such polynomials cannot be derived in small space. For if they could, there would be no commitment set $\mathcal{A}$ that would imply them (this fact is left as a not too hard, but potentially quite useful exercise). Just by applying Lemma 3.8, we get that since $|\mathcal{A}_t| \leq 2 \cdot Sp(\mathbb{P}_t) \leq n/4$, there exists a commitment set $\mathcal{A}_{t+1}$ such that $|\mathcal{A}_{t+1}| \leq 2 \cdot Sp(\mathbb{P}_{t+1})$ and $\mathcal{A}_{t+1}$ entails $\mathbb{P}_{t+1}$ over well-behaved assignments. And this is precisely what we need to make the induction step go through.

As already stated above, we will return to Theorem 1.2 in the next lecture and provide a full proof. But by now it should hopefully be clear that the whole argument hinges on Lemma 3.8, and indeed most of the the next and final regular lecture of the autumn term (again, the lecture after the guest lecture on SAT solving) will be spent on proving this lemma.

References

[ABRW02] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4):1184–1211, 2002. Preliminary version appeared in *STOC '00*.

[AD08] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, May 2008. Preliminary version appeared in *CCC '03*.

[BG03] Eli Ben-Sasson and Nicola Galesi. Space complexity of random formulae in resolution. *Random Structures and Algorithms*, 23(1):92–109, August 2003. Preliminary version appeared in *CCC '01*.

[BW01] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version appeared in *STOC '99*.

[FLN+12] Yuval Filmus, Massimo Lauria, Jakob Nordström, Neil Thapen, and Noga Zewi. Space complexity in polynomial calculus. In *Proceedings of the 27th Annual IEEE Conference on Computational Complexity (CCC '12)*, June 2012. To appear.

[Nor09] Jakob Nordström. A simplified way of proving trade-off results for resolution. *Information Processing Letters*, 109(18):1030–1035, August 2009. Preliminary version appeared in ECCC report TR07-114, 2007.

[Raz01] Alexander A. Razborov. Proof complexity of pigeonhole principles. In *5th International Conference on Developments in Language Theory, (DLT '01), Revised Papers*, volume 2295 of *Lecture Notes in Computer Science*, pages 100–116. Springer, July 2001.