# Propositional Search with $k$-Clause Introduction Can be Polynomially Simulated by Resolution

Allen Van Gelder
University of California, Santa Cruz
E-mail avg@cs.ucsc.edu.

November 15, 1997

**Abstract**

Purdom proposed "complement search" as a method to introduce additional constraints into a satisfiability search, without changing whether the formula is satisfiable, with the hope of reducing the amount of search. Kullmann proposed "blocked-clause introduction", which is closely related to "complement search", with the same aim. This paper generalizes the concept of "blocked clause" to "nondecisive clause". A "nondecisive clause" has the same useful property that it can be either added to or removed from a CNF formula without changing whether the formula is satisfiable. However, the "nondecisive" property is preserved under more operations than is "blocked".

Consider a class of satisfiability algorithms that uses as its operations resolution, subsumption, and the "splitting" rule, such as the classical branching algorithm of Davis, Loveland and Logemann, and its modern variants. It is known that these algorithms can be "polynomially simulated" by resolution proofs; therefore, exponential lower bounds known for resolution apply to these algorithms.

This paper analyzes the above class of algorithms with the enhancement that nondecisive clauses (including blocked clauses) may be introduced at any search node, with the restrictions that (1) the clause does not contain variables that are new to the formula, and (2) the clause has at most $k$ literals, where $k$ is a constant. This paper shows that this enhanced class of algorithms still permits polynomial simulation by resolution, but the degree of the polynomial depends on $k$ in the simulation described.

If only restriction (1) is removed, it is already known that introduction of ternary blocked clauses suffices to produce an exponential speed-up relative to resolution. If only restriction (2) is removed, the question is open.

## Key Words

Satisfiability, resolution, polynomial simulation, blocked clause, nondecisive clause, lower bound.

# 1 Introduction

We shall consider exclusively propositional formulas in *conjunctive normal form* (CNF), also called *clause form*. Each clause is a disjunction of literals, and clauses are joined conjunctively. There has been considerable attention devoted to the establishment of lower bounds for classes of algorithms to decide the satisfiability of CNF formulas.

An exponential lower bound for resolution proofs is known [Urq87]. Several other classes of algorithms are known to obey the same lower bound (up to a polynomial factor) because they can be *polynomially simulated* by resolution proofs. DPLL [DP60, DLL62] and many of its modern variants are in this category. For still other classes of algorithms, it is known that they *cannot* be polynomially simulated by resolution proofs, and their precise complexity remains open. Extended Resolution [Tse68] is in this category.

Recently, Kullmann introduced the concept of "blocked clauses" [Kul97b], with the property that they can be added to a formula or removed from a formula without affecting its satisfiability. Purdom's proposal for "Complement Search" [Pur84], as well as Tseitin's proposal for "Extended Resolution" [Tse68] can be cast as the introduction of blocked clauses [Kul97a]. However, there is an important difference between these two "applications" of blocked-clause introduction: Extended Resolution introduces new variables, while Complement Search does not. It is known that pigeon-hole formulas can be solved in polynomial time using Extended Resolution, but not using resolution [Tse68, Urq87].

A natural question is whether the introduction of blocked clauses *without new variables* might similarly speed up techniques that can otherwise be polynomially simulated by resolution. This paper provides a partial answer. For a certain class of algorithms that includes DPLL and related search-based algorithms, the capability to add blocked clauses of at most $k$ literals, and no new variables, can be polynomially simulated by resolution proofs, but the degree of the polynomial (in the simulation described) depends on $k$.

## 1.1 Summary of Results

Consider a class of satisfiability algorithms that uses as its operations resolution, subsumption, and the "splitting" rule. This class includes DPLL, the classical branching algorithm of Davis, Loveland and Logemann [DLL62], and its modern variants. We may also include another operation from the original Davis-Putnam resolution procedure [DP60], called "elimination of an atom" (Definition 3.1). This operation chooses a variable $x$ and performs all possible resolutions with $x$ as the clashing variable, then *eliminates* all clauses containing $x$ or $\neg x$. Thus the original Davis-Putnam procedure is also included in this class. It is well known that this class can be *polynomially simulated by resolution proofs* (i.e., the number of resolution operations in the shortest resolution proof, divided by the number of operations in the satisfiability algorithm, is bounded by a polynomial in the length of the input formula).

This paper defines *nondecisive* and *extended nondecisive* clauses (Definition 2.4). An *extended* clause is one that contains variables that are not present elsewhere in the formula. Unless this qualifier is present, this paper refers to clauses in which new variables are *not* present. Nondecisive clauses are a generalization of *blocked* clauses (provided that we include the qualifier "extended", where appropriate, for both classes). As with blocked clauses, adding or removing a nondecisive clause does not change whether a formula is satisfiable, and testing whether a clause is nondecisive is syntactic. However, the "nondecisive" property is inherited under the operation of "elimination of an atom", whereas the "blocked" property is not (see Section 3).

Now suppose a new operation is included in the above class of algorithms, which introduces a *nondecisive* clause having at most $k$ literals, where $k$ is constant for the algorithm. This paper shows that this enhanced class of algorithms can still be polynomially simulated by resolution. For an input formula of $n$ variables, the resolution simulation is longer by a factor of no more than $n^k$.

In view of the fact that Extended Resolution involves the introduction of *extended* clauses with at most three literals, this paper's result suggests that new variables are the key feature needed for exponential speed-up. Additional constraints in clause form using the existing variables *appear* to confer a polynomial speed-up, at best, although the question remains open if clauses of unlimited length may be introduced.

# 2 Preliminaries

This section collects notations and definitions used throughout the paper. Standard terminology for conjunctive normal form (CNF) formulas is used. Notations are summarized in Table 1.

**Definition 2.1: (assignment, satisfaction, model)** A partial assignment is a partial function from the set of variables into $\{false, true\}$. This partial function is extended to literals, clauses, and formulas in the standard way. If the partial assignment is a total function, it is called a *total assignment*, or simply an *assignment*.

A clause or formula is *satisfied* by a partial assignment if it is mapped to *true*; A partial assignment that satisfies a formula is called a *model* of that formula. □

A partial assignment is conventionally represented by the (necessarily consistent) set of *unit clauses* that are mapped into *true* by the partial assignment. Note that this representation is a very simple formula.

**Definition 2.2: (resolution, subsumption, tautologous, useless clause)** A clause is *tautologous* if it contains complementary literals. All tautologous clauses are considered to be indistinguishable and are denoted by $\top$.

If $C = [q, \alpha]$ and $D = [\neg q, \beta]$ are two non-tautologous clauses ($\alpha$ and $\beta$ are subclauses), then

$$\mathbf{res}(q, C, D) = \mathbf{res}(q, D, C) = \mathbf{res}(\neg q, C, D) = \mathbf{res}(\neg q, D, C) = [\alpha, \beta]$$

defines the *resolution* operation, and $[\alpha, \beta]$ is called the *resolvent*, which may be tautologous. Resolution is extended to include $\top$ as an identity element:

$$\mathbf{res}(q, C, \top) = C$$

provided $C$ contains $q$ or $\neg q$.

If clause $C \subset D$, we say $C$ *properly subsumes* $D$; if $C \subseteq D$, we say $C$ *subsumes* $D$. Also, any nontautologous clause properly subsumes $\top$. Notation $D^-$ is read as "$D$, or some clause that subsumes $D$".

| | |
|---|---|
| $a, \ldots, h$ | Propositional variable. |
| $p, \ldots, z$ | Literal; i.e., propositional variable or negated propositional variable. |
| $\neg x$ | Complement of literal $x$; $\neg\neg x$ is not distinguished from $x$. |
| $\lvert x \rvert$ | The propositional variable in literal $x$; i.e., $\lvert a \rvert = \lvert \neg a \rvert = a$. |
| $A, \ldots, E$ | Clause; i.e., set of disjunctively joined literals. |
| $[p_1, \ldots, p_k]$ | Clause consisting of literals $p_1, \ldots, p_k$. |
| $[]$ | The *empty clause*, which represents *false*. |
| $\top$ | The *tautologous clause*, which represents *true*; (see Definition 2.2). |
| $\alpha, \ldots, \delta$ | Subclause, in the notation $[p, q, \alpha]$, denoting a clause with literals $p$, $q$, and possibly other literals, $\alpha$. |
| $C^-$ | Read as "$C$, or some clause that subsumes $C$". |
| $\mathcal{F}, \ldots, \mathcal{I}$ | CNF formula; i.e., set of conjunctively joined clauses; abbreviated to "formula". |
| $\{C_1, \ldots, C_k\}$ | Formula consisting of clauses $C_1, \ldots, C_k$. |
| $\mathcal{A}$, $\mathcal{B}$, $\mathcal{M}$ | Set of conjunctively joined *unit* clauses (which is a special case of *formula*). Usually, $\mathcal{A}$, $\mathcal{B}$ denote sets of *assumptions* in a search, while $\mathcal{M}$ denotes a *model* (Definition 2.1). |
| $\mathcal{A} \rightarrow C$ | Clause consisting of *complements* of the literals occurring (as unit clauses) in $\mathcal{A}$ plus literals in $C$; i.e., if $\mathcal{A} = \{p_1, \ldots, p_k\}$ and $C = [q_1, \ldots, q_m]$, then $$(\mathcal{A} \rightarrow C) = [\neg p_1, \ldots, \neg p_k, q_1, \ldots, q_m].$$ As above, $(\mathcal{A} \rightarrow C)^-$ denotes *some subset* of these literals. |
| $p$ | In a context where a unit clause is expected, $[p]$ may be abbreviated to $p$. |
| $\{p_1, \ldots, p_k\}$ | In a context where a set of unit clauses is expected, $\{[p_1], \ldots, [p_k]\}$ may be abbreviated to $\{p_1, \ldots, p_k\}$. |
| $C$, $p$ | In a context where a formula is expected, $\{C\}$ may be abbreviated to $C$ and $\{[p]\}$ may be abbreviated to $p$. |
| $+$, $-$ | Set union and difference, as infix operators, where operands are formulas, possibly using the abbreviations above. |
| $\mathbf{res}(q, C, D)$ | Resolvent of $C$ and $D$, where $q$ and $\neg q$ are the clashing literals (see Definition 2.2). |
| $C \vert \mathcal{A}$, $\mathcal{F} \vert \mathcal{A}$ | $C$ (respectively $\mathcal{F}$) *strengthened* by $\mathcal{A}$ (see Definition 2.3). |

Table 1: Summary of notations.

A clause is said to be *useless* for formula $\mathcal{F}$ if it is subsumed by a clause in $\mathcal{F}$; $\top$ is always useless. (Normally, tautologous resolvents are discarded.) $\square$

**Definition 2.3: (strengthened formula)** Let $\mathcal{A}$ be a partial assignment for formula $\mathcal{F}$. The clause $C|\mathcal{A}$, read "$C$ strengthened by $\mathcal{A}$", and the formula $\mathcal{F}|\mathcal{A}$, read "$\mathcal{F}$ strengthened by $\mathcal{A}$", are defined as follows.

1. $C|\mathcal{A} = \top$, if $C$ contains any literal that occurs in $\mathcal{A}$.

2. $C|\mathcal{A} = C - \{q \mid q \in C \text{ and } \neg q \in \mathcal{A}\}$, if $C$ does not contain any literal that occurs in $\mathcal{A}$. This may be the empty clause.

3. $\mathcal{F}|\mathcal{A} = \{C|\mathcal{A} \mid C \in \mathcal{F}\}$; i.e., apply strengthening to each clause in $\mathcal{F}$.

   Usually, occurrences of $\top$ (produced by part (1)) are deleted in $\mathcal{F}|\mathcal{A}$.

The operation $\mathcal{F}|p$ (i.e., $\mathcal{F}|\{[p]\}$) is sometimes called "unit simplification". $\square$

**Example 2.1:** Let $\mathcal{F}$ consist of $[a, b]$, $[\neg a, c]$, and $[b, d]$. Then $\mathcal{F}|a = \{[c], [b, d]\}$, and $\mathcal{F}|\{a, c\} = \{[b, d]\}$. $\square$

**Definition 2.4: (nondecisive clause, blocking literal)** A clause $B$ containing literal $\neg q$ is said to be *nondecisive* for formula $\mathcal{F}$, with *blocking literal* $\neg q$, for every clause $C \in \mathcal{F}$ that contains $q$:

1. either $\mathbf{res}(q, B, C) = \top$ (the resolvent is tautologous), or

2. there is another clause $D \in \mathcal{F}$ ($D \neq B$) such that $D|q \subseteq \mathbf{res}(q, B, C)$; that is, either $D = [\neg q, \delta]$ and $\delta \subseteq \mathbf{res}(q, B, C)$ or $D \subseteq \mathbf{res}(q, B, C)$.

Clause $B$ may or may not be in $\mathcal{F}$. In our usage, all variables in $B$ must occur in $\mathcal{F}$. An *extended nondecisive* clause satisfies all the same conditions, except that it may contain variables that do not occur in $\mathcal{F}$. $\square$

Kullmann defined a *blocked clause* to be one for which every resolvent with clashing literal $q$ was tautologous, without any condition on whether it contained variables not in $\mathcal{F}$. Thus, *extended nondecisive clause* is a generalization. Since Extended Resolution provides an adequate mechanism for introduction of new variables, we shall not be concerned further with extended nondecisive clauses in this paper.

# 3 Properties of Nondecisive Clauses

The main property of blocked clauses holds also for nondecisive clauses, with the same proof idea as used by Kullmann.

**Lemma 3.1:** Let $B = [\neg q, \beta]$ be nondecisive for $\mathcal{F}$ with blocking literal $\neg q$. If $\mathcal{F} - B$ is satisfiable, then $\mathcal{F} + B$ is satisfiable.

**Proof**: Let $\mathcal{M}$ be a total model for $\mathcal{F} - B$, but not for $\mathcal{F} + B$. Replace $q$ in $\mathcal{M}$ by $\neg q$, giving $\mathcal{M}'$. We need to check that any clause in $\mathcal{F}$ containing $q$, say $C = [q, \gamma]$, is still satisfied by $\mathcal{M}'$. Let $E = \mathbf{res}(q, B, C)$. All literals of $\beta$ are false in $\mathcal{M}$ and in $\mathcal{M}'$. By Definition 2.4, either $E$ is tautologous, or else there is some clause $D \in \mathcal{F}$ such that $D|q \subseteq E$. In either case, some literal of $\gamma$ must be true in $\mathcal{M}'$. ∎

The motivation for generalizing blocked clauses to nondecisive clauses is that the "nondecisive" property is inherited under the operation called "elimination of an atom" by Davis and Putnam [DP60], whereas the "blocked" property may not be. (In this context "atom" means "propositional variable".)

**Definition 3.1:** The operation on $\mathcal{F}$ called *Elimination of atom* $|x|$, where $x$ is a literal that occurs in $\mathcal{F}$, consists of forming all resolvents involving $x$ as clashing literal, adding these to the formula, and finally deleting all clauses containing $x$ or $\neg x$. All tautologous resolvents are deleted. (In practice, all *subsumed* resolvents may be deleted.) $\square$

**Example 3.1:** In $\mathcal{F}$ let the clauses containing $x$ and $\neg x$ consist of:

$$B = [\neg q, \neg x, \beta] \qquad D = [q, x, \delta]$$
$$C = [\neg x, \gamma] \qquad E = [\neg q, x, \alpha]$$

We assume there are no complementary literals among $\alpha$, $\beta$, $\gamma$ and $\delta$.

Suppose $B$ is a blocked clause with blocking literal $\neg q$; that is, for any $A \in \mathcal{F}$ with $q \in A$, $\mathbf{res}(q, B, A) = \top$. Now if we carry out "elimination of atom $|x|$", these clauses are derived:

$$BE = [\neg q, \alpha, \beta], \qquad CD = [q, \gamma, \delta], \qquad CE = [\neg q, \alpha, \gamma]$$

We see that $BE$ is not necessarily a blocked clause, because

$$\mathbf{res}(q, BE, CD) = [\alpha, \beta, \gamma, \delta]$$

which need not be tautologous.

Thus the "blocked" property of $B$ is not inherited by $BE$. However, we see that $BE$ is nondecisive, because this resolvent is subsumed by $CE|q$.

We thank Oliver Kullmann for suggesting the inclusion of $\neg q$ in clause $E$. This shows that an earlier proposed definition of "nondecisive", which would have required $BE$ to be subsumed by $CE$ (rather than by $CE|q$), was not general enough to be inherited.

Now consider a modified clause $D_1 = [q, \delta]$ in place of $D$, above, and assume that $B$ is nondecisive. Thus, some clause $A = [\neg q, \neg x, \beta, \delta]^-$ must be in $\mathcal{F}$. Now if we carry out "elimination of atom $|x|$", $D_1$ remains unchanged, and there is no $CD$. We now have

$$BED_1 = \mathbf{res}(q, BE, D_1) = [\alpha, \beta, \delta]$$

If $A$ does *not* contain $\neg x$, then $A|q$ subsumes $BED_1$; and if $A$ *does* contain $\neg x$, the derived clause $AE = \mathbf{res}(x, A, E)$ is present and $AE|q \subseteq BED_1$. Thus the "nondecisive" property of $B$ *is* inherited by $BE$ in this case also. $\square$

This example can be generalized into the following theorem, which is stated without proof because it is not relied upon for any of the results in this paper.

**Theorem 3.2:** Suppose $B = [\neg q, \neg x, \beta] \in \mathcal{F}$ is a nondecisive clause for $\mathcal{F}$ with blocking literal $\neg q$. Suppose we carry out "elimination of atom $|x|$", as in Definition 3.1, deriving $BE = \mathbf{res}(x, B, E)$, for some $E \in \mathcal{F}$. Then $BE$ is a nondecisive clause in the resulting formula. ▮

# 4 Unsatisfiability Algorithm Schema

We are only concerned with the performance of our algorithms on unsatisfiable formulas, so we omit the parts to deal with satisfiable formulas. The general schema is a procedure **verifyUnsat** that takes two

**verifyUnsat**$(\mathcal{F}, \mathcal{A})$

1. If $([] \in \mathcal{F})$ return.
2. Introduce nondecisive clauses $\mathcal{H}$. Although $\mathcal{H}$ is usually treated as a set, it is actually a sequence, ordered by time of introduction.

   (If this step is omitted, the algorithm is called **verifyUnsat0**.)
3. Do resolutions that do not use unit clauses as operands, inferring clauses $\mathcal{I}$. Although $\mathcal{I}$ is usually treated as a set, it is actually a sequence, ordered by time at which the clause is derived. (Unit clauses may be *derived* in $\mathcal{I}$, e.g., by resolving $[x, y]$ and $[x, \neg y]$.)
4. Choose a splitting literal $q$ (which occurs in $\mathcal{F}$).
5. Call **verifyUnsat**$((\mathcal{F} + \mathcal{H} + \mathcal{I})|\neg q, \mathcal{A} + \neg q)$. Here $\neg q$ is the new *assumption*.
6. Call **verifyUnsat**$((\mathcal{F} + \mathcal{H} + \mathcal{I})|q, \mathcal{A} + q)$. Here $q$ is the new *assumption*.
7. Return.

Figure 1: Algorithm schema analyzed. For simplicity, behavior on satisfiable formulas is undefined.

---

parameters, the formula $\mathcal{F}$, and the prior *assumptions* $\mathcal{A}$. At the top level $\mathcal{F}$ is the input formula and $\mathcal{A}$ is empty. The schema appears in Figure 1.

This schema is general enough, due to step 4, to encompass subsumption (just ignore subsumed clauses when choosing $q$), the pure literal rule (just ignore clauses containing a pure literal when choosing $q$), and the unit clause rule (always choose a unit clause for $q$, if possible). If step 2 is excluded, we call the algorithm **verifyUnsat0**.

Note that all *uses* of unit clauses are accomplished with the splitting rule (steps 4–6), and so the empty clause can only be derived in this fashion. This formalism has the advantage that all appearances of the empty clause are known to be caused by the immediately preceding assumption and are detected in step 1. The number of operations is not affected if we do not call step 1 an "operation".

We now describe how we can "annotate" **verifyUnsat0** to generate a resolution refutation. The *objective* of each invocation of **verifyUnsat0**$(\mathcal{F}, \mathcal{A})$ is to return a derivation of a clause $S_0 = (\mathcal{A} \rightarrow [])^-$ (see Table 1 for notation). That is, $S_0$ consists of the *complements* of some subset of the "assumptions" (see steps 5–6 in Figure 1) on the execution path from the root (top level call) to this call.

1. At a leaf call $([] \in \mathcal{F})$, there must be some input clause or derived clause $(\mathcal{A} \rightarrow [])^-$, so return it as $S_0$.
2. At an internal call with splitting literal $q$:
   (a) First imitate the resolutions that inferred the clauses in $\mathcal{I}$, except use the original clauses, from

the top level formula $\mathcal{F}_0$ or as derived higher in the call tree, instead of the strengthened versions. Thus, if **verifyUnsat0**$(\mathcal{F}, \mathcal{A})$ computes $E = \mathbf{res}(p, C, D)$, the resolution proof will include

$$(\mathcal{A} \rightarrow E)^- = \mathbf{res}\left(p, (\mathcal{A} \rightarrow C)^-, (\mathcal{A} \rightarrow D)^-\right).$$

(b) Now, assume each of the recursive calls achieved its objective: step 5 returned (a derivation of) $S_{00} = ((\mathcal{A} + \neg q) \rightarrow [])^-$; step 6 returned (a derivation of) $S_{01} = ((\mathcal{A} + q) \rightarrow [])^-$.

(c) If $S_{00}$ contains $q$ and $S_{01}$ contains $\neg q$, define $S_0 = \mathbf{res}(q, S_{00}, S_{01})$; in this case $S_0$ is derived by concatenating the other two derivations and this resolution. Otherwise, if $S_{00}$ does not contain $q$, define $S_0 = S_{00}$. Otherwise, $S_{01}$ does not contain $\neg q$ and define $S_0 = S_{01}$. In all cases, $S_0 = (\mathcal{A} \rightarrow [])^-$. Then annotate step 7 to return (the derivation of) $S_0$.

A point that requires attention is that the empty clause found in some leaf call, say:

$$\mathbf{verifyUnsat0}(\mathcal{F}|\mathcal{B}, \mathcal{A} + \mathcal{B})$$

might have been derived from a clause $E$ that was inferred during *this* call of **verifyUnsat0**$(\mathcal{F}, \mathcal{A})$. But then none of the variables in $E$ occur in $\mathcal{A}$. So the complements of all literals in $E$ appear in $\mathcal{B}$, and all have been resolved out (except possibly $q$ or $\neg q$) as the recursive calls return to this node.

3. At the root, $\mathcal{A}$ is empty, so $[]$ is derived.

This general idea is well known. The question is how to adapt it to nondecisive-clause introduction.

# 5 Introduction and Removal of Nondecisive Clauses

This section addresses the question of how to simulate **verifyUnsat** with resolution. The idea is to replace each introduced nondecisive clause with a sequence of resolution operations, reducing the algorithm to a "clairvoyant" version of **verifyUnsat0**.

The first step is to assume (without proof, for now), without loss of generality, that any nondecisive clause with blocking literal $\neg q$ is introduced in the node where $q$ is the splitting literal. This is because strengthening on any literal other than the blocking literal produces another nondecisive clause (or deletes it altogether).

For simplicity and concreteness in the ensuing discussion, we assume that

$$B = [\neg q, \neg x, \neg y]$$

is the introduced nondecisive clause under discussion, and that $\neg q$ is the blocking literal. We choose $B$ to be a "lowest" introduced nondecisive clause, in this sense: there are no newly introduced nondecisive clauses in the subtrees of the node where $B$ is introduced. The arguments for this ternary clause generalize in the obvious way: anything that needs to be true of both $\neg x$ and $\neg y$ needs to be true for all literals of $B$ other than $\neg q$ in the general case, etc.

**Theorem 5.1:** Let $B = [\neg q, \neg x, \neg y]$ be a "lowest" introduced nondecisive clause with blocking literal $\neg q$, as defined above. Let the notation be as described above for $\mathcal{F}$, $\mathcal{H}$, $\mathcal{I}$, etc. If $(\mathcal{F}+\mathcal{H}+\mathcal{I})|\neg q$ is unsatisfiable, then there is a resolution proof (based on $(\mathcal{F}+\mathcal{H}-B)$ as the input formula) of $[\neg x, \neg y]^-$ (i.e., some clause that subsumes $[\neg x, \neg y]$). The number of resolutions in this proof is bounded by the sum of:

- the number of clauses in $\mathcal{F}$ that contain $q$;
- the number of clauses derived in $\mathcal{I}$ at the node where $B$ was introduced;
- the number of nodes in the subtree generated by **verifyUnsat** during its proof that $(\mathcal{F}+\mathcal{H}+\mathcal{I})|\neg q$ is unsatisfiable;
- the number of clauses derived in $\mathcal{I}$ at the nodes of the same subtree. (during all invocations of step 3).

This proof is called a *side derivation for B*. ∎

Since $B$ is subsumed in $(\mathcal{F}+\mathcal{H}+\mathcal{I})|\neg q$, and $B$ is strengthened to $[\neg x, \neg y]$ in $(\mathcal{F}+\mathcal{H}+\mathcal{I})|q$, carrying out the resolution proof mentioned in Theorem 5.1 is a substitute for introducing nondecisive clause $B$. Thus, one by one, each "lowest" introduced nondecisive clause $B$ can be replaced by a sequence of resolution operations. Moreover, no intermediate clauses derived during this "side derivation" for $B$ are used again, so their number does not increase the effective size of the formula after $[\neg x, \neg y]^-$ is derived. Of course, $[\neg x, \neg y]^-$ itself may be used, but it just replaces $B|q$.

Finally, we have a "clairvoyant" run of **verifyUnsat0**. (That is, **verifyUnsat** cannot be modified to do the needed resolutions deterministically, because it is possible that both $\neg q$ and $q$ are blocking literals in different clauses introduced at the same node. Therefore, in general, it is necessary to guess which resolutions are needed before the subtree is developed.)

At any node, the resolution operations for a specific nondecisive clause need to be done only once. Therefore, if introduced clauses are restricted to contain at most $k$ literals, there are $O(n^k)$ sets of literals for which the additional resolutions need to be done at each node of the overall tree for $\mathcal{F}$. This establishes the claim of polynomial simulation.

We conjecture that polynomial simulation along the same lines is possible without the restriction to $k$ literals, but do not see a way prove it. The problems will emerge as the simulation is described, in the following sections.

## 5.1 Replacement of Nondecisive Clauses

We now sketch the construction that supports the proof of Theorem 5.1. The nondecisive clause is still assumed to be $B = [\neg q, \neg x, \neg y]$ for concreteness. Let $\mathcal{F}_0 = \mathcal{F}+\mathcal{H}-B$. Once $\neg q$ is added to the assumptions, this construction needs to avoid using any clauses of $\mathcal{F}_0$ that contain $q$ or $\neg q$, while it derives $[\neg x, \neg y]^-$.

As a preliminary step we need to ensure that every resolvent of $B$ with clashing literal $q$ is subsumed. Because $B$ is nondecisive, for each clause $E$ such that $q \in E$ and $\mathbf{res}(q, B, E)$ is *not* subsumed in $\mathcal{F}_0$, there must be some clause $D \in \mathcal{F}_0$ with $\neg q \in D$ such that $D|q \subseteq \mathbf{res}(q, B, E)$. Now derive:

$$D^* \quad = \quad \mathbf{res}(q, D, E). \tag{1}$$

We see that $D^* \subseteq \mathbf{res}(q, B, E)$. Let $\mathcal{I}^*$ be the set of $D^*$ clauses.

Now we are interested primarily in the subtree generated with $(\mathcal{F}_0 + B + \mathcal{I})|\neg q$ at the root, and specifically in nodes of this subtree for which the set of assumptions *excludes* $\neg x$ and $\neg y$. There are two main parts.

1. Construct, where necessary, sets of derived clauses $\mathcal{I}_B$ that "imitate" the corresponding sets $\mathcal{I}$, but do not use clauses of $\mathcal{F}$ that contain $q$ or $\neg q$, directly or indirectly. This part is not necessary if **verifyUnsat** does not use step 3.

2. Extract a derivation of $[\neg x, \neg y]^-$ from the search tree generated for $(\mathcal{F}_0 + B + \mathcal{I})|\neg q$ without using any clauses that contained $q$ before strengthening (those with $\neg q$ have been deleted). It is also necessary to avoid using any clauses of $\mathcal{I}$ that were derived using a clause with $q$ (see Section 5.2), unless such clause happens to be in $\mathcal{I}^*$ also (see Eq. 1). In this way, the resolution proof of $[\neg x, \neg y]$ will not depend on the strengthening by $\neg q$, although it is constructed from a tree that *is* based on that strengthening.

First, we describe this latter construction as though all sets $\mathcal{I}$ were empty. Then in Section 5.2, we describe how to construct the sets $\mathcal{I}_B$ for the search subtree, and incorporate them.

Consider the subtree generated with $(\mathcal{F}_0 + B)|\neg q$ at the root. At any node whose set of assumptions *excludes* $\neg x$ and $\neg y$, let us represent the call as

$$\mathbf{verifyUnsat}\left((\mathcal{F}_0 + B)|(\neg q + \mathcal{B}), (\mathcal{A} + \neg q + \mathcal{B})\right)$$

I.e., $\mathcal{B}$ denotes the assumptions on the path below the assumption of $\neg q$. These nodes have an (additional) *objective for* $B$, which is to return a derivation of

$$S_B = ((\mathcal{B} - x - y) \rightarrow [\neg x, \neg y])^-$$

That is, $S_B$ is a clause containing some subset of the complements of $(\mathcal{A} + \mathcal{B})$ and some subset of $\{\neg x, \neg y\}$. Notice that this clause does not contain $q$. Furthermore, its derivation must not use any clause containing $q$ or a descendant of such a clause. All nodes retain their *primary* objective to return (a derivation of) $S_0$ as described above for **verifyUnsat0**. Let us see how such a node can achieve its additional objective for $S_B$.

First, suppose $x$ is the splitting literal. Note that the branch with assumptions $(\neg q + \mathcal{B} + \neg x)$ (step 5) has no additional objective for $B$. Assume that the branch with assumptions $(\neg q + \mathcal{B} + x)$ (step 6) achieves *its* additional objective for $B$, and simply return that derivation. This suffices because $((\mathcal{B} + x) - x - y) = (\mathcal{B} - x - y)$. Splitting literal $y$ is handled correspondingly.

Second, suppose the splitting literal is some neutral literal, $p$, and (at least one of) $x$ or $y$ is *absent* from $\mathcal{B}$. (We still assume both $\neg x$ and $\neg y$ are absent from $\mathcal{B}$.) Then assume both branches achieve their additional objectives, returning (derivations of) $T_0$ and $T_1$ such that

$$\begin{aligned} T_0 &= ((\mathcal{B} - x - y + \neg p) \rightarrow [\neg x, \neg y])^- \\ T_1 &= ((\mathcal{B} - x - y + p) \rightarrow [\neg x, \neg y])^- \end{aligned}$$

Then, $S_B = \mathbf{res}(p, T_0, T_1)$ if both clauses contain the clashing literal; otherwise $S_B$ is set to a clause, $T_0$ or $T_1$, that lacks the clashing literal. Thus, $S_B = ((\mathcal{B} - x - y) \rightarrow [\neg x, \neg y])^-$ in all cases.

Third, suppose the node is a leaf. (We still assume both $\neg x$ and $\neg y$ are absent from $\mathcal{B}$.) Then some clause $E$ in $\mathcal{F}_0$ has been strengthened to the empty clause; i.e., $E = ((\neg q + \mathcal{B}) \rightarrow [])^-$. Clearly, $E$ does not contain either $x$ or $y$. If $E$ does not contain $q$, simply return $S_B = E = ((\mathcal{B} - x - y) \rightarrow [\neg x, \neg y])^-$. Otherwise, by the definition of "nondecisive", there is some clause $D \in \mathcal{F}_0$ such that $D \subseteq \mathbf{res}(q, B, E)$. If $\neg q \in D$ there is a corresponding $D^* \in \mathcal{I}^*$ (see Eq. 1). If $D^*$ is defined, then $D^*|\mathcal{B} \subseteq [\neg x, \neg y]$; otherwise, $D|\mathcal{B} \subseteq [\neg x, \neg y]$. In either case, define $S_B = D^*$ or $S_B = D$, as appropriate. Thus, $S_B = ((\mathcal{B} - x - y) \rightarrow [\neg x, \neg y])^-$, so return this clause.

Fourth, consider a "highest" node such that both $x$ and $y$ are included in the node's assumptions, $\mathcal{B}$. That is, this node's parent has assumed $x$, and $y$ was assumed earlier, or *vice versa*. At this node (and throughout its subtree), we claim that all clauses of $\mathcal{F}_0$ that contain $q$ are subsumed in $(\mathcal{F}_0 + \mathcal{I}^*)|(\neg q + \mathcal{B})$. To prove the claim, let $E = [q, \alpha] \in \mathcal{F}_0$. If $E$ contains $x$ or $y$, $E|\mathcal{B} = \top$. If not, by the definition of "nondecisive", there is some clause $D \in \mathcal{F}_0$ such that $D \subseteq \mathbf{res}(q, B, E)$. But then either $D|\mathcal{B} \subseteq E|\mathcal{B}$, or $D^*|\mathcal{B} \subseteq E|\mathcal{B}$ (see Eq. 1), establishing the claim. Therefore, this node can accomplish the *primary* objective of

$$\mathbf{verifyUnsat0}((\mathcal{F}_0 + \mathcal{I}^*)|(\neg q + \mathcal{B}), (\neg q + \mathcal{B}))$$

which is to return a derivation of $S_0 = ((\neg q + \mathcal{B}) \rightarrow [])^-$, without using any clauses of $\mathcal{F}_0$ that contained $q$. Recognizing that $q$ is absent, $S_0 = ((\mathcal{B} - x - y) \rightarrow [\neg x, \neg y])^-$, which also suffices as the value for $S_B$.

This concludes the construction for the case that all sets $\mathcal{I}$ are empty. We still need to consider the possibility that resolutions involving clashing literal $q$ create clauses in $\mathcal{I}$ in the node where $B$ is introduced, and that these derived clauses assist in the generation of the subtree for $(\mathcal{F} + \mathcal{H} + \mathcal{I})|\neg q$.

## 5.2 Resolutions that Avoid the Blocking Literal

Now we turn to the problem of imitating resolutions that are performed by **verifyUnsat** in step 3. As before, the nondecisive clause is assumed to be $B = [\neg q, \neg x, \neg y]$ for concreteness, and $\neg q$ is the blocking literal. Let us denote the formula just before $B$ is introduced as $\mathcal{F}_0$, as before, and define $\mathcal{I}^*$ as before (see Eq. 1).

The idea is to construct a "shadow" sequence of resolutions, so that each set $\mathcal{I}$ is "shadowed" by $\mathcal{I}_B$, which is derived without resolving on $B$, or any clause that contains $q$, or any clauses derived from these subsequent to $\mathcal{I}^*$, or strengthened versions of such clauses.

In each search node the shadow sequence of resolutions is denoted as $\mathcal{I}_B$, and it will contain at most the same number of resolutions as there are in $\mathcal{I}$ for the same node. The sequence of resolutions that constructs $\mathcal{I}$ can be represented as a directed acyclic graph (DAG). To construct $\mathcal{I}_B$, we first make a copy of this graph, then modify it by considering nodes in the same sequence as they were derived in $\mathcal{I}$ during step 3. By construction, no clause in $\mathcal{I}_B$ will contain $q$, nor will it be derived using $B$. Any clauses in $\mathcal{I}$ or $\mathcal{I}_B$ that contain $\neg q$ will be deleted as soon as $\neg q$ is added to the assumptions, so they will not affect that subtree.

The key to the construction is to maintain the following relationship between each clause $C \in \mathcal{I}$ and the corresponding clause $C_B \in \mathcal{I}_B$:

$$C_B | \{x, y\} \subseteq C | \{x, y\}$$

In words, after possibly removing literals $\neg x$ and $\neg y$ from $C_B$, the remaining literals are a subset of those in $C$.

First, consider the node where $B$ is introduced: **verifyUnsat**$(\mathcal{F}_0, \{\})$. (We are suppressing assumptions leading up to $\mathcal{F}_0$.) Let $\mathcal{I}_0$ be the set of clauses derived in step 3 at this node. The source nodes of the DAG for $\mathcal{I}_0$ are various clauses in $\mathcal{F}_0 + \mathcal{I}^*$ (see Eq. 1). In the following description, note that the clause names $C$, $D$, $E$, etc., have varying "local" meanings, while $B$ is always the nondecisive clause.

1. For each source node $E$ such that $q \in E$, either $x \in E$, or $y \in E$, or there is some clause $D \in \mathcal{F}$ for which $D \subseteq C = \mathbf{res}(q, B, E)$, or there is some clause $D^* \in I^*$ for which $D^* \subseteq C$. In the first two cases, the corresponding (shadow) DAG node in $\mathcal{I}_B$ is $\top$; in the third case, it is $D$, and in the fourth case $D^*$.

   For each source node in the DAG for $\mathcal{I}_0$ that does *not* contain $q$, the corresponding (shadow) DAG node in $\mathcal{I}_B$ is unchanged from $\mathcal{I}_0$.

2. Suppose $C = \mathbf{res}(p, D, E)$ is the earliest derivation that has not yet been considered, where literal $p$ is not related to $q$, $x$, or $y$. (Thus, neither $D$ nor $E$ can be clause $B$.) Let $D_B$ and $E_B$ be the clauses in $\mathcal{I}_B$ that are shadows of $D$ and $E$ in $\mathcal{I}_0$. If $C_B = \mathbf{res}(p, D_B, E_B)$ is defined, then $C_B$ is the "shadow clause" for $C$. (Recall that **res** is generalized to define $\mathbf{res}(p, D, \top) = D$ and $\mathbf{res}(p, \top, E) = E$. Thus if $E_B = \top$, then $C_B = D_B$, etc.) The resolution operation is *not* defined if the clashing literal is missing from $D_B$ or $E_B$; this case is considered below. However, when $C_B$ is defined as above, it is immediate that $C_B | \{x, y\} \subseteq C | \{x, y\}$.

   Now, suppose that $\mathbf{res}(p, D_B, E_B)$ is *not* defined, because the clashing literal is missing from $D_B$. Then just define $C_B = D_B$ and it follows that $C_B | \{x, y\} \subseteq C | \{x, y\}$. Similarly, if the clashing literal is missing from $E_B$, define $C_B = E_B$.

3. Suppose $C = \mathbf{res}(x, D, E)$ is the earliest derivation that has not yet been considered. The treatment for clashing literal $y$ is similar. Suppose for concreteness that $x \in D$ and $\neg x \in E$. (Possibly, $E = B$, in which case $E_B = B$.) As before, Let $D_B$ and $E_B$ be the clauses in $\mathcal{I}_B$ that are shadows of $D$ and $E$ in $\mathcal{I}_0$. Then define $C_B = E_B$ as the "shadow clause" for $C$. Although $C_B$ may have an "extra" $\neg x$, compared to $C$, we still have $C_B | \{x, y\} \subseteq C | \{x, y\}$.

4. Suppose $C = \mathbf{res}(q, D, E)$ is the earliest derivation that has not yet been considered, where $q \in D$. (Possibly, $E = B$.) Then define $C_B = D_B$ as the "shadow clause" for $C$. It follows that $C_B | \{x, y\} \subseteq C | \{x, y\}$.

Thus the set of clauses $\mathcal{I}_B$ has the required properties at this search node. Although $B$ remains as a source node in the DAG of $\mathcal{I}_B$, it never is an operand for resolution within $\mathcal{I}_B$. Also, as soon as $\neg q$ is assumed, $B$ is deleted, so it is not involved in any resolutions in the subtree.

The construction of $\mathcal{I}_B$ for search nodes in the subtree rooted at

$$\mathbf{verifyUnsat}\left(((\mathcal{F}_0 + B + \mathcal{I}_0)|\neg q), \{\neg q\}\right)$$

is similar, except that source nodes of the DAG for $\mathcal{I}_B$ are identified with a shadow clause of an $\mathcal{I}_B$ in some ancestor whenever the source node of the DAG for $\mathcal{I}$ was derived in that ancestor.

The procedure of Section 5.1 can be extended to allow for derived clauses in sets $\mathcal{I}$. First, **verifyUnsat** is extended to derive $\mathcal{I}_B$ in addition to $\mathcal{I}$ in all nodes where neither $\neg x$ nor $\neg y$ is among the assumptions $\mathcal{B}$, including the node where $B$ was introduced.

A derived clause $E$ in some $\mathcal{I}$ affects the search tree generated by **verifyUnsat** only when it is strengthened to become empty. If this happens in any branch where neither $\neg x$ nor $\neg y$ is among the assumptions $\mathcal{B}$, then the shadow clause $E_B$ in $\mathcal{I}_B$ *also* has been strengthened to become empty. Thus $E_B = (\mathcal{A} + \mathcal{B} \rightarrow [\neg x, \neg y])^-$. Now modify the procedure of Section 5.1 (for the case of a leaf in the search) to return $S_B = E_B$ in this case.

# 6 Conclusion

All known proposed satisfiability algorithms that introduce blocked clauses use short clauses, having 2–3 literals [Pur84, Kul97b, Kul97a]. Introduction of new variables is *excluded* in these proposals. We have shown that such introductions can be polynomially simulated by resolution. The class of algorithms covered is quite broad, including those using any combination of resolution, "elimination of an atom" [DP60], and the splitting rule [DLL62]. Pure literal elimination, unit clause rule, and subsumption are also covered.

This simulation raises the question of how effective such introductions actually are. For realistic algorithms, tight bounds are not known. Thus, proving a better upper bound by using blocked-clause introduction may mean the analysis is tighter, but the algorithm is no faster in the worst case, or it may mean the worst case really has improved. From a practical viewpoint, reduction by a polynomial factor may still be very significant.

On the other hand, if new variables may be introduced, reductions to polynomial proof length are known on certain examples, such as pigeon-hole formulas, introducing clauses having 2–3 literals. However, no algorithmic method for introducing clauses with new variables has been reported, to the best of this writer's knowledge. This seems like a fruitful direction for future research.

## Acknowledgements

This work was supported in part by NSF grants CCR-8958590 and CCR-9503830, by equipment donations from Sun Microsystems, Inc., and software donations from Quintus Computer Systems, Inc. Oliver Kullmann made helpful suggestions on an earlier draft of this manuscript.

# References

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[Kul97a] O. Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 1997. (to appear).

[Kul97b] O. Kullmann. A systematical approach to 3-SAT decision, yielding 3-SAT decision in less than $1.5045^n$ steps. *Theoretical Computer Science*, 1997. (to appear).

[Pur84] P. W. Purdom, Jr. Solving satisfiability with less searching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(4):510–513, July 1984.

[Tse68] G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Seminars in Mathematics v. 8: Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Steklov Math. Inst., Leningrad, 1968. (English trans., 1970, Plenum.).

[Urq87] A. Urquhart. Hard examples for resolution. *Journal of the Association for Computing Machinery*, 34(1):209–219, January 1987.