Lecture 1— Basics of Proof systems, and Resolution

Massimo Lauria — lauria.massimo@gmail.com
Office 1107, Ookayama West 8th Building

Tuesday — October 20th, 2015 (This document was updated on June 21, 2017)

We introduce the concept of proof system, and of proof complexity, which is the study of propositional proofs. We motivate this study with connections with computational complexity and with SAT algorithms. Then we introduce resolution, the most important proof system in propositional logic. We setup the proof of the fact that the pigeohole principle requires large refutations in resolution.

The course deals with proof of propositional tautologies, or alternatively with refutations of CNF formulas (Conjunctive Normal Form). CNF formulas are obviously falsifiable but it is NP-complete to decide whether they are satisfiable (i.e. to decide whether there exists an assignment that satisfies all the clauses). This means that there is no **known** efficient algorithm for it, and indeed many people think that such algorithm does not exists at all. Unfortunately proving such a claim is one of the most important and hard problem in contemporary mathematics.

We limit ourself to propositional formulas in form of a CNF. That is not too restrictive, since from any propositional formula Ψ it is possible to get a CNF formula ϕ that is unsatisfiable if and only if Ψ is a tautology (i.e. a true formula in propositional logic). When we refer to a CNF we will often use the words "proof" as a synonymous of "refutation", with the intended meaning that the proof of a CNF is actually its refutation. Furthermore we recall that deciding whether a CNF is unsatisfiable is a coNP-complete problem, and it is essentially equivalent to proving propositional theorems.

Exercise 1. Prove that for any propositional formula $\Psi(\vec{x})$ over connectives $\{\neg, \vee, \wedge, \to\}$ there exists a CNF $\phi(\vec{x}, \vec{y})$ such that Ψ is a tautology if and only if ϕ is unsatisfiable, and ϕ has length at most linear in the length of Ψ.

Let UNSAT denote the set of all unsatisfiable CNFs, a *proof system* is a mechanism to certify (or "witness") that a certain input ϕ is indeed in UNSAT. The standard definition of a proof system is due to Cook et al.[1].

[1] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979

Definition 2 (Classic definition)**.** *A proof system is a surjective polynomial time computable function*

$$P : \{0,1\}^* \longrightarrow \text{UNSAT},$$

and a refutation of $\phi \in$ UNSAT is a string $\pi \in \{0,1\}^*$ so that $P(\pi) = \phi$.

The classic definition is very neat, but an alternative definition is closer in spirit to the process of proof verification.

Definition 3 (Verifier definition)**.** *A proof system is a polynomial time Turing machine $M(\cdot,\cdot)$ so that*

$\phi \in$ UNSAT *there is some $\pi \in \{0,1\}^*$ so that $M(\phi, \pi)$ accepts;*
$\phi \notin$ UNSAT *for any $\pi \in \{0,1\}^*$ $M(\phi, \pi)$ rejects.*

If $M(\phi, \pi)$ accepts then π is a refutation of ϕ.

The definition of proof system in these lectures has been given just for the language UNSAT, but it is easy to see that you can define proof systems for any language.

The length of proofs and computational complexity

Definition 3 is almost the same as the definition of the complexity class NP. The **essential** difference is that in the latter the length of the proof is **required** to be polynomial in the length of the formula. The length of proofs is the principal complexity measure studied in proof complexity. If a statement requires a too long proof then for all practical purposes it has no proof at all. Gödel himself had thought about this issue[2].

[2] Kurt Gödel. A letter to John von Neumann, March 20, 1956. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 7–9. Oxford University Press, 1993

Definition 4. *A proof system is called* polynomially bounded *if all $\phi \in$ UNSAT have at least one refutation of polynomial size with respect to length of ϕ.*

Fact 5. *A polynomially bounded proof system for* UNSAT *exists if and only if* $\mathsf{NP} = \mathsf{coNP}$. *In particular if no polynomially bounded proof system for* UNSAT *exists, then* $\mathsf{P} \neq \mathsf{NP}$.

Exercise 6. Prove Fact 5.

The previous fact is indeed the *historical reason* of studying the length of proofs. The idea at the time was to prove the existence of formulas with no short proofs, for increasingly strong proof systems. The hope was that at some point it could have been possible to find a formula with no short proof in **any** proof system. Compare this with the way circuit classes were studied in computational complexity. Lower bounds on the size of circuits were presented as partial steps toward the final goal of separating P from NP.

The goal of proof complexity.

> The main theoretical goal of proof complexity is to prove **unconditional** lower bounds for the refutation length of specific families of CNFs.

Simulations and relative strength

The strength of two proof systems can be compared using the length of proof as measure.

Definition 7 (Simulation and p-simulation)**.** *Consider two proof system M and N. We say that M* simulates *N if there exists $c > 0$ such that for every $\phi \in$ UNSAT and π where $N(\phi, \pi)$ accepts, there is another string π' of length at most $|\pi|^c$ such that $M(\phi, \pi')$ accepts.*

We say that M p-simulates *N if there exists a polynomial time algorithm A such that for every $\phi \in$ UNSAT and π where $N(\phi, \pi)$ accepts, then $M(\phi, A(\pi))$ accepts.*

The concept of simulation is useful to classify proof systems. An immediate consequence of the notion of simulation is that when M simulates N, any proof length lower bound for M holds for N as well, and that any short proof in N can be transformed into a short proof in M. In general we say that a proof system is *stronger* than another one if it can simulate it (or even better p-simulate it).

Proof complexity and SAT solvers

Determining satisfiability seems a purely theoretical problem, nevertheless it is expressive enough to capture many problems coming from industrial and scientific applications as

- artificial intelligence;
- software static analysis;
- cryptography;
- hardware verification;
- and many more...

The reason for such a boost in applications is that since the end of the '90s the algorithms for practical SAT have improved dramatically[3]. Now it is routine to solve formulas with up to 100000 or 1000000 variables, which was absolutely impossible earlier.

[3] João P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *Computers, IEEE Transactions on*, 48(5):506–521, 1999; and M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001

> If a SAT solver claims that a CNF is unsatisfiable, then the trace of its execution is a refutation of the CNF in the sense of Definition 3. This also means that a lower bound for proof length gives a lower bound of the running time of the SAT solver.

When run on unsatisfiable formulas, many SAT solvers (or at least their main components) produce traces which are, up to minor translation, refutations expressible in relatively weak proof systems for which we know how to prove lower bounds. That means that we know formulas that are hard for these SAT solvers.

> SAT solvers look for proofs in weak proof systems. Since such proofs may be very long, why don't we use solvers that correspond to stronger systems?

This is a reasonable question. Unfortunately using a stronger proof system could make even more difficult to find a short proof, since usually the proof language has more flexibility and more degrees of freedom, and the search space is much larger. This is the so called problem of *automatization* of a proof system.

Definition 8 (Proof search and automatization)**.** *A proof search algorithm A_P for a proof system P is an algorithm that gets a CNF formula ϕ in input and*

$\phi \in$ UNSAT *outputs π such that $P(\phi, \pi)$ accepts;*

$\phi \notin$ UNSAT *reports that ϕ is satisfiable.*

A proof system P is automatizable *(resp. quasi-automatizable) if it has a proof search algorithm A_P such that for every $\phi \in$ UNSAT algorithm $A_P(\phi)$ runs in time at most polynomial (resp. quasi-polynomial) with respect to the length of the shortest proof of ϕ.*

Resolution proof system

Resolution[4] is definitely the most important proof system in literature. It is for sure the most studied by theoretician and by people developing SAT solvers.

To introduce resolution we need some definitions. A *literal* over a Boolean variable x is either the variable x itself (a *positive literal*) or its negation that is denoted either as $\bar{x}$ or $\neg x$ (a *negative literal*).

A *clause* $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals, and k is the *width* of clause C.

A *k-clause* is a clause with k literals. A *CNF formula* $\phi = C_1 \wedge \cdots \wedge C_m$ is a conjunction of clauses. If all clauses in a CNF have at most k literals each, then we call it a *k-CNF*. We denote the logical true value as $\top$ and the logical false value as $\bot$. Sometime we encode truth as 1 and false as 0, too. [5] The clause with no literals is the *empty clause*, it is always false and is denoted either as $\square$ or $\bot$. The empty CNF formula is the CNF with no clauses and it is always true.

[4]

[5] While this is a standard encoding, we may see other ways to encode booleans. In algebraic proof systems it is convenient to use 0 for true and 1 for false. In the so called *Fourier encoding* it is convenient to use -1 for true and 1 for false.

Definition 9 (Resolution proof system)**.** *A* resolution derivation *of a clause C from a CNF formula ϕ is a sequence of clauses $\pi = (C_1, \ldots, C_\ell)$ such that $C_\ell = C$ and for $1 \leq i \leq \ell$ the clause C_i is obtained by one of the following inference rules:*

- ***Axiom:*** *C_i is a clause in ϕ (an* axiom clause*);*
- ***Resolution:*** *$C_i = A \vee B$, where $C_j = A \vee x$ and $C_{j'} = B \vee \bar{x}$ for $1 \leq j, j' < i$;*
- ***Weakening:*** *$C_j \subseteq C_i$ for some $1 \leq j < i$.*

A resolution derivation of $\bot$ from ϕ is called a resolution refutation *of ϕ. The* width *of a resolution proof π is the maximal number of literals among clauses in any clause C_i in π, and the* size *(or* length*) of $\pi = (C_1, \ldots, C_\ell)$ is ℓ.*

Exercise 10 (Correctness)**.** Prove that if a CNF formula ϕ has a refutation, then it is unsatisfiable.

Exercise 11. Prove that any resolution derivation of C from ϕ that uses weakening rule can be transformed into a resolution derivation of some $C' \subseteq C$ that does not use the weakening rule and has width and length no larger than the original derivation. In particular this shows that the weakening rule is not necessary for resolution refutations.

Every resolution derivation $\pi = (C_1, \ldots, C_\ell)$ is essentially directed acyclic graph (DAG) with vertices labelled by clauses C_i in π and edges (C_j, C_i) if C_i is obtained by a resolution or a weakening step and C_j is used as a premise in that step. The derivation π is said to be *tree-like* if that directed graph is a rooted tree, oriented from leaves to the root.

Definition 12. *A decision tree for a CNF ϕ is a rooted binary tree where each internal node is labeled by a variable of ϕ and has two outgoing edges labeled by* 0 *and* 1*, respectively, and where no variable label occurs twice on*

any root-to-leaf path. For any node q in the tree we associate an assignment ρ_q as follows: if q is the root then $\rho_q = \emptyset$; if q has parent p then $\rho_q = \rho_p \cup \{x = b\}$ where x is the variable labeling node p and $b \in \{0,1\}$ is the value associated to the edge connecting q to p. The label on a leaf node q depends on the partial assignment corresponding to that node. It is either

- *the value $\top$ when ρ_q satisfies formula ϕ; or*
- *some clause of ϕ falsified by ρ_q otherwise.*

Exercise 13. An unsatisfiable CNF formula ϕ has a decision tree with at most S nodes if and only if it has a tree-like resolution refutation of length at most S.

Exercise 14 (Completeness)**.** Prove that any unsatisfiable ϕ has a resolution refutation.

How to find a resolution refutation? It is possible to find the shortest one or some other refutation not too much longer? Finding the shortest refutation is NP-hard[6], and even finding one at most polynomially larger is difficult, unless the very plausible complexity hypothesis FPT $\neq$ W[P] fails[7].

[6] Kazuo Iwama. Complexity of finding short resolution proofs. In Igor Prívara and Peter Ruzicka, editors, *MFCS*, volume 1295 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 1997

[7] Michael Alekhnovich and Alexander A. Razborov. Resolution is not automatizable unless W[P] is tractable. *SIAM J. Comput.*, 38(4):1347–1363, 2008; and K. Eickmeyer, Martin Grohe, and Magdalena Grüber. Approximation of natural w [p]-complete minimisation problems is hard. In *23rd Annual IEEE Conference on Computational Complexity*, pages 8–18. IEEE, 2008

Next lecture

In the next lecture we will show that a $2^{\Omega(n)}$ lower bounds for the CNF formulation of the pigeonhole principle with $n+1$ pigeons and n holes. The formula is

$$\bigvee_{j\in[n]} p_{i,j} \qquad \text{for every } i \in [n+1]; \tag{1}$$

$$\bar{p}_{i,j} \vee \bar{p}_{i',j} \qquad \text{for every distinct } i, i' \in [n+1] \text{ and } j \in [n]; \tag{2}$$

where the first clauses claim that all pigeons must have a hole (*pigeon axioms*) and the other clauses claim that no two pigeons can sit in the same hole (*hole axioms*).

Further reading

In this lecture we saw a brief introduction to the main concepts of proof complexity. Other than the references already mentioned, it may be useful to check the following surveys or book chapters. More references specific to each lectures will be included in later handouts.

Surveys

- *The Complexity of Propositional Proofs.* general-purpose survey of proof complexity. [8]

[8] Nathan Segerlind. The complexity of propositional proofs. *Bulletin of symbolic Logic*, 13(4):482–537, 2007

- *Propositional Proof Complexity: Past, Present and Future.* An older general purpose survey. [9]
- *Towards NP–P via Proof Complexity and Search.* A survey motivated by questions related to search problems, and to the P vs NP problem. [10]

Books and book chapters

- Krajíček: *Bounded Arithmetic, Propositional Logic, and Complexity Theory* (Cambridge University Press)
- Chapter 5 in Clote, Kranakis : *Boolean Functions and Computation Models* (Springer).
- Chapter 18-19 in Jukna : *Boolean Function Complexity* (Springer)

[9] Paul Beame and Toniann Pitassi. Propositional proof complexity: Past, present, and future. In *Current Trends in Theoretical Computer Science*, pages 42–70. World Scientific Publishing, 2001

[10] Samuel R. Buss. Towards NP–P via proof complexity and search. *Annals of Pure and Applied Logic*, 163(7):906–917, 2012

References

[AR08] Michael Alekhnovich and Alexander A. Razborov. Resolution is not automatizable unless W[P] is tractable. *SIAM J. Comput.*, 38(4):1347–1363, 2008.

[BP01] Paul Beame and Toniann Pitassi. Propositional proof complexity: Past, present, and future. In *Current Trends in Theoretical Computer Science*, pages 42–70. World Scientific Publishing, 2001.

[Bus12] Samuel R. Buss. Towards NP–P via proof complexity and search. *Annals of Pure and Applied Logic*, 163(7):906–917, 2012.

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

[EGG08] K. Eickmeyer, Martin Grohe, and Magdalena Grüber. Approximation of natural w [p]-complete minimisation problems is hard. In *23rd Annual IEEE Conference on Computational Complexity*, pages 8–18. IEEE, 2008.

[Göd93] Kurt Gödel. A letter to John von Neumann, March 20, 1956. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 7–9. Oxford University Press, 1993.

[Iwa97] Kazuo Iwama. Complexity of finding short resolution proofs. In Igor Prívara and Peter Ruzicka, editors, *MFCS*, volume 1295 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 1997.

[MMZ+01] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[MSS99] João P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *Computers, IEEE Transactions on*, 48(5):506–521, 1999.

[Seg07] Nathan Segerlind. The complexity of propositional proofs. *Bulletin of symbolic Logic*, 13(4):482–537, 2007.