

Propositional Satisfiability (SAT) as a language problem

José M. Castaño and Rodrigo Castaño
jcastano,rcastano@dc.uba.ar

Depto. de Computación, FCEyN, UBA, Argentina

Abstract. We present an approach to propositional satisfiability as a Finite State Automata automata construction problem. From a theoretical point of view it has consequences for languages beyond context free power. There are no consequences on complexity issues due to Automata construction (using intersection) is PSPACE-complete. From a practical point of view it was shown that this approach is competitive with ALL-SAT approaches and even with state of the art SAT solvers on traditional hard problems. Here, we show that techniques used in DPLL can be used in an automata approach. This kind of approach opens a new path of research on propositional satisfiability.

Keywords: ALL-SAT, model counting, FSA intersection, regular expression compilation, Non-clausal formula, clause learning

1 Introduction

There are tons of research done on the satisfiability problem (SAT) in propositional logic. This research has been mainly dominated by a search approach, in particular the DPLL algorithm. This is explained partly by the success obtained in this approach. A few other approaches were also pursued such as resolution based algorithms (DP), graph based algorithms (BDD).

Early studies by Büchi, Elgot [6, 10] and Trakhtenbrot, analyzed transformations from formulas to automata and vice-versa in the context of the relation between FSA and Monadic Second Order logic (MSO). Sometimes referenced as the Büchi-Elgot-Trakhtenbrot Theorem it was established that FSA and MSO have the same expressive power [17]. This was the basis of the approach that uses Büchi automata to decide satisfiability of modal logic formulas (LTL) [18]. This is also the only reference to an automata based approach to satisfiability found in [5]

Schützenberger, McNaughton, Papert and Kamp, established the equivalence between star-free regular expressions, counter-free finite state automata, first-order logic and temporal logic, see [15].

The approach presented here is framed into a more general enquiry about propositional satisfiability as a language or automata problem. As far as we know this question has not been pursued. It is difficult to know whether some of the

results that we will bring into attention, were not considered interesting or were not known, even though the existing connections are or look so obvious after presented. There are no references to it in either classical books on language and automata nor in the recently published Handbook of Satisfiability [5]. There is only one exception in [19], but this approach was not pursued further as a propositional satisfiability problem.

Using a different perspective, there are a number of works that relate, satisfiability, to language problems in order to show NP-completeness of the recognition problem for a particular language. Examples of this approach are the proofs that the word and generation problem for Two Level Morphology (TLM) are NP-complete. [3]. There are a number of proofs for many other language classes. It should be stressed that the focus in this case is the NP-completeness property and not an approach to SAT solving.

The only *machines* related to Satisfiability are binary decision diagrams (BDDs), which cannot be strictly regarded as language 'machines' (they do not generate/recognize a language), although they can be considered as a sort of finite state machine. Logic circuits are considered finite state machines, they are classically related to transducers like Mealy or Moore state machines. However the satisfiability of logic circuits (circuit-SAT), is tested using DPLL algorithms.

In the approach presented in [7, 8], given an ordering of variables valuations as strings in a binary alphabet. In other words, given a set of n variables and an ordering p_1, \dots, p_n and a binary alphabet, say $\{0, 1\}$, the set of possible valuations can be represented as the set of 2^n strings of the language $L_n = \{0_1, \dots, 0_n, \dots, 1_1, \dots, 1_n\}$.

Therefore the set of satisfying valuations for a formula in propositional logic with n variables can be represented as a subset of L_n . This is an acyclic regular language, where each word has the same length. The propositional formula is interpreted as a regular expression that specifies the language (and the corresponding automaton. Satisfiability, and model counting are computed by construction the finite state automata (in the same way that satisfiability for an LTL formula is computed constructing a Büchi automata. The costly operations to construct the automaton will be intersection and union [20]. From a theoretical point of view, this is no improvement, given that automata intersection is P-SPACE complete [11].

In [7, 8] it was shown that for every propositional formula in CNF, it is possible to construct a regular acyclic automata that describes the set of satisfying valuations. It also brings into question the power of FSA intersection to capture phenomena that are supposedly beyond context freeness [7].

From a practical point of view, this approach opens a path that has not been explored, with many possible ramifications. In [8] it was shown that the cost of constructing the automaton to solve a SAT problem will be influenced by choices such as ordering of variables, ordering of clauses (sub-automata). Here we also show how it is possible to DPLL strategies using a finite state automata approach. This alternative produces a change of perspective on SAT solvers which can provide a better understanding on the limits of current approaches. It

also provides a better understanding on Language problems which are considered NP-complete. The remainder of this paper is structured as follows, in section 2 we provide some definitions. In Section 3 we review the automata construction approach to SAT. In Section 4 we show how to preprocess CNF formula to add 'learned' clauses, using limited resolution. In section 5 we show how to compute non-clausal formulae on Iscas 85 benchmark, using XFST. Finally we present conclusions and future work.

2 Definitions

Most of these definitions and notation follow the ones given in [14].

$L(A)$ denotes the language generated by an Automaton or Grammar, A .

A clause is a propositional formula of the form $l_1 \vee \dots \vee l_n$, where each l_i is a *literal* a positive or negated propositional variable.

A term is a propositional formula of the form $l_1 \wedge \dots \wedge l_n$.

Valuations, are defined as functions v on a set of variables Var and with values in $\{0, 1\}$. Valuations assign a truth value from $\{0, 1\}$ to each propositional variable $p \in Var$. We denote the set of literals (positive or negated variables) determined by the set of variables Var by Lit . Then if $|Var| = n$, $|Lit| = 2n$. We say that a valuation v *satisfies* a formula ϕ or $v \models \phi$.

Complete set of literals:

A complete set of literals is a set $S \subseteq Lit$ such that for every $p \in Var$ exactly one of $p, \neg p$ belongs to S . There is a bijective correspondence between valuations and complete sets of literals. One such mapping associates positive literals with 1 and negative literals with 0. An alternative mapping associates positive literals with 0, and negative literals with 1. It follows that if $|Var| = n$, then there are 2^n complete sets of literals over the set Var

Total ordering. Truth values are ordered, $0 \leq 1$. Given an arbitrary ordering of Var there is a total (linear) ordering of valuations, which can be ordered lexicographically or anti-lexicographically. Valuations can be thought as elements of the Cartesian product $Val = \prod_{p \in Var} \{0, 1\}$. The Cartesian product Val can be ordered lexicographically or anti-lexicographically.

Therefore the ordered elements of Val may be represented as boolean n-tuples (given $|Var| = n$). Consequently the least element in Val , is the n-tuple $(0_1, \dots, 0_i, \dots, 0_n)$, where $0 \leq i \leq n$, and the maximum element in a valuation is $(1_1, \dots, 1_i, \dots, 1_n)$. Ordered elements in Val also can be represented as strings in $\{0, 1\}^n$, which can be (anti-)lexicographically ordered. We will say that a word w satisfies a formula ϕ ($w \models \phi$) iff w is the string representation of an element $v \in Val$ and $v \models \phi$.

3 CNF Satisfiability as a Regular Language Problem

We describe how to construct a FSA automaton A for each formula ϕ in CNF, such that the formula is satisfiable iff the language of A is not empty and for every word w in the language of A , $w \models \phi$, i.e. $L(A) = \{w \in \{0, 1\}^n \mid w \models \phi, n = |V_\phi|\}$. This means that the language of the automaton is the string representation of the set of valuations v such that $v \models \phi$.

The construction is based on the mapping between clauses and the dual terms. It is also based on the direct translation between boolean formulas and regular expressions, given the direct correspondence between \vee, \wedge, \neg and $|, \&, \sim$, respectively and the closure properties of finite state automata.

Following [14], we can define inductively the semantics of propositional formulae with the operators \neg, \vee, \wedge in terms of the set of words w in $\{0, 1\}^n$ representing valuations.

1. $w \models \top, w \not\models \perp, \forall w \in \{0, 1\}^n$
2. $w \models p$ if p is a variable and $w_p = 1$,
where w_p denotes the order position of p in w and we use W_p to denote $\{w \mid w \models p\}$.
3. $w \models \neg\phi$ if $w \not\models \phi$ or equivalently $w \notin W_\phi$
4. $w \models \phi \wedge \psi$ if $w \in W_\phi$ and $w \in W_\psi$ or equivalently $w \in W_\phi \cap W_\psi$
5. $w \models \phi \vee \psi$ if $w \in W_\phi$ or $w \in W_\psi$ or equivalently $w \in W_\phi \cup W_\psi$

Let T be a set of formulas, we write $w \models T$ if for every $\phi \in T$, $w \in W_\phi$.

Each clause in a CNF formula will be interpreted as a regular expression that describes the automaton representing the set of valuations that satisfy each single clause. For a formula with m clauses the automata to be constructed will correspond to the intersection of m clauses. For instance, a clause of a formula in CNF, such as $v_1 \vee v_2 \vee v_3$, with $|Var| = 10$ the number of variables in the formula, will be translated as the regular expression $\wedge^{\wedge}000\dots\dots'$ with an extended use of the complement operator (\wedge) with scope over the complete expression. Thus $\wedge^{\wedge}000\dots\dots'$ matches any string in $(0|1)^{10}$, that does not start with 000 .

The novelty in the approach is the use of the following steps:

- a. Reorder variables,
- b. reorder clauses,
- c. translate formula into A regular expression.
- d. use an available FSA toolbox to compile the regular expression into a DFA

The first two steps mean BOTH variable ordering (also known to be important in BDD and DPLL, to properly prune the search/construction space) and sub-automata intersection ordering (clause ordering) are important for obtaining an efficient processing time. The computational cost as reflected by the size of intermediate automata is evidenced by processing time.

Step c. above is obtained as follows:

- Represent the set of possible valuations as $(0|1)^n$, this is the first initial r
- translate each clause as a regular expression r of length n such that:
 - each positive literal with variable order $p \geq 1 \leq n$ is translated as the character '0' in position p in r .
 - each negative literal with variable order $p \geq 1 \leq n$ is translated as the character '1' in position p in r .
 - for each variable p which has not a corresponding literal use the wild-card expression '.' in position p
 - construct the regular expression r .
 - join the set R of r expressions with the intersection operator.

Clearly the asymptotic complexity of the automata construction is $O(n^m)$, which might be worse than the exponential boundary $O(2^n)$. This is no surprise given that the automata intersection problem is a PSPACE problem, therefore there is no theoretical interesting result in terms of computational complexity. It is however at the heart of a number of correlated problems, see [13].

This well known fact about the complexity automaton intersection and union [20], provides an explanation, on why even if this approach is so transparent, it was not pursued. A direct implementation will result in an inefficient approach as we will see in the next section. However, in [8] we have shown that it is possible to address ALL-SAT and model counting [12] using finite state automata construction, and that this approach is capable of reaching a competitive performance compared to state of the art sat solvers. In [8] we showed that variable and clause order highly affect the automata intersection. It is also shown that this approach is capable to reach a reasonable performance as compared to other equivalent approaches. In particular it can be compared with BDD (binary decision diagrams) and NNF (negated normal form) approaches. It is more efficient in a number of traditionally hard benchmarks, in particular in unsatisfiable cases. The following table shows a comparison of the construction of the automata using XFST[4], with FORCE [1] heuristic to order variables, and anti-lexicographic ordering of clauses. These and other results are discussed in [8].¹

Heuristics	Force-AL	sbsat	ebddres	c2d	relsat	sharpsat	clasp
Total time	10847,41	18050,99	10764	16969	21637,99	18320,99	15471
Not Solved	15	28	17	27	29	29	21
Solving Time	1847	4250	564	2569	4237	920	2871
Solved	36	23	34	24	22	22	30
Average	51,39	184.78	16.58	107.04	192.58	41.81	95.7

Table 1. Summary: FSA with Force and AL clause ordering vs other solvers

¹ See also the times for some of the same problems using Minisat2 and other solvers at <http://evanescent.googlecode.com/files/results-DAC2002-deescover-v0.1.html>.

4 Automata Generalization and Clause learning.

Clause learning [16], one of the improvements to the DPLL algorithm has been shown to be equivalent to unrestricted resolution and there have been different proposals to restrict it (e.g. Extended Resolution [2]). We present here clause learning as a sort of regular expression deduction, bounded in clause width and deductions order. This alternative and simple limitations to unrestricted resolution in this setting shows how clause learning can be directly implemented as a preprocessing step to the automata construction.

The algorithm we use to simulate a sort of *clause learning* uses two loops based on obtained variable order and anti-lexicographic clause order heuristics (steps a. and b. in previous section).

We use two *for loops*. The first starts by the first (lowest and most frequent) variable and compares all the clauses (following the anti lexicographic order) that have this variable, for any clause that has the dual variable, adds any resolvent clause that is not tautological. For instance, if we have the clauses $v_1 \vee v_3 \vee v_5$ and $\neg v_1 \vee \neg v_2 \vee v_6$ then the clause $v_1 \vee \neg v_2 \vee v_3 \vee v_5 \vee v_6$ is added in the corresponding anti-lexicographic order. The second loop, starts from the last (highest and less frequent) variable and performs the same operation. For instance, if there are clauses $v_4 \vee v_5 \vee v_6$ and $\neg v_1 \vee \neg v_2 \vee \neg v_6$, then clause $\neg v_1 \vee \neg v_2 \vee v_4 \vee v_5$ is added.

Deduced clauses in the previous loop will be available as they are stored in the anti-lexicographic ordering, and given we started from the last variable, the deduced clauses in this loop will be also available. In order to avoid exponential blow-up of resolution techniques only clauses that satisfy a limit in the number of literals are added. As a consequence of this preprocessing step, a number of 'learned' clauses are added to the original formula, and the original clauses are kept.

Table 2 summarizes some experiments using clause learning with different limits in the number of literals. Lit. limit is the maximum number of literals allowed in a *learned clause* and # clauses are the total number of clauses after the clause learning process finished. We can observe that the complexity of automata intersection (as reflected by the processing time), is inversely influenced by the number of intersections which are reduced in this case as we add more intersections.

problem	lit. limit	time	# clauses
uf100-01	4	248.78	3976
uf100-01	5	49.57	9091
uuf100-01	5	4.84	9667
uuf200-01	6	> 38min	48967

Table 2. Clause learning: examples of number of clauses added

The effect of clause learning can be appreciated in Figure 1 below, where in the upper part a formula with 20 variables is represented in anti-lexicographic

order. Each vertical line represents a clause with a literal in each position. The leftmost corner contains the shortest span in the most frequent variables. The rightmost part contains the largest span (from variable 1 to 20). The lower part shows the *space* marked with a line in the upper part that is now *filled* with new learned clauses, again in anti-lexicographic order. These new clauses restrict the space of possible solutions from variable 1 to 5.

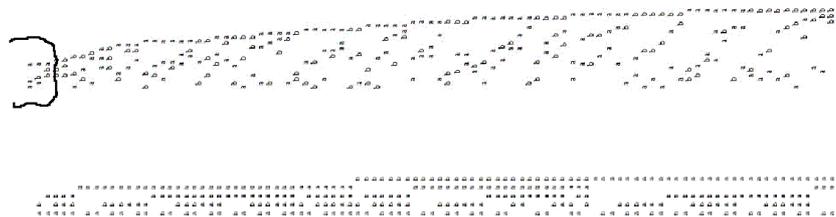


Fig. 1. Effect of clause learning

5 Non-clausal formula satisfiability.

One of the advantages of using a translation of propositional formula into regular expressions is that it is not necessary to restrict the input to clausal formulae. Therefore we can translate non clausal formula into regular expressions, using the mapping \vee, \wedge, \neg to $|, \&, \sim$ respectively. We describe how we used XFST to encode formulas in Iscas 85 format. Iscas 85 Benchmark files have the following syntax and one or more lines of each of the following types:

```
INPUT (VAR)
OUTPUT (VAR)
DEFVAR = OP(VARLIST)
```

where OP are for instance NOT, AND, NAND, NOR, OR and XOR. VARLIST is a list of one or more (according to the operator arity) INPUT or DEFVARs variables.

The translation into regex using XFST was as follows.

Start with the first line of the type VAR = OP(VARLIST), and replace it by *define gVAR*. The function *define*, in XFST defines an automata, and *gVAR* is the variable used by XFST to refer to this automata. In the body of the automata definition we replace the operator by corresponding regular expression operator mapping (i.e. , |or&). If there is an input variable in VARLIST, replace it by the regular expression [? ? ...1... ? ?] or if it is negated by [? ? ...0... ? ?]². There are as many positions available, as input variables in the formula. Last define the OUT variables. If there is only one OUT variable, the formula is satisfiable if there is an assignment of values for the input variables that satisfies the conditions of the OUT variable (and all the variables under it). If there are more OUT variables, it allows to test the properties of each OUT variable separately. This can be seen as a bottom up computation of the different expressions that compose the formula. The structural definition of the formula is respected.

Example 1 (Iscas non-clausal formula (left) XFST translation (right)).

```
# c17
# 5 inputs
# 2 outputs
# 0 inverter
# 6 gates ( 6 NANDs )

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)

OUTPUT(22)
OUTPUT(23)

20 = NAND(1, 3)
21 = NAND(2, 6)
16 = NAND(2, 11)
19 = NAND(11, 7)
22 = NAND(10, 16)
23 = NAND(16, 19)

define g10 ~[a ? a ? ?] & [a|b]^5;
define g11 ~[? ? a a ?] & [a|b]^5;
define g16 ~([? a ? ? ?] & g11 & [a|b]^5);
define g19 ~([? ? ? ? a] & g11 & [a|b]^5);
define g22 [a|b]^5& ~( g10 & g16);
define g23 [a|b]^5& ~( g16 & g19);
define OUT g23 & g22;
```

² We repeat that we used *a* for 1 and *b* for 0. In some cases, to the definition of the variable we add an intersection with $[0|1]^n$, where *n* is the number of input variables. This can be redundant but necessary to restrict the empty string as a possible value.

It can be observed in the results presented in table 3 that the ordering of variables has an impact on the performance of the XFST machinery. The reordering of variables was made using a simple strategy according to the most frequent variable across the sub-formulas. However no reordering of clauses was done to compute intersections or disjunctions. In the following table the reordering of variables is referenced by Var R(enamings), and Var O(riginal) with no variable mapping. We tried c2d ³ We report them c2d^t, c2d^r stands for the compilation time reported at the web page (see also [9]). OM means it has been reached the memory limit available at XFST.

Problem	Var R	Var O	c2d ^r	c2d ^t
c499.bench	6.54	2.44	8	9.40
c880.bench	0.36	134.62	46	49.92
c1355.bench	6.55	4.38	18	20.46
c1908.bench	1.24	1.99	81	222
c2670.bench	324.57	OM	350	1018.18
c3540.bench	73.47	107.94	-	-
c5315.bench	678.58	OM	-	-
c7552.bench	OM	OM	384	> 2700

Table 3. Non clausal tests against NNF compilation

The results however are not directly comparable, because the XFST implementation as it is apparent from the description, did not use a CNF input, however c2d was run on the CNF translation of the iscas format.

6 Conclusions and Future Work

We have used a direct transformation from a propositional logical formula into a regular expression. This transformation allows to construct the automata that generates the set of valid valuations for such formula. This approach represents a novel approach to SAT solving. Given this approach involves finding all the possible valuations it is not directly comparable with standard SAT solvers. We compared our approach with well established approaches both in clausal and non clausal formulae. We found that those approaches do not have an advantage over a FSA construction in the benchmarks that were used. This research opens a number of paths that need to be explored in order to assess in a proper way the capabilities of a FSA approach to SAT problems, and their relations and connections to other approaches. In particular we showed how to address clause learning and non clausal formulae. We also performed some initial experiments on assigning values to variables, by adding a conjunctive clause (a term with the assigned values). This initial and simple experiment showed an improvement of 10% in running time for the uf50 class in SATLIB, that contains 1000 formula.

³ We used the same parameters for c2d (-in memory -dt_method 0 -dt_count 25 -count) reported in c2d web page, but have some differences on timing.

References

1. F. A. Aloul, I. L. Markov, and K. A. Sakallah. FORCE: a fast and easy-to-implement variable-ordering heuristic. In *ACM Great Lakes Symposium on VLSI*, pages 116–119. ACM, 2003.
2. G. Audemard, G. Katsirelos, and L. Simon. A restriction of extended resolution for clause learning sat solvers. In *24th Conference on Artificial Intelligence(AAAI'10)*, 2010. To appear.
3. G. E. Barton. Computational complexity in two-level morphology. In *Proc. of the 24th ACL*, pages 53–59, New York, 1986.
4. K. Beesley and L. Karttunen. *Finite State Morphology*. CSLI Publications, 2003.
5. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2009.
6. J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik. Grund. Math.*, pages 66–92, 1960.
7. J. Castaño. Two views on crossing dependencies, language, biology and satisfiability. In *1st International Work-Conference on Linguistics, Biology and Computer Science: Interplays*. IOS Press, 2011.
8. J. M. Castaño and R. Castaño. Variable and clause ordering in an FSA approach to propositional satisfiability, 2011.
9. A. Darwiche. New Advances in Compiling CNF into Decomposable Negation Normal Form. In *ECAI*, pages 328–332, 2004.
10. C. C. Elgot. Decision problems of automata design and related arithmetics. *Transactions of the American Mathematical Society*, 1961.
11. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
12. C. P. Gomes, A. Sabharwal, and B. Selman. Model Counting. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 633–654. IOS Press, 2009.
13. George Karakostas, Richard J. Lipton, and Anastasios Viglas. On the complexity of intersecting finite state automata and $n \mid$ versus $n \mid p$. *Theor. Comput. Sci.*, 302(1-3):257–274, 2003.
14. V. W. Marek. *Introduction to Mathematics of Satisfiability*. Chapman and Hall/CRC, 2010.
15. I. Schiering and W. Thomas. Counter-free automata, first-order logic and star-free expressions. In *Developments in Language Theory II*, pages 166–175, Magdeburg, Germany, 1995.
16. J. P. Marques Silva and K. A. Sakallah. Grasp—a new search algorithm for satisfiability. In *in Proceedings of the International Conference on Computer-Aided Design*, pages 220–227, 1996.
17. M. Vardi. Logic and Automata: A Match Made in Heaven. In J. Baeten, J. Lenstra, J. Parrow, and G. Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *LNCS*, pages 193–193. Springer, 2003.
18. M. Y. Vardi and P. Wolper. Automata-Theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32:183–221, April 1986.
19. N. R. Vempaty. Solving Constraint Satisfaction Problems Using Finite State Automata. In *AAAI*, pages 453–458, 1992.
20. S. Yu, Q. Zhuang, and K. Salomaa. The state complexities of some basic operations on regular languages. *Theor. Comput. Sci.*, 125(2):315–328, 1994.