Satisfiability Problems Complete for Deterministic Logarithmic Space

Jan Johannsen

Institut für Informatik
Ludwig-Maximilians-Universität München
jjohanns@informatik.uni-muenchen.de

Abstract. The satisfiability and not-all-equal satisfiability problems for boolean formulas in CNF with at most two occurrences of each variable are complete for deterministic logarithmic space.

Introduction

The satisfiability problem (SAT) for formulas of propositional logic in conjunctive normal form (CNF) is the canonical complete problem for the complexity class **NP** [1] of nondeterministic polynomial time. Similarly, SAT problems restricted to several subclasses of CNF formulas are complete for smaller complexity classes.

For Horn formulas, i.e., CNF formulas where every clause contains at most one positive literal, satisfiability is complete for deterministic polynomial time **P** [2]. For formulas in 2-CNF, i.e., formulas where every clause contains at most two literals, satisfiability is complete for nondeterministic logarithmic space **NL** [3]. We exhibit the first known natural special cases of SAT that are complete for deterministic logarithmic space **L**.

Let CNF(2) be the class of formulas $F \in$ CNF such that every variable occurs at most twice in F, and let SAT(2) be the problem SAT restricted to instances in CNF(2). It is well-known that SAT(2) can be decided in linear time (see e.g. the book by Kleine Büning and Lettmann [4]). We will show that SAT(2) is complete for **L**.

The not-all-equal-satisfiability problem (NAE-SAT) is a variant of SAT that is studied in many contexts. Given a formula in CNF, the question is whether there is a satisfying assignment that also falsifies at least one literal in every clause.

In general, NAE-SAT is **NP**-complete for those classes of CNF-formulas for which also SAT is **NP**-complete. NAE-SAT restricted to formulas in 2-CNF is complete for symmetric logarithmic space **SL** [3, 5]. Recently, Porschen et al. [6] have shown that NAE-SAT(2), defined analogously to SAT(2), is solvable in linear time, and is in the parallel complexity class **NC**, their proof actually shows it is computable in parallel logarithmic time by a nearly linear number of processors, and thus is in $\mathbf{AC}^1$. We will show here that NAE-SAT(2) is in, and in fact complete for **L**.

It should be noted that our logarithmic space algorithms, in contradistinction to the algorithms mentioned above, only solve the decision problems SAT(2) and NAE-SAT(2), they do not give a witnessing assignment in case of a positive answer. However, after a draft of this paper was circulated, Stephen Cook (personal communication) and Mark Braverman [7] have given algorithms to construct satisfying assignments for satisfiable CNF(2)-formulas in logarithmic space.

It is easily checked that all the reductions we construct can be written as first-order reductions, given the usual encoding of the problem instances as logical structures (see Immerman [8] for background on these notions.) Therefore, all our reductions are uniform $\mathbf{AC}^0$ many-one reductions.

Satisfiability

In this section we show the $\mathbf{L}$-completeness of SAT(2). To this end, we reduce SAT(2) to a problem on a certain class of graphs:

A *tagged graph* $G = (V, E, T)$ is an undirected multigraph (V, E) with a distinguished set $T \subseteq V$ of vertices. We refer to the vertices in T as the *tagged* vertices.

We call a connected component in G tagged, if it contains at least one tagged vertex, and untagged otherwise.

From a formula $F \in \mathrm{CNF}(2)$, we construct a tagged graph $G(F)$ as follows:

- $G(F)$ has a vertex v_C for every clause C in F.
- If clauses C and D contain a pair of complementary literals x and $\bar{x}$, then there is an edge e_x between v_C and v_D.
- If C contains a pure literal, i.e., a literal a such that the complementary literal $\bar{a}$ does not occur in F, then v_C is tagged.

Note that there can be parallel edges between clauses containing more than one pair of complementary literals.

The assignment of a value to a variable x in F corresponds to giving the edge e_x in $G(F)$ a direction, from the clause containing the literal among x, $\bar{x}$ that gets the value 1 to the one that gets the value 0. Thus a clause C is satisfied by an assignment if v_C has nonzero outdegree.

Since clauses that contain pure literals can always be satisfied, the following characterization of satisfiability is rather obvious:

Proposition 1. *A formula $F \in \mathrm{CNF}(2)$ is satisfiable iff the edges in $G(F)$ can be directed so that in the resulting directed graph, there is no untagged sink.*

This characterization leads us to the following lemma:

Lemma 2. *A formula $F \in \mathrm{CNF}(2)$ is satisfiable iff every connected component in $G(F)$ contains a tagged vertex or a cycle.*

Proof. It suffices to show that the condition on the right-hand side is equivalent to the condition from Proposition 1. Since it is obviously necessary, we only need to show it is sufficient.

Let a connected component C of $G(F)$ contain a tagged vertex v. Perform a depth-first-search of C starting from v, and direct every edge in the resulting tree towards the root v. This way, every vertex in C other than v will have an outgoing edge, so the only sink is v, which is tagged. The back-edges can be directed arbitrarily.

If a connected component C contains a cycle $v_1, v_2, \ldots, v_k$, then we direct the edges around the cycle. To obtain the direction of the other edges, perform a depth-first-search starting from $v_1, v_2, \ldots, v_k$ in order, but during the search from v_i, do not visit the vertices v_j for $j > i$. In the resulting forest, direct as above all edges in every tree towards the root v_i. This way, every vertex in C will have an outgoing edge, and the remaining edges can be directed arbitrarily. $\square$

In other words, F is unsatisfiable iff $G(F)$ contains a connected component that is an untagged tree.

Theorem 3. SAT(2) *is in* L.

Proof. It suffices to show that the condition in Lemma 2 can be verified in logarithmic space. We employ a technique that was used by Cook and McKenzie [9] to test in logarithmic space whether a graph is acyclic.

For a tagged graph $G = (V, E, T)$, let $\mathrm{D}(G) := \big\{(v, e)\,;\, e \text{ incident on } v\big\}$ be the set of darts of G, i.e., the ends of edges in G. For a dart $d = (v, e) \in \mathrm{D}(G)$, we denote v by $v(d)$ and e by $e(d)$. We consider permutations of the set $\mathrm{D}(G)$. The disjoint-cycle representations of the following two permutations can be easily constructed from G:

ρ_G is the product of the cycles $\big((v, e_1) \ldots (v, e_k)\big)$ for every vertex v, where $e_1, \ldots, e_k$ are all the edges incident on v.
σ_G is the product of the transpositions $\big((v, e)\,(u, e)\big)$ for every edge e, where e is an edge between vertices u and v.

By a result of Cook and McKenzie [9], from the disjoint-cycle representations of two permutations, one can compute the representation of their product in logarithmic space.

Hence we can obtain the disjoint-cycle representation of the product $\pi_G = \rho_G \circ \sigma_G$. We will show how, using this representation of π_G, we can decide whether G contains a connected component that is an untagged tree.

We start a search from every dart $d \in \mathrm{D}(G)$. If the search is successful for every d, then we accept, otherwise we reject.

The search procedure performs two nested walks of the graph along the orbits of π_G. The outer walk is started at $w_1 := d$, then the inner walk is started at $w_2 := w_1$. It repeatedly remembers $e' := e(w_2)$, and then sets $w_2 := \pi_G(w_2)$, until either a tagged vertex is found, i.e., $v(w_2) \in T$, or the walk returns to w_1, i.e., $v(w_2) = v(w_1)$. In the first case, the search terminates successfully. In the second case, the search is successful if the walk did not return to $v(w_1)$ through $e(w_1)$, i.e., $e' \neq e(w_2)$.

If none of these cases occur, then the outer walk is continued by updating $w_1 := \pi_G(w_1)$. If $w_1 = d$, then the search terminates unsuccessfully, otherwise the inner walk is started again.

Note that the algorithm only stores two darts and one edge, so it runs in logarithmic space. The problem is therefore in L, since logarithmic space functions are closed under composition. To verify the correctness of the algorithm, we need to prove the following claim:

Claim. For every dart $d \in \mathrm{D}(G)$, the search from d terminates unsuccessfully if and only if the connected component of G containing $v(d)$ is an untagged tree.

The "if" direction is obvious. For the other direction, we use the following observation: if for every d' in the orbit of d, the walk along π_G returns to $v(d')$ through the edge $e(d')$, then the component of $v(d)$ is a tree, which is seen as follows:

If a vertex is reached through the edge $e = e_1$, then the walk will traverse every other edge leaving v before returning on e. In fact, if

$$\big(\,(v, e_1)\,(v, e_2)\,\ldots\,(v, e_k)\,\big)$$

is the orbit of (v, e_1) in ρ_G, then the walk will traverse the edges $e_2, \ldots e_k$ in that order before returning on e_1: if u_i is the other vertex incident with e_i, then $\pi_G\big((u_i, e_i)\big) = (v, e_{i+1})$.

It follows inductively that the walk visits the entire component of $v(d)$. It also follows that the component contains no cycle, by the following argument of Cook and McKenzie [9]:

Let $v_1, \ldots, v_k, v_{k+1} = v_1$ be a cycle, with edges e_i between v_i and v_{i+1}, and with v_1 reached first through edge e_0. By the above observation, for every i, at v_{i+1} the walk would traverse e_{i+1} before returning on e_i. Therefore, the walk returns through $v_1 = v_{k+1}$ through $e_k \neq e_1$, in contradiction to the assumption.

Therefore, if the search from d is unsuccessful, the component of $v(d)$ is a tree, which is untagged, since the walk would have encountered any tagged vertices present. □

Let SAT(2)$^-$ be the restriction of SAT(2) to instances that contain no pure literals, and let TF (tree-freeness) denote the following problem:

> TF: Given an undirected graph G, does every connected component in G contain a cycle?

As a consequence of Lemma 2, we obtain the following equivalence:

Proposition 4. SAT(2)$^-$ *is equivalent to* TF.

Proof. One direction is given by the construction above, which produces no tagged vertices when F contains no pure literals.

For the other direction, we can reverse the reduction as follows: For an undirected graph $G = (V, E)$, we construct a formula $F(G)$ as follows: we introduce one variable x_e for every edge $e \in E$, and for each vertex $v \in V$, we construct a clause C_v that contains one literal for each edge e incident to v. This literal is x_e, if e connects v to a higher numbered vertex, and $\bar{x}_e$ otherwise.

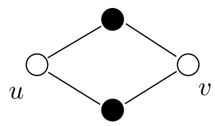
Obviously, $F(G)$ is a formula in CNF(2) with no pure literals, and $G(F(G)) = G$, so by Lemma 2, the construction is a reduction from TF to SAT$(2)^-$. □

Proposition 5. TF *is* **L**-*complete.*

Proof. TF is in **L** by Proposition 4 and Theorem 3. Its **L**-hardness remains to be shown.

We reduce the following problem $\overline{\text{UFA}}$, which is known to be complete for **L** [9], to TF: Given an undirected forest G consisting of exactly two trees, and vertices u and v in G, are u and v in different trees?

The reduction adds two new vertices to G, and connects them both by edges to u and v, as shown below, giving G'.



Now if u and v are in the same tree, then the other tree is still a tree in G'. If u and v are on different trees, then G' has only one connected component, which contains a cycle. Thus the construction reduces $\overline{\text{UFA}}$ to TF. □

From Propositions 5 and 4 above, we get that SAT$(2)^-$ is **L**-hard, therefore also SAT(2) is **L**-hard. Together with Theorem 3, this proves the main result of this section:

Theorem 6. SAT(2) *is* **L**-*complete.*

Not-all-equal-satisfiability

We are now going to show the **L**-completeness of NAE-SAT(2). We first consider the problem for the special case of monotone formulas, which turns out to be equivalent to another problem on tagged graphs.

Let an *isolated* clause be a unit clause such that the variable in this clause does not occur in any other clause. In this section we assume w.l.o.g. that formulas do not contain isolated clauses. This is possible, since no formula with an isolated clause is in NAE-SAT, and on the other hand such formulas are easily recognized.

Let mCNF(2) be the class of monotone formulas in CNF(2), i.e., formulas that contain only positive literals, and let mNAE-SAT(2) be the restriction of NAE-SAT(2) to instances in mCNF(2). Whereas satisfiability is trivial, NAE-SAT is **NP**-complete even for monotone formulas.

For a formula $F \in$ mCNF(2), we define the tagged graph $G'(F)$ by

- $G'(F)$ has a vertex v_C for every clause C in F.
- If clauses C and D contain the same literal x, then there is an edge e_x between v_C and v_D.
- If C contains a literal, that does not occur in another clause, then v_C is tagged.

Let E2C (edge-2-colorability) denote the following problem:

> E2C: given a tagged graph $G = (V, E, T)$, can the edges in G be colored by two colors such that every untagged vertex $v \in V \setminus T$ has incident edges of both colors.

The following characterization of mNAE-SAT(2) is rather obvious.

Proposition 7. *A formula $F \in$ mCNF(2) is in NAE-SAT iff $G'(F)$ is in E2C.*

Note that for a formula F with an isolated clause, the graph $G'(F)$ contains a tagged isolated vertex. If an isolated clause is added to a formula $F \in$ NAE-SAT$(2)^-$, then the resulting formula F' is no longer not-all-equal satisfiable, whereas $G'(F')$ is in E2C. Thus our assumption is needed for the equivalence to hold.

In fact, we can show that the two problems are equivalent.

Proposition 8. mNAE-SAT(2) *is equivalent to* E2C.

Proof. One direction is Proposition 7. For the other direction, given a tagged graph $G = (V, E, T)$, we define a formula $F(G) \in$ mCNF(2) as follows: for every edge $e \in E$, there is a variable x_e. For every vertex we form a clause C_v containing the variables x_e for the edges e incident on v. Finally, for every tagged vertex $v \in T$, we add a variable x_v to the clause C_v. It is easily seen that $G'(F(G)) = G$, and hence by Proposition 7, the construction reduces E2C to mNAE-SAT(2). □

Lemma 9. *An undirected graph G is in* E2C *iff the following two conditions hold:*

1. *every untagged vertex has degree at least two, and*
2. *there is no untagged connected component that is a simple odd length cycle.*

Proof. Both conditions are obviously necessary. To see that they are sufficient, we first show the following claim:

Claim. If the conditions above hold, then every untagged component C contains either an even length cycle, or two edge-disjoint odd cycles.

Start a walk from some vertex on C, that never leaves a vertex on the same edge it came from, which is possible by condition 1. Since C is finite, we must find a cycle Z that way. Either Z is of even length, or else by condition 2 there must be a vertex v on Z of degree at least 3. Start another walk leaving v on an edge not on Z. Again, this walk must end in a cycle Z'. Now either Z' is of even length, or otherwise it either is edge-disjoint from Z, or it shares a common part with

Z. But in the latter case, the cycle following Z and Z', leaving out the common part, is of even length.

The task to show that a graph satisfying the two conditions can be edge-colored, can now be split into three subtasks, to show how to color each type of connected component.

Claim. Every tagged component can be edge-colored.

This is shown by induction on the number of vertices in the component. The induction basis is trivial.

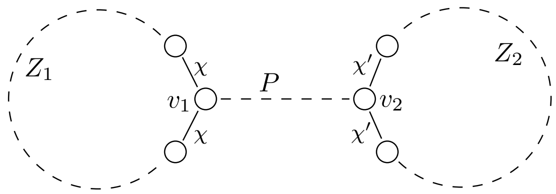
For the induction step, let a tagged component C be given, and let v be a tagged vertex in C. We modify C by deleting v and all incident edges, and by tagging all neighbors of v. The result C' is a union of several smaller tagged components, which can be colored by the induction hypothesis. This coloring can be extended to a coloring of C: if for a neighbor u of v, all incident edges in C' receive the same color, then we give the edge between u and v the other color. By induction, any tagged component can be colored.

Claim. A component C that contains an even length cycle can be edge-colored.

We color the edges around the cycle by alternating colors. For a vertex on the cycle, the incident edges other than the two cycle edges can now be colored arbitrarily. We therefore modify C by deleting the edges in the cycle, and by tagging the vertices on the cycle. The result is a union of tagged components, which can be colored by the previous case. Thus we can color all of C.

Claim. A component C that contains two edge-disjoint odd length cycles Z_1 and Z_2 can be edge-colored.

Choose vertices v_1 on Z_1 and v_2 on z_2 that are connected by a simple path P (possibly of length 0.) As in the previous claim, it suffices to color the edges on Z_1, Z_2 and P. We color the two edges on Z_1 incident with v_1 by the same color χ, and the two edges on Z_2 incident with v_2 by χ', where $\chi = \chi'$ if P is of odd length, and $\chi \neq \chi'$ otherwise.



The coloring can now be completed by coloring P and the rest of Z_1 and Z_2 by alternating colors. □

From this characterization we see that E2C $\in$ **L**, by the following algorithm: First check that condition 1 holds, which is easy. Then for every dart $d = (v, e) \in D(G)$, start a walk leaving v via e as in the above proof, until either a tagged vertex or a vertex of degree at least 3 is found, in which case the walk terminates successfully. If neither happens before the walk returns to v, then v lies on a simple cycle, thus we count the number of steps in the walk to decide whether the cycle is of even or odd length, and terminate with success or not accordingly.

By Proposition 8, we obtain the following result:

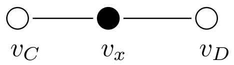
Proposition 10. mNAE-SAT(2) *is in* **L**.

We now show that the general case is in **L** as well:

Theorem 11. NAE-SAT(2) *is in* **L**.

Proof. We reduce NAE-SAT(2) to E2C. The definition of $G'(F)$ is extended to non-monotone formulas in CNF(2) by adding the clause:

– if C and D contain complementary literals x and $\bar{x}$, then we add a new vertex v_x and connect it to v_C and v_D as shown below.

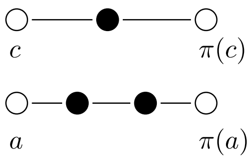


The presence of the vertex v_x enforces that the two edges get different colors, therefore $F \in$ CNF(2) is in NAE-SAT iff $G'(F)$ is in E2C. □

Proposition 12. E2C *is* **L**-*complete.*

Proof. We reduce the following problem DCA, which is **L**-complete by a result of Cook and McKenzie [9], to E2C: given a permutation π, and two points a and b, do a and b lie on the same orbit of π?

The reduction produces a graph $G(\pi)$ as follows: there are two vertices c and c' for each point c, plus two extra vertices a'' and b''. In the graph $G(\pi)$, every c other than a, b is connected to $\pi(c)$ by a path of length 2 going through c', as shown below. Similarly, a is connected to $\pi(a)$ by a path of length 3 going through a' and a'', as shown below, and analogously for b.



Note that $G(\pi)$ consists of disjoint cycles corresponding to the orbits of π. Now if a and b lie on the same orbit, then $G(\pi)$ has only even length cycles, thus is in E2C. Otherwise $G(\pi)$ has two odd cycles, thus is not in E2C. Thus the construction reduces DCA to E2C, and hence E2C is **L**-hard. We have shown E2C $\in$ **L** above, therefore E2C is **L**-complete. □

From Propositions 12 and 8 above, we get that mNAE-SAT(2) is **L**-hard, therefore also NAE-SAT(2) is **L**-hard. Together with Theorem 11, this proves the main result of this section:

Theorem 13. NAE-SAT(2) *is* **L**-*complete.*

Acknowledgments I would like to thank Albert Atserias and Jacobo Torán for helpful conversations about the subject.

References

1. Cook, S.A.: The complexity of theorem proving procedures. In: Proc. 3rd ACM Symposium on Theory of Computing. (1971) 151–158
2. Plaisted, D.A.: Complete problems in the first-order predicate calculus. Journal of Computer and System Sciences **29** (1984) 8–35
3. Jones, N.D., Lien, Y.E., Laaser, W.T.: New problems complete for nondeterministic log space. Mathematical Systems Theory **10** (1976) 1–17
4. Kleine Büning, H., Lettman, T.: Aussagenlogik: Deduktion und Algorithmen. Teubner, Stuttgart (1994)
5. Lewis, H.R., Papadimitriou, C.H.: Symmetric space-bounded computation. Theoretical Computer Science **19** (1982) 161–187
6. Porschen, S., Randerath, B., Speckenmeyer, E.: Linear time algorithms for some not-all-equal satisfiability problems. In: Proc. 6th International Conference on Theory and Applications of Satisfiability Testing, Springer LNAI (2003)
7. Braverman, M.: Witnessing SAT(2) and NAE-SAT(2) in L. Unpublished notes (November 2003)
8. Immerman, N.: Descriptive Complexity. Springer (1999)
9. Cook, S.A., McKenzie, P.: Problems complete for deterministic logarithmic space. Journal of Algorithms **8** (1987) 385–394