Complexity of Symbolic Computing

Peter van Emde Boas
Logic & Computation Theory Group
Department of Mathematics and Computer Science
University of Amsterdam, P.O. Box 19268, 1000 GG Amsterdam
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

We present a short introduction to complexity theory and the fundamental questions on complexity classes, like $P=NP$, $NP=co-NP$, etc. We discuss the link between these questions and the problem of propositional inferencing, and illustrate their impact on Artificial Intelligence.

1. Triumph and defeat

The area of symbolic computation includes a number of different subjects. At the same time symbolic computation can be used as a stage to illustrate the current developments of complexity theory. We clearly observe here the difference in speed and extent to which progress occurs in science. On the one hand we witness the major breakthroughs in the field of polynomial arithmetic, where due to the discovery of the Lovasz Lattice Reduction Algorithm the complexity of polynomial arithmetic has been brought down from exponential to polynomial level. At the same time we are also faced with the discouraging state of affairs that our knowledge on the subject of the complexity of propositional inferencing is about as poor as about twenty years ago. We don't know whether it is reasonable to assume that an inference engine, even if it is just faced with the simple job of performing propositional inferences, will be capable of performing such inferences within a reasonable amount of time. As it turns out we even are ignorant with respect to the question of whether there exist any short proofs the inference engine should be looking for, either in the logical formalism for which it is designed or in any formalism at all. The two problems above, in fact, are nothing but incarnations of the two notorious open questions in elementary complexity theory: the $P=NP$? and the $NP=co-NP$? problems.

The two complexity theory problems above have been among us now for about 18 years; their interpretation in the terminology of propositional inference has been understood as such for long. Still, the fact that these problems still are unsolved does not imply that there has been no progress on this subject in the context of inferencing. Quite to the contrary the theory has yielded a number of results along two marching roads for attacking the problems:

establishing the insufficiency of proposed inference methods and providing short proofs for hard propositional formulas.

The present paper is inspired by the recent appearance of one more result in the latter area. In [1] SAMUEL R. BUSS shows that a set of propositional formulas which expresses the truth of the well-known pigeon-hole principle can be proven using short proofs in a standard Hilbert-like proof system. Rather than explaining how this result was proved, I would like to use the opportunity of being invited to contribute to the special issue of the CWI-quarterly on symbolic computation to explain what this result is about and what it has to say to those researchers who should worry the most on the complexity aspects of inferencing, but who in practice seem to care least about it: the designers of AI systems.

This paper is organized as follows. I will present a short introduction to complexity theory and to the fundamental problems on complexity classes like P,NP, and $co-NP$. Next I will explain the link between these fundamental questions and the unknown complexity status of the problem of propositional inferencing. The paper concludes by indicating some recent results on the complexity of inferencing and their impact on Artificial Intelligence in general and for the viability of the well-known resolution proof strategies which are highly popular among AI researchers in particular.

2. WHAT CONSTITUTES COMPLEXITY THEORY

In order to explain what the two fundamental problems in complexity theory mentioned above involve I must first explain how complexity theory is brought about. This theory deals with the complexity of computations based on some machine model. So we first should select our favourite model of a (sequential) machine, like the Turing Machine or RAM. Subsequently we select our natural time/space measures, say atomic steps of the Turing machine as a measure for time and tape squares used during the computation as a measure for space. In order to have these machines do something useful we must make it possible for them to solve problems; therefore we must invent some natural method of encoding problems using a finite alphabet, and using for example binary notation of numbers and so on. We must explain what it means for some machine to solve a problem. Having done so we stipulate a basic assumption, based on the collective experience of the programmers of this world, which was for the first time stated explicitly by EDMONDS [3]: All that we can hope to achieve in Practice is solving problems which are *good*. In this assumption the word *good* is explained by the following definition:

Problem X is good if there exists a deterministic algorithm M and a constant k such that M solves any instance x of X in time $\leqslant k.|x|^k$, where $|x|$ denotes the length of the encoding of x.

Solving a problem can mean providing a yes/no answer (decision problem), evaluating a function (functional problem) or finding a proof (theorem proving), etc... .

We have now achieved the point where we can introduce the protagonist in the tragedy of complexity theory: P denotes the class of decision problems

which are good in the above sense.

The introduction of the antagonist in our tragedy requires the concept of a nondeterministic computation. This concept is far less intuitive than the concept of a computation itself. We can express it by the following definition:

A nondeterministic machine M is a device which when processing an input x first generates by performing arbitrary choices an additional segment of input y, called the certificate. Next the machine performs a deterministic subcomputation on the combined input (x,y) during which it will test whether the certificate y is a witness for the solvability of x. This subcomputation will allways terminate and will either produce a solution to the original problem instance x, or it will determine that the certificate y is not a good witness for x.

If the instance x is solvable then there exists a certificate y which can be obtained by the machine by guessing, and the machine may subsequently perform its deterministic subcomputation on the pair (x,y) producing the right solution. But if the instance x is not solvable then all certificates will turn out to be incorrect.

Since the concept of nondeterminism is quite unnatural I explain it by means of the example of the problem of factorizing an integer n. Our nondeterministic machine solves this problem by guessing a trial divisor d and subsequently perform the division of n by d. If n is not a prime number there exists a proper divisor d of n which the machine may guess as a certificate of the fact that n is composite. But if the number n is prime then no proper divisor of n exists so the trial division will fail whatever d is proposed.

Now it is crucial to observe that although the deterministic subcomputation, i.e., the trial division, can be performed by a good (polynomial time bounded) algorithm we have no deterministic good algorithm for solving the factorization problem. This problem represents the main irritating open problem of elementary number theory (it was already recognized as a problem by Gauss). The above nondeterministic algorithm is however good in the sense that it consumes polynomial time.

This brings us to the next definition: NP denotes the class of decision problems X for which there exists a nondeterministic solution which runs in polynomial time. A consequence of this definition is that for a problem in NP the certificate y which establishes the solvability of a solvable instance x must have length which is polynomially bounded by $|x|$. Otherwise the machine will not be able to see the whole certificate during its deterministic subcomputation.

The classes P and NP belong to a much larger hierarchy of fundamental complexity classes:

$$\text{LOGSPACE} \subseteq N\text{LOGSPACE} \subseteq P \subseteq NP \subseteq$$
$$P\text{SPACE} = NP\text{SPACE} \subseteq EXP\text{TIME}.....$$

but the other classes in this hierarchy are not relevant for our present paper.

At this point I must explain something which should have bothered the well-trained, mathematical reader. I start with the selection of some machine model with complexity measures and next I define the property of an algorithm of being *good* in terms of the running time of algorithms on such machines. Consequently everything defined so far is machine model dependent. It turns out, however, that the above hierarchy of fundamental complexity classes is machine independent, provided the right choices have been made in selecting your machine model, complexity measures and encoding. This is a consequence of the so-called *invariance* thesis which states that all reasonable models of sequential computation simulate each other with polynomially bounded overhead in time and constant factor overhead in space. The standard models which are used in complexity theory obey this thesis, at least if they are presented with sufficient care; for more details on this question I refer to my chapter in the forthcomming handbook of theoretical computer science [11].

What is more relevant for our present considerations is the fact that all inclusions represent an unsolved problem: Is the inclusion proper or not? So in particular we face the problem of whether $P = NP$? or not. We do know however that somewhere in the hierarchy proper inclusions must occur since $P \neq EXP$TIME and LOGSPACE $\neq P$SPACE.

There exists a third role in our tragedy which so-far has not been introduced: the class $co-NP$. Let me just state for the moment that this class consists of all set-theoretical complements of sets in NP; a set belongs to $co-NP$ in case there exists a good nondeterministic algorithm which decides that elements x don't belong to the language. For example, the trial division algorithm for detecting nondeterministically composite numbers illustrates the fact that the set of prime numbers belongs to $co-NP$.

As an aside I can mention at this point that there exists also a nondeterministic algorithm based on certificates for primality of polynomial bounded length; this algorithm establishes that the set of primes belongs to NP (and consequently that the set of composite numbers belongs to $co-NP$). The required polynomial estimate on the length of these primality certificates was given by PRATT [10]. The set of prime numbers therefore is a member of $NP \cap co-NP$.

3. PROPOSITIONAL LOGIC

Next we will indicate the connection between the above fundamental problems on complexity classes and propositional logic. I first give a short description of propositional logic.

The traditional language of propositional formulas is obtained as follows: we have constants (truth values) and (propositional) variables. Furthermore there are auxiliary symbols, like connectives and parentheses, for syntactic disambiguation. From these basic objects we define formulas (not to be conserved with the formulas from predicate logic as defined in the paper of Marc Bezem) by an inductive definition:

Constants: 0, 1 .
Variables: $p[0],p[1],p[10],p[11],....$
connectives: $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, ...$
parentheses: (,)
Formulas:
Variables and constants are formulas
If f and g are formulas then so are $(\neg f),(f \vee g),(f \wedge g),(f \Rightarrow g),(f \Leftrightarrow g)$

The formulas in propositional logic are subsequently provided with a semantics which, given an assignment of truth values 0 or 1 to the variables in a formula, yields a truth value for this formula by use of the traditional truth table definition. The formula f is *satisfiable* if and only if there exists a truth value assignment to its variables making it true. The formula f is *valid* (in which case f is called a *tautology*) if and only if every truth value assignment to its variables makes it true.

Next we can introduce the following two problems on propositional formulas:
SATISFIABILITY: the problem of determining whether a given formula is satisfiable;
TAUTOLOGY: The problem of deciding whether a given formula is tautological and finding a proof for its validity.

The fundamental link between propositional logic and complexity theory is based on the following observations:

FACT 1: *The problem SATISFIABILITY belongs to NP.*
This is an easy consequence of the fact that any satisfying truth assignment represents a certificate for satisfiability; it is not hard to see that the deterministic subcomputation which evaluates the formula is a good algorithm.

FACT 2: *The problem SATISFIABILITY is a hardest problem for the class NP.*
This fact, which is known as Cook's theorem [6], is far more difficult to establish. It is shown by proving that any other problem in *NP* can be translated with polynomial bounded size overhead in polynomial time into SATISFIABILITY by encoding complete records of machine computations into propositional variables. A complete record of a computation can be represented by a two-dimensional array of characters, which, in turn, can be encoded by a bit-string. This bit-string can be seen as a value assignment to a large collection of propositional variables (as many as the length of the string). The necessary conditions guaranteeing the correctness of the encoding and the fact that the machine computation described represents an accepting computation are locally testable conditions on individual bits expressible by reasonably short propositional formulas; the conjuction of these formulas for all instances of these local conditions now expresses the existence of an accepting computation and becomes the transformed version of the given instance of the original *NP* problem.

This encoding has the property that the resulting propositional formula is satisfiable if and only if the instance of the given *NP*-problem is solvable. For if the instance is solvable a witness exists and therefore also some accepting computation record, and its encoding as a bit-string provides the value assignment which will satisfy all local conditions and consequently the entire formula. Conversely, from an assignment of truth values to the propositional variables in this formula which makes the formula true the original accepting computation and consequently the witness of solvability of the original instance can be reconstructed.

FACT 3: *Formula f is satisfiable if and only if* $\neg f$ *is not a tautology.*
This observation indicates, to some extent, that for a mathematician SATISFIABILITY and TAUTOLOGY represent the same problem. However, the second complementation involved in this translation shows that TAUTOLOGY belongs to the class $co-NP$ of complements of *NP* problems rather than to the class *NP* itself. In this class it is a hardest problem due to the *NP*-completeness of SATISFIABILITY. Also the role of the witnesses becomes different. A satisfying value assignment is a witness for the satisfiability of a formula and, similarly, a falsifying assignment is a witness for its non-validity. Witnesses for validity of a formula in mathematics are of a different nature; we know such witnesses under the name of proofs. The completeness theorem for propositional logic which states that the valid formulas and the provable formulas coincide, implies that proofs can be used as witnesses for validity and conversely.

FACT 4: $P=NP$ *if and only if SATISFIABILITY and recognizing tautologies are good problems.*

FACT 5: $NP=co-NP$ *if and only if tautologies have polynomially bounded proofs.*
These last two facts are based on very general observations on the reducibilities involved in the proof of Cook's theorem. They show that two fundamental problems in elementary complexity theory have equivalent expressions in terms of problems on formulas in propositional logic. Both problems have been open for almost two decades.

The connection with work in artificial intelligence can be explained as follows. A basic requirement on intelligent behaviour is that a machine should not need to have everything explained to it. Having been told some basic facts, the machine should be able to infer the trivial consequences itself. Since the fact that formula f is a consequence of the facts $g_1,\ldots,g_k$ is equivalent to the validity of the implication $(g_1\wedge\ldots\wedge g_k)\Rightarrow f$ this latter formula should be a tautology and the machine should be able to recognize it as such. Moreover the machine should be able to justify its behaviour by providing explanations on request and these explanations must entail the validity of the above implication. These explanations should better be proofs for this implication. Even if its validity has been obtained initially by the machine by throwing dices we

still would like to see a short proof.

It is curious to observe that in some sense the dream of AI researchers of the intelligent machine (which seems to presuppose that $P=NP$) is in total contradiction with the basic assumption underlying all work in contemporary complexity based cryptography [8,9], where all proposed cryptosystems are based on intractability assumptions on factoring of integers or other number theoretical problems, the existence of one-way functions (i.e., functions which are easy to compute but hard to invert) or other hard problem instances, all of which are nonexistent in case $P=NP$. So here we have a prime example of two schools of active computer science researchers (both being represented at the CWI) which are betting at incompatible states of affairs in the real world. It is our collective ignorance on the true state of affairs in the real world which keeps both groups supported.

4. The complexity of propositional inferencing

This paper is definitely not the proper place to inform the reader about all the fundamental work which has been performed in the attempts of solving the $P=NP$ and $NP=co-NP$ problems. The reader is referred to the monograph of Garey & Johnson [6] and the subsequent updates by Johnson in the *Journal of Algorithms*. What I want to discuss in this paper is the work in this problem area which is specifically connected to the expression of these fundamental problems in problems about propositional logic. Work in this area has proceeded along two lines of attack:

I. Establish of a given proof procedure that it must yield exponential size proofs on well designed tautologies.
   The work by Tseitin & Galil [4,5] on Resolution proof procedures described below is a prime example of this approach.

II. Establish 'difficult' classes of formulas for which still short proofs exist.

This second line of attack I will illustrate with the work by Cook & Reckhow [2], Haken [7] and Samuel R. Buss's [1] work on the propositional Pigeonhole Principle.

Let us first introduce the resolution proof method which has become very popular for AI work since it bases theorem proving on a single proof rule. Resolution proof techniques operate on formulas in Disjunctive Normal Form (DNF); this is a class of propositional formulas which is defined as follows:

A DNF formula is the conjunction of a set of clauses.
A clause is a disjunction of literals.
A literal is a variable or a negated variable.

An example of a DNF formula: $(p[0]\vee\neg p[1]\vee p[10])\wedge(\neg p[0]\vee\neg p[10])$.

It can be shown that every propositional formula can be transformed into an equivalent formula in DNF. A straightforward translation procedure which does not introduce new propositional variables may produce an exponential blow-up of the formula size, in particular when the DNF expansion involves

many applications of the distributive laws or when the connective $\Rightarrow$ or the exclusive-or operator has to be expanded. But if we are willing to introduce fresh propositional variables the transformation can be performed in such a way that the length of the formula is multiplied by at most a constant factor; moreover, the number of fresh variables required for this transformation is bounded by the number of connectives in the original formula.

The *Resolution* rule which forms the basis of many working theorem provers which are in use in the AI world can be formulated as follows:
From clauses $p[i] \vee f$ and $\neg p[i] \vee g$ infer the new clause $f \vee g$ provided the formulas f and g have no other complementary pair of literals.

A *Resolution proof* of a formula is obtained as follows:
In order to prove that f is a tautology express $\neg f$ as a DNF formula (conjunction of clauses). Apply to the resulting set of clauses the resolution rule in order to see whether the empty clause can be derived. This empty clause represents contradiction. The resulting refutation is structured as a resolution proof tree.

In the language of resolution proofs the full complexity of theorem proving remains visible:

FACT 6: *Satisfiability of formulas in DNF remains NP-complete (even if clauses are assumed to have length at most 3).*

FACT 7: *Refutation of formulas in DNF (i.e., finding existence of resolution proofs) is complete for co $-$NP.*
The resolution rule itself doesn't tell you how to select the pairs of clauses on which it should be applied, and in which order these pairs should be selected. Finding resolution proof trees is a nondeterministic procedure. However, given a candidate tree its validity as a resolution tree can be established in time proportional to its size. So if for every refutable DNF formula a short resolution refutation exists the set of unsatisfiable DNF formula's will become a member of NP, from which it will follow that $NP = co - NP$.

The algorithm of applying the resolution rule in order to find a proof is a nondeterministic procedure, which easily leads to fruitless or even nonterminating computations. In order to prevent the theorem prover from attempting resolution steps which don't lead to any profit various ways of controlling this procedure have been proposed. Some of the proposed heuristics are:
David-Putnam Procedure: eliminate all literals in some order.
Regular resolution: Never reintroduce a literal which has been resolved away before on that branch in the proof tree.
k-Bounded resolution: Never generate clauses involving more than k literals (for some constant k).

TSEITIN & GALIL [4,5] have shown that none of the above heuristics yields a good resolution proof strategy: Either the strategy fails to solve correctly some

instance of a valid propositional formula or it can be shown to consume exponential time. This is established by confronting these strategies with 'bad' instances of formulas which are obtained as follows:
Consider a graph $G=(V,E)$ of bounded degree k. Assign to each node $v \in V$ a parity $a(v)=0$ or 1. Assign to each edge e in E a literal $x[e]$. Next consider the statement that for all v the exclusive or of the edge literals corrresponding to incident edges equals $a(v)$. If the sum of all parities $a(v)$ is odd this statement is false.

The above statement is not yet represented as a DNF formula, but expressing it in DNF requires a polynomial overhead in size, since expressing the condition at each node requires 2^{k-1} clauses of k literals, and k is a constant.

It has been shown by Tseitin and Galil that each resolution refutation which used the David Putnam strategy or any other regular resolution strategy requires the generation of clauses corresponding to all connected subgraphs of the given graph. Since there are exponentially many of those connected subgraphs this yields an exponential lower bound for the size of a proof tree. At the same time it can be shown that the strategy of k-bounded resolution (which by definition can produce at most a polynomially bounded number of clauses) will fail to produce a resolution refutation at all for these evidently false formulas. So the proposed strategies either fail to work or they require exponential running time.

In defence of resolution it must be mentioned that up to now not all resolution proof strategies have been shown to require exponential size proofs. The strategy of Extended resolution is still a viable candidate for doing the job in a polynomially bounded proof tree. This technique is based on the introduction of new concepts by abbreviation: fresh variables are introduced by throwing in new clauses which contain these variables and which define the meaning of these variables as a propositional expression in the original ones. A set of clauses will never become inconsistent by this process of adding defined concepts, and therefore a refutation of the extended set of clauses will only be found if the original set already could have been refuted. However, the refutation of the larger set may turn out to be much shorter than that of the original one.

This phenomenon could be invoked as a justification for the everyday practice of mathematicians: the introduction of defined notions. One occasionally hears the formalistic objection against mathematics that nothing is created in mathematics since the truth of the theorems already is determined when the axioms are formulated; everything else is a direct consequence which could be reduced by standard methods to the application of trivial manipulations of a logical nature. (Let the symbols do the work!) This objection however overlooks the impact of the introduction of defined concepts. They might be eliminated from the proofs of all theorems but presumably this would blow up the size of the proofs in mathematics to such an extent that no human being could understand what is happening. Moreover these new concepts are the inspiring part of mathematics; mathematicians survive by inventing new concepts.

The results above show that, even if there exists a resolution proof technique which provides short proofs for propositional formulas, so far we have not found it. The fact that extended resolution proofs may be short won't help us as long as nobody provides us with the correct definitions for the new variables. None of the proposed strategies for standard resolution seems to work. Still we have insufficient justification for abandoning the resolution methods, since the same observations might hold for other propositional proof procedures as well, where examples of alternative proof procedures are provided by the Hilbert style proof systems known from classical logic textbooks or natural deduction proofs.

It is with respect to this issue that the results of Haken, Cook & Reckhow and Samuel Buss provide us with more bad news for the resolution method and its believers. These authors consider another example of a class of bad propositional formulas, the propositional pigeonhole principle, which is obtained as follows:

Consider the evidently true statement: If $k+1$ objects are mapped into k slots some slot will contain at least two objects. Letting the literal $p[i,j]$ denote the fact that object i is mapped onto slot j this statement is expressed by the following formula:

$$\wedge_{0\leq i\leq k}(\vee_{0\leq j<k}(p[i,j]))\Rightarrow\vee_{0\leq j<k}(\vee_{0\leq i<m\leq k}(p[i,j]\wedge p[m,j])).$$

The negation of this formula can be transformed easily into standard DNF and in this transformation the generalized conjunctions and disjunctions can be expanded with a constant factor overhead in size.

HAKEN [7] has shown that standard resolution proofs for this formula require exponential size, but polynomial size extended resolution proofs exist. COOK & RECKHOW [2] provide polynomial size proofs in an Extended Hilbert-like system where introduction of new variables by abbreviation is allowed. The new result by SAMUEL BUSS [1] shows how to get polynomial size proofs even without this extension feature. His proof technique is based on ideas which originate from the design of computer hardware: it can be argued that the core of the problem amounts to efficient counting and adding small numbers, and this is performed by a similar method as the one used in carry-look ahead addition.

These results therefore indicate that indeed resolution after all might not be the powerful proof procedure on which all AI work could be based. The propositional pigeonhole principle provides us with an example where extension is required for providing short resolution proofs, but Buss has shown that short proofs in a Hilbert style exist even without the use of the extension mechanism. So Hilbert systems seem to be more powerful than resolution.

On the other hand these results are still far away from solving the real fundamental problems from complexity theory: $P=NP$? or $NP=co-NP$? In all cases above the specific instances of tautologies considered still had a short proof for some ad hoc proof strategy.

References

1. Samuel R. Buss (1987). Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic 52,* 916-927.
2. S. Cook & R. Reckhow (1979). The relative efficiency of propositional proof systems. *Journal of Symbolic Logic 44,* 36-50.
3. J. Edmonds (1965). Paths, trees, and flowers. *Canad. J. Math. 17,* 449-467.
4. Z. Galil (1977). On the complexity of regular resolution and the Davis-Putnam procedure. *Theoretical Computer Science 4,* 23-46.
5. Z. Galil (1977). On resolution with clauses of bounded size. *SIAM J. Comput. 6,* 444-459.
6. M.R. Garey & D.S. Johnson (1979). *Computers and Intractability, a Guide to the Theory of NP Completeness,* Freeman.
7. A. Haken (1985). The intractability of resolution. *Theoretical Computer Science 39,* 297-308.
8. E. Kranakis (1986). *Primality and Cryptography,* John Wiley & Sons.
9. A. Lempel (1979). Cryptography in transition. *Comp. Surveys 11,* 285-303.
10. V.R. Pratt (1975). Every prime has a succint certificate. *SIAM J. Comput. 4,* 214-220.
11. P. Van Emde Boas. Machine models and simulations. *Handbook of Theoretical Computer Science* (forthcoming).