

1. Let us consider the following decision problem::

$$Q(x, y) := \text{“is there a proof for } x \in L \text{ having } y \text{ as a prefix?”}$$

We know that verifying that  $p$  is a proof for  $x \in L$  is in  $P$ . Thus deciding if there is a proof starting with  $y$  is in  $NP$  (we can “guess” the rest of the solution, and then verify it). But, as we assumed that  $P = NP$ , we can answer this question deterministically in polynomial time. Thus we can construct the algorithm for the function  $f$ :

- if not  $Q(x, \epsilon)$ , return “I’m very sorry, no luck this time”
- $p := \epsilon$
- while not  $V(x, p)$  do
  - if  $Q(x, p0)$  then  $p := p0$  else  $p := p1$

The algorithm tries out consecutive bits of the proof and uses the decider for  $Q$  to check if it is following the right path. As a single check can be done in polynomial time and the length of the proof is polynomial in the length of  $x$ , our function requires only polynomial time to compute.

2. Suppose that there exists a polybounded automatizable PPS. Given any  $\phi$  in TAUT we know, that it has a proof of polynomial size. It means that we can find a proof of  $\phi$  in time polynomial of length of  $\phi$ . As TAUT is  $co - NP$ -complete, we get that  $P = co - NP$  and, what follows,  $P = NP$ .

Let us now assume that  $P = NP$ . Then SAT can be decided by some deterministic polytime Turing machine  $M$ . Take any formula  $\phi$  and notice, that it is a tautology if and only if  $\neg\phi$  is not satisfiable. Consider the computation of  $M$  on  $\neg\phi$ .  $M$  finishes its work in polynomial time, thus it can use only polynomial space. Therefore the “encoding” of the computation has polynomial size. But this “encoding” is a great proof that  $\phi \in \text{TAUT}$  (we can verify it in polynomial time, simply checking that it is a correct computation of  $M$  on  $\phi$ ), and it can be produced in polynomial time (as  $M$  produces it). Thus we have found a polybounded automatizable PPS.

3. (a) Let  $\mathcal{C}(A)$  be a set of clauses with the property that  $\mathcal{C}(A) \cup \{q_A\}$  is satisfiable iff  $A$  is satisfiable. We define  $\mathcal{C}(A)$  by structural induction on  $A$ : if  $A := x$ , then  $\mathcal{C}(A) := \{\{q_A, \bar{x}\}, \{\bar{q}_A, x\}\}$ . If  $A := \neg B$ , then  $\mathcal{C}(A) := \{\{q_A, q_B\}, \{\bar{q}_A, \bar{q}_B\}\} \cup \mathcal{C}(B)$ , and if  $A := B_1 \vee B_2$ , then  $\mathcal{C}(A) := \{\{\bar{q}_A, q_{B_1}, q_{B_2}\}, \{q_A, \bar{q}_{B_1}\}, \{q_A, \bar{q}_{B_2}\}\} \cup \mathcal{C}(B_1) \cup \mathcal{C}(B_2)$ . Similarly for conjunction. Also need to show inductively that the correctness condition holds.
- (b) If we could translate an arbitrary boolean formula into equivalent CNF *without* adding new variables and with at most a polynomial increase in size, we would effectively have a polybounded proof system (and  $NP = co - NP$ ). The reason is that a formula in CNF is a tautology iff each clause contains some variable  $x$  and its negation; this can be checked in linear time in the length of the formula.

4. Let us consider an  $NC^1$  family of circuits. We can represent each circuit by a Boolean formula. We need to repeat any sub-formulae representing sub-circuits that have been used multiple times. But as the depth of the circuit was bounded by  $O(\log n)$ , and each gate had at most 2 inputs, the total size of the resulting formula will be bounded by  $2^{O(\log n)} = n^{O(1)}$ . This means that  $NC^1 \subseteq BF$ . I will now prove by induction that any Boolean formula having no more than  $n$  logical connectives ( $\wedge, \vee, \neg$ ) can be represented by an  $NC$  circuit of depth not greater than  $4 \log(n+1)$ . For  $n = 0$  (a single variable) it is obviously true. Let us now fix any Boolean function  $f$  with  $n$  connectives. The tree representing this formula has  $n$  internal nodes. Let us denote by  $s(T)$  the size (number of nodes) of subtree  $T$ . Consider the subtree  $T$  of smallest size larger than  $n/3$ . It must be that  $n/3 < s(T) \leq 2n/3$  (otherwise one of the subtrees of  $T$  would be a better candidate). Let us now look at the formula  $f$  built from sub-formulae  $f_1, \dots, f_k$  such that  $f_i$  is the sub-formula represented by  $T$ :

$$f(x_1, \dots, x_m) = f(f_1(x_1, \dots, x_m), \dots, f_i(x_1, \dots, x_m), \dots, f_k(x_1, \dots, x_m))$$

and note that

$$f(f_1, \dots, f_i, \dots, f_k) = (f_i \wedge f(f_1, \dots, 1, \dots, f_k)) \vee (\neg f_i \wedge f(f_1, \dots, 0, \dots, f_k))$$

The formula  $f_i$  has size less than  $2n/3$ . When we replace  $f_i$  in  $f$  by 0 or 1, we will get a formula of size bounded by  $2n/3$  (because the size of  $f_i$  was at least  $n/3$ ). Both these formulae can be represented (according to inductive assumption) by circuits of depth bounded by

$$4 \log(2n/3 + 1) \leq 4(\log(2/3) + \log(n+1)) \leq 4 \log(n+1) - 2$$

Then we can represent our original formula  $f$  by circuit of depth bounded by

$$4 \log(n+1) - 2 + 2 = 4 \log(n+1)$$

Our circuit has logarithmic depth and constant fan-in, so its size will be polynomial. We can create an appropriate circuit for every function in the given family. Thus we get that  $BF \subseteq NC^1$ .

5. To create a clausal representation of  $v_{\text{new}} = \alpha$ , it is enough to transform the sentence

$$(v_{\text{new}} \wedge \alpha) \vee (\neg v_{\text{new}} \wedge \neg \alpha)$$

into CNF (possibly introducing more new variables).

Following is the (simplified) refutation of  $PHP_n^{n+1}$  in  $ER$ :

- introduce variables  $B_{i,j}^n$ :

$$B_{i,j}^n := P_{i,j} \text{ for } i = 1, \dots, n+1 \text{ and } j = 1, \dots, n$$

- for  $k = n - 1, \dots, 1$  do:
  - introduce variables  $B_{i,j}^k$ :

$$B_{i,j}^k := B_{i,j}^{k+1} \vee (B_{i,k+1}^{k+1} \wedge B_{k+2,j}^{k+1}) \text{ for } 1 \leq i \leq k+1, 1 \leq j \leq k$$

- from clauses

$$\{\{B_{i,1}^{k+1}, \dots, B_{i,k+1}^{k+1}\}, \{\neg B_{i,l}^{k+1}, \neg B_{j,l}^{k+1}\}\}$$

deduce

$$\{\{B_{i,1}^k, \dots, B_{i,k}^k\}, \{\neg B_{i,l}^k, \neg B_{j,l}^k\}\}$$

- refute the resulting set of clauses

$$\{\{B_{1,1}^1\}, \{B_{2,1}^1\}, \{\neg B_{1,1}^1, \neg B_{2,1}^1\}\}$$

The basic idea is to encode the  $PHP_{n-1}^n$  using new variables and prove it from  $PHP_n^{n+1}$ . Then, repeating the process go to  $PHP_1^2$ , which is easy to refute. Each step of induction requires polynomially many formulae, so the whole refutation is polynomial in size.