The Complexity of Pure Literal Elimination

Jan Johannsen[*]
Ludwig-Maximilians-Universität München

Introduction and Preliminaries

A literal a is *pure* in a CNF formula F if $\bar{a}$ does not occur in F. Pure literals can always be set to true without affecting satisfiability, which amounts to the same as to remove clauses containing them. Since this can lead to other literals becoming pure, the process needs to be iterated to obtain a satisfiability-equivalent formula without any pure literals. This process is known as pure literal elimination.

The elimination of pure literals is a common heuristic used in many satisfiability algorithms. In particular it is employed by those backtracking (DLL-type) algorithms that achieve the best worst-case upper bounds for 3-SAT [6, 12].

The current best implementations of DLL-type SAT solvers, like Chaff [10] or BerkMin [1] sacrifice this heuristic in order to gain efficiency in unit propagation. Nevertheless, pure literal elimination becomes essential again for the efficient implementation of solvers for quantified boolean formulas (QBF): it appears to be crucial (according to Letz [8]) for the performance of Semprop [7], currently one of the most efficient QBF solvers.

We will determine precisely the computational complexity of pure literal elimination for different classes of formulas. Next to the complexity class P of problems computable in polynomial time, we will consider classes defined by logarithmic space bounded algorithms employing different forms of non-determinism. Among these, the classes L of deterministic and NL of non-deterministic logarithmic space are the most familiar ones.

Less known perhaps is the class SL defined by *symmetric* non-deterministic logarithmic space [9], which lies between L and NL. Just as NL exactly captures the complexity of the reachability problem in directed graphs, SL is the

[*]Address: Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstraße 67, D-80538 München, Germany. E-mail: `jan.johannsen@ifi.lmu.de`

precise complexity of reachability in undirected graph, since this problem (UGAP) is complete for SL [5].

Our main result shows that the elimination of pure literals is *inherently sequential*; technically, this means that it is complete for P. This means that there is no hope for efficient parallel or small-space implementations of the heuristic.

For a formula F in CNF, let $\mathrm{pl}(F)$ be the formula obtained from F by deleting all clauses that contain a pure literal. Let F^* denote the least fixed point of this operation, i.e. define

$$\begin{array}{ll} F_0 := F & F_{i+1} := \mathrm{pl}(F_i) \\ F^* := F_r & \text{where } r \text{ is the least } i \text{ s.t. } F_i = F_{i+1}\,. \end{array}$$

This algorithm computes F^* in polynomial time, since the operation $\mathrm{pl}()$ is computable in logarithmic space (even in the much smaller class AC^0), and the number r of iterations is bounded by n.

The following decision problem PL is obviously equivalent to the problem of computing F^*:

> Given a formula $F = C_1 \wedge \ldots \wedge C_m$ in CNF, and $1 \leq i \leq m$, does the clause C_i occur in F^* ?

Therefore, we concentrate on the complexity of this decision problem. By the above algorithm, PL is in P. We will study the complexity of the problem PL for various classes of formulas.

For $k, \ell \in \mathbb{N}$, let k-CNF and CNF(ℓ) denote the classes of formulas in CNF having at most k literals per clause and at most ℓ occurrences of each variables, respectively. k-CNF(ℓ) denotes the class of formulas obeying both restrictions.

The complexity of the satisfiability problem for these classes is well known: it is NP-complete already for 3-CNF(3), but NL-complete for 2-CNF [5] and L-complete for CNF(2) [3]. We will completely classify the complexity of the problem PL for these formula classes.

We show that as in the case of satisfiability, the problem PL for 3-CNF(3) is already as hard as possible, in this case P-complete. For 2-CNF formulas, PL is exactly as hard as satisfiability, viz. NL-complete.

The most unexpected case, which was the starting point of this whole investigation, is that of CNF(2). It was suggested to the author several times that the algorithm showing that satisfiability for these formulas is in L [3] could be simplified by first eliminating pure literals. This way the algorithm would only need to work with ordinary graphs instead of the so-called *tagged* graphs (see definition below.) We show here that this is not an option, since removing pure literals from a CNF(2)-formula is actually more complex than testing its satisfiability: the problem PL for these formulas is SL-complete.

The general case

We first show that for general formulas in CNF, the problem is P-complete. Later we will verify that the reduction still works if the numbers of literals per clause and occurrences of variables are bounded by 3.

The following problem AGAP, the alternating graph accessibility problem, is well-known to be P-complete (cf. [2]):

> Given a directed graph $G = (V, E)$ with a partition $V = \forall \uplus \exists$, and vertices $s, t \in V$, does $\textsc{Apath}(s, t)$ hold, where the the predicate $\textsc{Apath}(x, y)$ is inductively defined by:
>
> - $x = y$, or
> - $y \in \exists$, and there is a z with $(z, y) \in E$ and $\textsc{Apath}(x, z)$, or
> - $y \in \forall$, and $\textsc{Apath}(x, z)$ holds for all z with $(z, y) \in E$.

Theorem 1. PL *is complete for P.*

Proof. As remarked above, the problem is in P. To show it is hard for P, we reduce AGAP to PL as follows:

For a given instance (G, s, t) of AGAP, we construct a formula $F(G, s)$. There is a variable y_e for every edge $e \in E$, a variable x_v for every vertex $v \in \forall$, and variables $x_v^1, \dots, x_v^k$ for every vertex $v \in \exists$ of in-degree k.

Let v be a vertex with ingoing edges $e_1, \dots, e_k$ and outgoing edges $e'_1, \dots e'_\ell$. If $v \in \forall$, then there is a clause

$$C_v = x_v \vee \bar{y}_{e'_1} \vee \dots \vee \bar{y}_{e'_\ell}$$

and for each of the edges e_j for $1 \leq j \leq k$, the clauses

$$\bar{x}_v \vee y_{e_j} \text{ and } y_{e_j} .$$

If $v \in \exists$, then there is a clause

$$C_v = x_v^1 \vee \dots \vee x_v^k \vee \bar{y}_{e'_1} \vee \dots \vee \bar{y}_{e'_\ell}$$

and for each of the edges e_j for $1 \leq j \leq k$, the clauses

$$\bar{x}_v^j \vee y_{e_j} \text{ and } y_{e_j} .$$

Additionally, the clause C_s contains a further variable z that does not occur anywhere else.

Lemma 2. *For every $v \in V$, if* $\textsc{Apath}(s,v)$ *holds, then* $C_v \notin F(G,s)^*$.

Proof. We prove by induction along the inductive definition of $\textsc{Apath}(s,v)$ that for every v with $\textsc{Apath}(s,v)$ there is an i such that $v \notin F_i$.

For $v = s$, the clause C_s does not occur in $F_1 = \text{pl}(F)$, since it contains the pure literal z.

Now let v have predecessors $u_1, \ldots, u_k$ joined to v by edges $e_j = (u_j, v)$ for $1 \leq j \leq k$.

If $v \in \forall$, and $\textsc{Apath}(s,u_j)$ holds for every j, then by the induction hypothesis there is an i_j such that $C_{u_j} \notin F_{i_j}$ for every j. Thus in F_{i_j}, the literal y_{e_j} is pure, and thus the clause $y_{e_j} \vee \bar{x}_v$ does not occur in F_{i_j+1}. Thus for $r = \max_{1 \leq j \leq k} i_j + 1$, the literal x_v is pure in F_r, and hence C_v does not occur in F_{r+1}.

Similarly, if $v \in \exists$, and $\textsc{Apath}(s,u_j)$ holds for some j, then by the induction hypothesis there is an i such that $C_{u_j} \notin F_i$. By the same reasoning as in the previous case, x_v^j is pure in F_{i+1}, and hence $C_v \notin F_{i+2}$. □

Lemma 3. *For every $v \in V$, if $C_v \notin F(G,s)^*$, then* $\textsc{Apath}(s,v)$ *holds.*

Proof. Let $C_v \notin F(G,s)^*$. We prove the claim by induction on i such that $v \in F_i \setminus F_{i+1}$. For $i = 0$, the only clause in $F_0 \setminus F_1$ is C_s, and $\textsc{Apath}(s,s)$ holds by definition, which gives the base case.

Let again v have predecessors $u_1, \ldots, u_k$ joined to v by edges $e_j = (u_j, v)$ for $1 \leq j \leq k$, and let $C_v \in F_i \setminus F_{i+1}$.

If $v \in \forall$, then x_v must be pure in F_i, since due to the unit clauses y_{f_v}, the literals $\bar{y}_{f_v}$ cannot become pure as long as C_v is present. Thus for each edge e_j, the clause $y_{e_j} \vee \bar{x}_v$ does not occur in F_i, and thus for some $i_j < i$, it is in $F_{i_j} \setminus F_{i_j+1}$. Therefore, y_{e_j} is pure in F_{i_j}, and hence $C_{u_j} \in F_{i'_j} \setminus F_{i'_j}$ for some $i'_j < i_j$. By the induction hypothesis, $\textsc{Apath}(s,u_j)$ holds for every j, and consequently $\textsc{Apath}(s,v)$ holds as well.

The case where $v \in \exists$ is similar. □

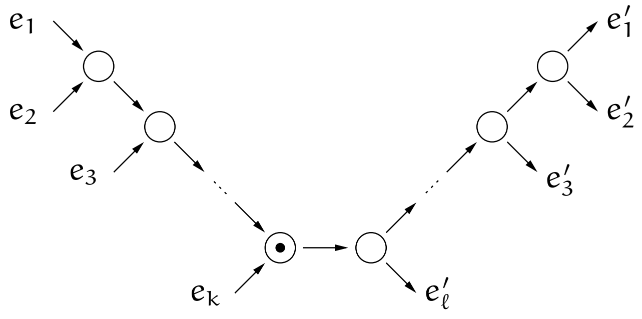
It follow that $\textsc{Apath}(s,t)$ holds iff $C_t \notin F(G,s)^*$, and thus the construction reduces AGAP to PL. □

For a vertex v in a directed graph, let the in-degree in-deg(v) denote the number of edges going into and the out-degree out-deg(v) the number of edges leaving v, so that $\deg v = \text{in-deg}\, v + \text{out-deg}\, v$. Observe that the width of the clause C_v is $1 + \text{out-deg}\, v$ for $v \in \forall$, and $\text{in-deg}\, v + \text{out-deg}\, v$ for $v \in \exists$. Also, the number of occurrences of the variables x_v for $v \in \forall$ is $1 + \text{in-deg}\, v$, and all other variables occur at most 3 times.

Thus the reduction yields a formula in 3-CNF(3) if the graph G has the following properties:

- every vertex v has $\deg v \leq 3$,
- every vertex v has $\text{in-deg}\, v \leq 2$ and $\text{out-deg}\, v \leq 2$.

It is easily verified that the problem AGAP remains complete for P for such graphs. We can reduce the general case to this special case by replacing each vertex v with ingoing edges $e_1, \ldots, e_k$ and outgoing edges $e'_1, \ldots, e'_\ell$ by a chain of $k + \ell - 2$ vertices as follows:



All the $k + \ell - 2$ new vertices are of the same type as v: if $v \in \exists$, then they all are in $\exists$, and if $v \in \forall$, they all are in $\forall$. Moreover, in the new graph the vertex s will be the $k - 1^{\text{st}}$ vertex (marked by a dot in the image above) of the chain corresponding to s in the original graph, and similarly for t.

Corollary 4. PL *for* 3-CNF(3) *formulas is complete for P.*

The case of CNF(2)

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.

From a formula $F \in \text{CNF}(2)$, 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, then v_C is tagged.

Theorem 5. PL *for formulas in* CNF(2) *is complete for SL.*

Proof. Consider a formula F. The graph $G(\mathrm{pl}(F))$ is obtained from $G(F)$ by removing the tagged vertices, and tagging the remaining vertices that used to be their neighbors. Thus, by iterating we see that $G(F^*)$ is obtained by removing all connected components from $G(F)$ that contain tagged vertices.

Therefore the following algorithm decides PL: given F and a clause C in F, loop through all tagged vertices in $G(F)$ and verify for each whether it is connected to v_{C_i}. This is a logarithmic space algorithm with an oracle for UGAP, thus PL is in L^{SL}, which is known to be the same as SL [11].

To show hardness for SL, we reduce UGAP to PL 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. Finally we add an additional variable y_s to the clause C_s. Obviously, $C_t \in F(G)^*$ if and only if t is reachable from s. □

The case of 2-CNF

RC is the following decision problem:

> Given a directed graph G and vertex s in G, is there a cycle in G reachable from s.

This problem is easily seen to be NL-complete: it is obviously in NL, and the NL-complete problem of deciding whether G contains a cycle [4] can be reduced to it by adding a new source s and edges from s to every vertex in G.

Theorem 6. PL *for* 2-CNF *formulas is NL-complete.*

Proof. We consider the same directed graph $G(F)$ that is also used in the NL-algorithm for 2-SAT. It has a vertex v_a for every literal a, and for every clause $a \vee b$, there are two edges, one from $\bar{a}$ to b and one from $\bar{b}$ to a. Moreover, for each unit clause a there is an edge from $\bar{a}$ to a.

Note that each occurrence of the complementary literal $\bar{a}$ yields an edge out of v_a, therefore the pure literals in F correspond to sinks in $G(F)$. A literal becomes a pure in some F_i if all paths starting from v_a in $G(F)$ end in a sink, i.e., no cycle is reachable from a.

An induction on i shows that this sufficient criterion is also necessary: the base case $i = 0$ is obvious, and for the induction step consider a that is pure in F_i for $i > 0$. Then all literals b occurring together with $\bar{a}$ in a clause must be pure in some F_j for $j < i$. By the induction hypothesis, every path starting from any of the vertices ν_b for these literals b ends in a sink. Since these ν_b are all the successors of ν_a, all paths starting from ν_a end in a sink as well.

Therefore, a clause C does occur in F^* iff no literal in C is pure in some F_i iff for every literal a in C, a cycle is reachable from ν_a in $G(F)$. This can be tested in nondeterministic logarithmic space, thus the problem is in NL.

To show it is NL-hard, we reduce RC to PL. To this end, we build a formula $F(G)$ from a directed graph $G = (V, E)$ and $s \in V$, where w.l.og. we assume that s is a source, as follows: There is a variable x_ν for every vertex $\nu \in V$, and for every edge $(u, \nu) \in E$ we add a clause $\bar{x}_u \vee x_\nu$. Moreover, we add a unit clause x_u for every source u in G. Thus the only pure literals in $F(G)$ are x_ν for the sinks ν in G. As above, it follows that the unit clause x_s occurs in $F(G)^*$ if and only if a cycle is reachable from s in G. □

References

[1] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Design, Automation, and Test in Europe (DATE '02)*, pages 142–149, Mar. 2002.

[2] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation*. Oxford University Press, 1995.

[3] J. Johannsen. Satisfiability problems complete for deterministic logarithmic space. In V. Diekert and M. Habib, editors, *21st International Symposium on Theoretical Aspects of Computer Science (STACS 2004)*, Springer LNCS 2996, pages 317–325, 2004.

[4] N. D. Jones. Space bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:65–85, 1975.

[5] N. D. Jones, Y. E. Lien, and W. T. Laaser. New problems complete for nondeterministic log space. *Mathematical Systems Theory*, 10:1–17, 1976.

[6] O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1–2):1–72, July 1999.

[7] R. Letz. Lemma and model caching in decision procedures for quantified boolwan formulas. In U. Egly and C. G. Fermüller, editors, *TABLEAUX 2002*, pages 160–175. Springer LNAI 2381, 2002.

[8] R. Letz. Personal communication, October 2004.

[9] H. R. Lewis and C. H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19:161–187, 1982.

[10] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[11] N. Nisan and A. Ta-Shma. Symmetric Logspace is closed under complement. *Chicago Journal of Theoretical Computer Science*, 1995.

[12] I. Schiermeyer. Pure literal lookahead: An $O(1.497^n)$ 3-satisfiability algorithm. In J. Franco, G. Gallo, H. Kleine Büning, E. Speckenmeyer, and C. Spera, editors, *Workshop on the Satisfiability Problem*. Universität zu Köln, Report No. 96-230, April–May 1996.