# Resolution and Clause Learning for Multi-Valued CNF

David Mitchell
mitchell@cs.sfu.ca

Simon Fraser University

**Abstract.** Conflict-directed clause learning (CDCL) is the basis of SAT solvers with impressive performance on many problems. CDCL with restarts (CDCL-R) has been shown to have essentially the same reasoning power as unrestricted resolution (formally, they p-Simulate each other). We show that this property generalizes to multi-valued CNF formulas. In particular, for Signed (or Multi-Valued) CNF formulas, and Regular Formulas, we show that a natural generalization of CDCL-R to these logics has essentially the same reasoning power as natural generalizations of resolution from the literature. These formulas are possible reduction targets for a number of multi-valued logics, and thus a possible basis for efficient reasoning systems for these logics.

## 1 Introduction

Multi-valued logics are among the most established formal methods for reasoning with uncertainty. In this paper, we study a property relating resolution proofs and a family of satisfiability algorithms for multi-valued CNF formulas, as defined in, for example, [8, 3].

The dominant algorithm in modern SAT solvers is the conflict-directed clause learning algorithm, introduced in [12], with restarts [10], here denoted CDCL-R. Good solvers based on CDCL-R have remarkable performance on some classes of problems. Consequently, many reasoning tasks are carried out by reduction to propositional CNF, or by adaptation of CDCL to other families of formulas. It has been shown that CDCL-R with unlimited restarts has, up to a small polynomial factor, the same reasoning power as unrestricted propositional resolution [14]. (It remains open whether restarts are essential or not.) Independently, using essentially the same method plus a probabilistic argument, it was shown in [1] that CDCL-R can refute any CNF formula with a width $k$ resolution refutation in time $O(n^{2k+2})$.

Because most SAT solvers require input in CNF, it is standard to solve propositional logic reasoning problems by reduction to CNF. Validity or satisfiability of many multi-valued and fuzzy logics (as well as annotated logics, and others) can be reduced to Signed or Multi-Valued CNF formulas [7]. The main purpose of this paper is to show that the result and method from [14] generalize naturally to the Signed CNF setting. In particular, we present a natural generalization

of CDCL-R to Signed CNF, and show that this algorithm has essentially the same reasoning power as standard binary resolution proofs for these formulas. Our proof is essentially an adaptation of that in [14], with parts influenced by [1], although our presentation is distinct. Rather than proceed from a detailed examination of CDCL-R, we proceed from the key properties of resolution, to a simplified derivation algorithm, and then to a highly abstracted version of CDCL-R.

We assume the reader is familiar with the standard CDCL-R algorithm. (A self-contained description of CDCL and its relationship to resolution can be found in [13], among other places. The reader may also want to refer to [14] and [1] for distinct presentations of the algorithm, as well as the original proofs ours is based on, and [2], where a careful examination of the relation between resolution and the implication graph method for obtaining conflict clauses appears.)

A number of algorithms and solvers for signed or multi-valued formulas, or special cases of them, have been described in the literature. We leave a review of these to a longer paper. Here we point out only that our algorithm, while presented very differently, seems essentially the same as that described in [11], but with the addition of restarts. As far as we know, restarts are essential to our result. Restarts also seem to be essential in practical SAT solvers.

The organization of the paper is as follows. In Section 2 we define Signed CNF formulas, the specific families of formulas we study, and binary resolution for these formulas. The properties of resolution proofs which are central to the proof are defined in Section 3. In Section 4 we describe an algorithm which embodies the core reasoning in (our generalization of) CDCL-R, while Section 5 shows that repeated calls to this algorithm can refute formulas as efficiently as resolution. In Section 6 we give our generalized CDCL-R algorithm, define p-simulation and give the main theorem. We conclude briefly in Section 7.

## 2 Signed CNF Formulas and Resolution

Let $D$ be a finite set of truth values and $\mathcal{P}$ a countably infinite set of multi-valued propositional atoms. Signed CNF formulas for $D$ are constructed from literals of the form $p{\in}S$, where $S$ is a non-empty proper subset of $D$ and $p \in \mathcal{P}$. (In the literature on Signed CNF, literals are usually written $S{:}p$, but we prefer the set notation for readability.) Formally $S$ is a string of constant symbols, denoting elements of $D$, enumerating the set $S$. This is typically glossed over, so that $S$ is used both for the set of truth values and the string representing it.

A clause is a disjunction of literals, and a formula is a conjunction of clauses. When convenient we identify clauses with sets of literals, and formulas with sets of clauses. An assignment $\alpha$ for formula $\Gamma$ and truth value set $D$ is a function mapping the propositional symbols in $\Gamma$ to $D$. By "assignment", we often mean partial assignment. Assignment $\alpha$ satisfies literal $p{\in}S$ if $\alpha(p)$ is in the subset of $D$ denoted by the string $S$. Definitions of satisfaction, implication, and equivalence of CNF formulas follow in the standard manner [3]. The semantics of formulas

are multi-valued, in that models are multi-valued, but the connectives in Signed CNF are classical. In complexity analyses, we will assume that $D$ is fixed.

A number of variants of this basic logic have been studied. Typical variants restrict the allowed literals, or impose structure on the truth set $D$. We consider the following three. (Unfortunately, terminology is not uniform in the literature, so our terms may correspond only roughly to those in some other papers.)

1. **Multi-Valued CNF (MV-CNF):** Formulas as just described, with the requirement that each propositional symbol $p$ occurs in at most one literal in any clause.
2. **Regular CNF over a lattice (Reg-CNF):** Let $\mathcal{D} = \langle D, \prec \rangle$ be a lattice. We call a literal $p{\in}S$ regular for $\mathcal{D}$ iff $S$ is either the upset $\uparrow i = \{s \in D \mid i \preceq s\}$, or the downset $\downarrow i = \{s \in D \mid s \preceq i\}$ of some $i$ in $D$. A formula $\Gamma$ is regular for $\mathcal{D}$ if every literal in $\Gamma$ is regular for $\mathcal{D}$.
3. **Regular CNF with Complements, over a total order (Reg-N-CNF):** Let $\mathcal{D} = \langle D, \prec \rangle$ be a set with total order. For each literal $p{\in}S$, there is some $i \in D$ for which $S$ is $\uparrow i$, $\downarrow i$, or a complement of one of these, $\overline{\uparrow i}$ or $\overline{\downarrow i}$.

For generic remarks, we use the term "signed CNF" or "signed formula", and we use one of these specific terms when remarks apply to a specific case.

*Example 1.* Consider the signed formulas with $D = \{0, 1\}$. These are equivalent to classical CNF formulas.

*Example 2.* Consider regular formulas for $\mathcal{D} = \langle D, < \rangle$, where, $D$ is the set $\{\frac{0}{s-1}, \frac{1}{s-1}, \frac{2}{s-1}, \ldots \frac{s-1}{s-1}\}$, and $<$ is the standard order on $\mathbb{Q}$. Formulas for $\mathcal{D}$ correspond to the standard multi-valued logics with finite truth value set, which were the original motivation for the study of signed CNF formulas.

The complement of a regular atom is not necessarily regular, and may not even be equivalent to any disjunction of regular atoms. Thus, the regular CNF do not allow complements of literals. We included the third case, regular CNF over a total order, with complements, because it corresponds naturally to standard multi-valued logics. However, the proof for Multi-Valued CNF captures this case, so we mention it directly only on occasion.

### 2.1 Signed Resolution

Let Signed Binary Resolution be the following derivation rule

$$\frac{p{\in}S \ \vee \ A \qquad p{\in}R \ \vee \ B}{p{\in}(S \cap R) \ \vee \ A \ \vee \ B} \tag{1}$$

We say the two antecedent (top) clauses in (1) were resolved on $p$ to produce the resolvent (bottom) clause. Two literals $p{\in}S$ and $p{\in}R$ clash if $S \neq R$. If $R{\cap}S$ is non-empty, we call the literal $p{\in}(R \cap S)$ the residue, and otherwise we say that the clash is annihilating. A pair of clashing literals which are annihilating are inconsistent.

Rule (1) is the basis of resolution proof systems for our three families, but we need two variants. In each case, a resolution derivation $\Pi$ of clause $C$ from a set $\Gamma$ of clauses is a sequence of clauses $\langle C_1, \ldots, C_s \rangle$, where each $C_i$ is either in $\Gamma$ or derived from two earlier clauses in $\Pi$ by the resolution rule, and $C_s = C$. The length of the derivation is $s$. A refutation of $\Gamma$ is a derivation of the empty clause, denoted $\Box$. A resolution rule is sound and refutation complete for a family of formulas if every formula in the family is unsatisfiable iff it has a refutation using the rule.

**Multi-Valued CNF:** We obtain a sound and refutation-complete proof system if we imbue rule (1) with implicit merging and annihilation [8]. That is:

1. Whenever $(S \cap R)$ is empty, the false "literal" $p{\in}\emptyset$ is omitted from the resolvent;
2. If two literals $p{\in}S$ and $p{\in}R$ with the same propositional symbol occur in the resolvent, they are replaced by $p{\in}(S \cup R)$.

**Regular CNF:** In this case, arbitrary literals on the same propositional symbol cannot be merged because the result may not be regular. Hence, we allow a clause to contain multiple literals on the same propositional symbol. The following restricted signed resolution rule is sound and complete for regular formulas over a lattice [4].

$$\frac{(p{\in}{\uparrow}i \ \vee \ A) \qquad (p{\in}{\downarrow}j \ \vee \ B)}{(A \vee B)} \quad \text{provided} \ \ i \not\preceq j. \tag{2}$$

**Regular CNF with Complements:** The same as given for MV-CNF.

In the restriction for Regular formulas, and also in the case of Regular formulas with negation, there are no residuals: the clashing literals are always annihilated.

If $\Gamma$ is a set of clauses and $L$ a set or sequence of literals, we may write $\Gamma, L$ as an abbreviation for $\Gamma \cup \{(l) \mid l \in L\}$. We say literal $p{\in}S$ is at least as strong as $p{\in}R$ if $S \subseteq R$, and stronger if $S \subsetneq R$. If $p{\in}S$ is at least as strong as $p{\in}R$, then also $(p{\in}S) \models (p{\in}R)$.

## 3 Empowering and Absorbed Clauses

The ability of CDCL-R to efficiently simulate resolution proofs is closely tied to a property of resolution refutations involving unit resolution. (See [14, 1] for the original versions, for classical CNF.) A unit clause is a clause with exactly one literal, and unit resolution is the use of the resolution rule when at least one antecedent is a unit clause. We write $\Gamma \stackrel{ur}{\vdash} (l)$, or simply $\Gamma \stackrel{ur}{\vdash} l$, if $(l)$ can be derived from $\Gamma$ by unit resolution alone. We write $\Gamma \stackrel{ur}{\vdash} \Box$ if there is a refutation of $\Gamma$ using only unit resolution. As in the classical case, with appropriate data structures, it is possible to check if $\Gamma \stackrel{ur}{\vdash} l$ or $\Gamma \stackrel{ur}{\vdash} \Box$ in linear time. In MV-CNF, unit resolution does not necessarily annihilate the literal of the unit clause, unlike in the classical case.

For a set or sequence of literals $L$, we denote by $\overline{L}$ the of literals which are the complements of literals in $L$. In particular, if $C$ is a clause, then $\overline{C}$ the set of complements of literals in $C$. Thus, $\Gamma, \overline{C} \stackrel{ur}{\vdash} \Box$ indicates, intuitively, that the restriction of $\Gamma$ obtained by setting all literals of $C$ false can be refuted by unit propagation. For regular formulas, this operation is always defined, since the complement of a regular literal need not be regular. For this case, we make use of the following sets. For each sequence $L = l_1, l_2, \ldots l_k$ of literals, $\underline{L}$ denotes the set of sets of literals of the form $C = (l'_1, l'_2, \ldots l'_k)$ where each $l'_i$ is inconsistent with $l_i$. In particular, for each clause $C = (l_1, l_2, \ldots l_k)$, we denote by $\underline{C}$ the set of sequences of literals of the form $L = l'_1, l'_2, \ldots l'_k$ (except order does not matter) where each $l'_i$ is inconsistent with $l_i$.

**Definition 1 (Empowering and Absorbed Clauses)** *Let $\Gamma$ be a set of clauses and $C$ a clause with $\Gamma \models C$. For sets $\Gamma$ of MV-CNF or Reg-N-CNF formulas (those with complements) we say $C$ is a-empowering for $\Gamma$ iff $C = (A \vee a)$ and*

1. $\Gamma, \overline{C} \stackrel{ur}{\vdash} \Box$,
2. $\Gamma, \overline{A} \stackrel{ur}{\nvdash} \Box$,
3. $\Gamma, \overline{A} \stackrel{ur}{\nvdash} b$, *for any literal $b$ that is inconsistent with $a$.*

*For Reg-CNF, which does not have complements, we say that $C$ is a-empowering for $\Gamma$ iff $C = (A \vee a)$ and*

1. *For some $C' \in \underline{C}$, $\Gamma, C' \stackrel{ur}{\vdash} \Box$,*
2. *If $A' \in \underline{A}$ then $\Gamma, A' \stackrel{ur}{\nvdash} \Box$,*
3. *If $A' \in \underline{A}$, and $b$ a literal inconsistent with $a$, then $\Gamma, A' \stackrel{ur}{\nvdash} b$.*

*$C$ is empowering for $\Gamma$ if it is a-empowering, for some $a \in C$, and is absorbed by $\Gamma$ otherwise.*

Intuitively, a clause $C$ is empowering for $\Gamma$ if $\Gamma, \overline{C}$ has a unit refutation, but $\Gamma, \overline{A}$ does not, where $C = (A \vee a)$. Notice that $\Gamma, \overline{C}$ is $\Gamma$ with one or more unit clauses added, in contrast to $\Gamma \cup \{\overline{C}\}$, which is $\Gamma$ with one more clause.

**Lemma 1 (Existence of Empowering Clauses).** *Let $\Gamma$ be a set of signed clauses for which $\Gamma \stackrel{ur}{\forall} \Box$, and $\Pi$ a signed resolution refutation of $\Gamma$. Then $\Pi$ contains a clause that is empowering for $\Gamma$.*

*Proof.* The first part may be expressed identically for formulas with or without complements. Let $C$ be the first clause in $\Pi$ that does not satisfy condition 1 of Definition 1. Such a clause exists, because the $\Box$ suffices if no earlier clause does. $C$ is the resolvent of two earlier clauses of $\Pi$, say $C_1 = (p{\in}S_1 \vee A_1)$, and $C_2 = (p{\in}S_2 \vee A_2)$, where $p{\in}S_1$ and $p{\in}S_2$ clash. We claim one of $C_1$ or $C_2$ is empowering for $\Gamma$. Both are logically implied by $\Gamma$, because they are in $\Pi$ and signed resolution is sound. Both satisfy condition 1 of Definition 1, by choice of C.

We complete the argument for formulas with complements as follows. Both satisfy condition 2 of Definition 1, because $C = (p{\in}(S_1 \cap S_2) \vee A_1 \vee A_2)$, so if

$\Gamma, \overline{A_1} \stackrel{ur}{\vdash} \square$ or $\Gamma, \overline{A_2} \stackrel{ur}{\vdash} \square$ then $\Gamma, \overline{C} \stackrel{ur}{\vdash} \square$, contradicting choice of $C$. Now, suppose both $C_1$ and $C_2$ fail condition 3 of Definition 1. That is, for some $R_1 \subset S_1$ and $R_2 \subset S_2$, we have $\Gamma, \overline{A_1} \stackrel{ur}{\vdash} (p{\in}R_1)$ and $\Gamma, \overline{A_2} \stackrel{ur}{\vdash} (p{\in}R_2)$. Resolving $(p{\in}R_2)$ and $(p{\in}R_1)$ produces a unit clause containing atom $p{\in}(R_1 \cap R_2)$, which is at least as strong as $p{\in}(S_1 \cap S_2)$. Then $\Gamma, \overline{C} \stackrel{ur}{\vdash} \square$, again contradicting choice of $C$. So at least one of $C_1$ or $C_2$ is empowering for $\Gamma$.

The completion for formulas without complements is the same, but messier: Both $C_1$ and $C_2$ satisfy condition 2, because $C = (p{\in}(S_1 \cap S_2) \vee A_1 \vee A_2)$, so if $\Gamma, A_1' \stackrel{ur}{\vdash} \square$ for some $A_1' \in \underline{A_1}$ or $\Gamma, A_2' \stackrel{ur}{\vdash} \square$ for some $A_2' \in \underline{A_2}$, then $\Gamma, C' \stackrel{ur}{\vdash} \square$ for some $C' \in \underline{C}$, contradicting choice of $C$. Now, suppose both $C_1$ and $C_2$ fail condition 3. That is, for some $A_1' \in \underline{A_1}$, $A_2' \in \underline{A_2}$, $R_1 \subset S_1$ and $R_2 \subset S_2$, we have that $\Gamma, A_1' \stackrel{ur}{\vdash} (p{\in}R_1)$ and $\Gamma, A_2' \stackrel{ur}{\vdash} (p{\in}R_2)$. Resolving $(p{\in}R_2)$ and $(p{\in}R_1)$ produces a unit clause containing atom $p{\in}(R_1 \cap R_2)$, which is at least as strong as $p{\in}(S_1 \cap S_2)$. Then there is a $C' \in \underline{C}$ with $\Gamma, C' \stackrel{ur}{\vdash} \square$, again contradicting choice of $C$. So either $C_1$ or $C_2$ is empowering for $\Gamma$.

## 4 Probing with Learning

The core of the CDCL algorithm can be viewed as a back-and-forth between two processes, one which guesses at partial assignments, and one which derives new clauses. We first consider an algorithm, that we call Probe-and-Learn, which embodies a "single round" of this interaction. Describing the algorithm requires some terminology.

For multi-valued formulas, the guessing involves restrictions on assignments, rather than assignments. We will call such restrictions "decision sequences".

**Definition 1.** *A decision sequence $\delta$ for $\mathcal{D}$ and $\Gamma$ is a sequence $\delta = \langle a_1, a_2, \ldots a_s \rangle$ of distinct literals such that:*

1. *If $p{\in}S$ appears in $\delta$, then $S \subseteq D$, and $p$ appears in $\Gamma$.*
2. *If $a_i$ is $p{\in}S$, then $\bar{S} \cap \bigcap\{R \mid p{\in}R = a_j \text{ and } j < i\} \neq \emptyset$. (That is, each successive literal further restricts the possible assignments.)*
3. *The set of literals in $\delta$ is satisfiable.*

*For any non-empty decision sequence $\delta = \langle l_1, \ldots l_{k-1}, l_k \rangle$, let $\delta^-$ denote the maximal proper prefix $\delta^- = \langle l_1, \ldots l_{k-1} \rangle$.*

Each decision sequence defines a set of truth assignments, namely those consistent with each literal in the sequence. For any decision sequence $\delta$ and literal $l$, we say that $\delta$ makes $l$ false if no assignment consistent with $\delta$ is consistent with $l$. $\delta$ makes $l$ true if every assignment consistent with $\delta$ satisfies $l$.

Unit propagation is a central feature in CDCL algorithms, and in particular of the "back-and-forth" process embodied in Probe-and-Learn. We will define unit propagation for multi-valued formulas, as used in our algorithm, in terms of decision sequences.

**Definition 2 (UP($\Gamma, \delta$))** *For any clause set $\Gamma$ and decision sequence $\delta$ for $\Gamma$, we denote by UP($\Gamma, \delta$) the decision sequence $\delta'$ defined by the fixpoint of the following operation:*

> *If $\Gamma$ contains a clause $C = (l \vee B)$ where $\delta$ makes every literal of $B$ false, but neither makes $l$ true or false, extend $\delta$ with $l$.*

Unit propagation corresponds to unit resolution, in a context where we are interested in collecting implied restrictions on truth assignments rather than derived clauses. In particular, $\mathrm{UP}(\Gamma, \delta)$ makes a clause of $\Gamma$ false if and only if $\Gamma \cup \delta \overset{ur}{\vdash} \square$.

A second process, closely related to unit propagation, involves derivation of clauses called asserting clauses.

**Definition 2 (Conflict Clause and Asserting Clause).** *Clause $C$ is an asserting clause for clause set $\Gamma$ and decision sequence $\delta$ iff*

1. *$\Gamma, \overline{C} \overset{ur}{\vdash} \square$, or in the case of Reg-CNF, for each $A \in \underline{C}$, $\Gamma, A \overset{ur}{\vdash} \square$;*
2. *For each literal $a \in C$, there is a literal $\beta$ at least as strong as $\overline{a}$, s.t. $\Gamma, \delta \overset{ur}{\vdash} \beta$;*
3. *For exactly one literal $a \in C$, $\Gamma, \delta^- \overset{ur}{\nvdash} \beta$, for any atom $\beta$ at least as strong as $\overline{a}$.*

*$C$ is a conflict clause for $\Gamma$ and $\delta$ if it satisfies only the first two conditions.*

The Probe-and-Learn algorithm, is presented as Algorithm 1. It takes a clause set $\Gamma$ and decision sequence $\delta$ as an argument. If unit propagation from $\delta$ satisfies $\Gamma$, the procedure returns. If unit propagation from $\delta$ makes a clause of $\Gamma$ false, we call this a conflict. If necessary, the procedure extends $\delta$ (by unspecified means) until one of these cases holds. In the case of a conflict, the handle-conflict procedure is called. Handle-conflict returns an asserting clause and an appropriate prefix of $\delta$. The newly derived clause $C$ is constructed so that $\alpha$ makes all literals but one false, so after adding it to $\Gamma$, unit propagation will set at least one more literal and satisfy this clause. It is possible that further propagation happens, and a sequence of new clauses is derived and added to $\Gamma$.

**Algorithm 1**: Probe And Learn

**Input**: Finite poset $\mathcal{D}$, signed clause set $\Gamma$ and decision sequence $\delta$.
**Output**: Clause set $\Gamma'$, and a decision sequence $\delta'$

```
α ← a minimal extension of δ s.t. either UP(Γ, α) ⊨ Γ or Γ, α ⊢^ur □ ;
while UP(Γ, α) ⊭ Γ and □ ∉ Γ and Γ, α ⊢^ur □ do
    α, C ← handle-conflict(Γ, α). ;
    Γ ← Γ ∪ {C} ;
end
return Γ, α ;
```

There is no restriction on the method by which handle-conflict can generate an asserting clause. For correctness of Probe-and-Learn, it is sufficient that, whenever the call handle-conflict$(\Gamma, \delta)$ returns $\delta', C$, then

1. $\delta'$ is a proper prefix of $\delta$;

2. $C$ is an asserting clause for $\delta', \Gamma$, as defined in Definition 2, unless it is the empty clause.
3. if $C$ is the empty clause, $\delta$ is empty.

For the p-simulation results of Section 6.1, handle-conflict() must run in polynomial time. For the concrete simulation bounds of Sections 5 and 6, it must run in linear time (as is the case in the standard implementations in SAT solvers.)

The standard "clause-learning schemes" used in CDCL SAT solvers involve a resolution derivation closely connected to the unit propagation sequence that establishes a conflict. A generalized version of this process can also be used in our multi-valued handle-conflict. In the following sub-section, we demonstrate the derivation of a particular one. (This one derives a clause analogous to the so-balled "1UIP asserting clause", which is the basis of the asserting clause derived in most CDCL SAT solvers.)

### 4.1 Asserting Clause Derivation

We need to show that, whenever Algorithm 1 calls handle-conflict, an asserting clause exists and can be constructed efficiently. For the formula with complements, this is trivial: If $\delta$ is a minimal decision sequence with $\Gamma, \delta \vdash^{ur} \Box$, then $\overline{\delta}$, the set of complements of decision literals, is an asserting clause for $S = \langle \Gamma, \delta \rangle$. For the case of regular formulas, we give a concrete construction, upon which implementation of handle-conflict can be based. This construction is the generalization of the standard method used in most CDCL SAT solvers.

Consider a decision sequence $\delta$. We may carry out unit propagation in $\Gamma$ from $\delta$, incrementally constructing extensions of $\delta$ by appending literals that unit propagation sets. Each time we observe a clause $C = (a \vee A)$ for which the current decision sequence makes every literal in $A$ false but leaves $a$ undetermined, we append $a$ to the decision sequence and call $C$ the "reason for $a$". A conflict is detected when a literal that is inconsistent with $a$ is already in the decision sequence (in other words, $C$ has been made false.) Let $\gamma$ denote the resulting decision sequence, up to but not including the conflicting literal. For each literal $l$ in $\gamma$, define the decision level of $l$ to be the size of the minimum prefix $\delta^l$ of $\delta$ such that $\Gamma, \delta^l \vdash^{ur} l$. If $l$ is in $\delta$, then its decision level is its index in $\delta$. If $l$ is set by unit propagation from $\Gamma$ only its decision level is 0.

For signed formulas, when a conflict is detected, we must have a clause $C = (A \vee l)$, with $l = p{\in}S$, in which all atoms are made false by the sequence $\delta^l$. This does not entail existence of a clause $(B \vee p{\in}R)$, where $R \cap S = \emptyset$. The reason is that resolution steps need not be annihilating (there may be a residue of the clashing literals). If a clause $C = (A \vee p{\in}R \vee l)$, involved in unit propagation, is the reason for extending the decision sequence with $l$, then some set of previously derived unit clauses on the atom $p$ annihilated $p{\in}R$. It does imply that the collection of literals on propositional symbol $p$ which appear on $\delta^l$ eliminate all values in $S$ as candidates for assignment to $p$. More precisely, among the literals in $\delta^l$ is a sub-sequence of literals $\delta_p = \langle p{\in}R_1, \ldots p{\in}R_r \rangle$ such that $(\cup R_i) \cap S = \emptyset$.

Mark each literal, of each clause, involved in the unit propagation sequence, with this sequence of literals. We begin with a clause $C_0 = C$, and generate a sequence of clauses $C_i$, by means of the following algorithm:

**Algorithm 2**: Signed-CNF 1UIP Clause Derivation

```
1 while C contains more than one literal with decision level |δ| do
2     p∈S ← the last literal in δ which clashes with a literal in C;
3     p∈R ← a literal of C marked with p∈S;
4     A ← the reason for p∈S;
5     C ← the resolvent of C and A;
6 end
7 return C
```

**Lemma 2.** *Each clause generated by Algorithm 2 is a conflict clause, and the clause returned by the algorithm is an asserting clause, according to Definition 2.*

*Proof.* Let $C_0, C_1, \ldots$ be the sequence of clauses derived by Algorithm 2. $C_0$ is trivially a conflict clause. Assume that $C_i = (p{\in}S \vee B_i)$ is a conflict clause, that $p{\in}R$ is the literal of $C_i$ identified in line 3, and that $A = (p{\in}S \vee A_i)$ is the reason for $p{\in}S$ being added to the assignment. Then $C_{i+1} = (p{\in}(S \cap R) \vee B_i \vee A_i)$ is the resolvent of $C_i$ and $A_i$. Clearly, for any $C' \in \overline{C_{i+1}}$, $\Gamma, C' \vdash^{ur} \square$, because, intuitively, setting all literals of $C_{i+1}$ false makes $A$ effectively unit, thus setting $p{\in}R$, after which we use the fact that $C_i$ is a conflict clause. Since unit propagation from $\alpha$ makes $C_i$ false and makes $A$ unit, it also makes $C_{i+1}$ false. Thus, $C_{i+1}$ is a conflict clause.

The clause returned by this algorithm is the analog of the 1UIP clause used in standard CDCL-R solvers, and we believe is the same clause as generated by the method in [11]. If we modify the termination condition of the loop in Algorithm 2 to "C contains a literal not in $\delta$", this corresponds to the DECISION learning scheme of classical CDCL. In this case the asserting clause returned contains only literals which clash with decision literals.

### 4.2 Running Time of Probe-and-Learn

**Proposition 1.** *Unit propagation and derivation of an asserting conflict clause can be carried out in time linear in $|\Gamma|$.*

**Proposition 2.** *Let $\delta$ be a decision sequence for $\Gamma$ s.t. $\Gamma \cup \delta \vdash^{ur} \square$. Then*

1. *Probe-and-Learn($\Gamma, \delta$), using a poly-time handle-conflict, runs in time polynomial in $|\Gamma|$.*
2. *Probe-and-Learn($\Gamma, \delta$), with linear-time handle-conflict, runs in time $O(|\Gamma|^2)$.*

*Proof.* Each iteration of the body of the loop performs unit propagation and executes handle-conflict, which performs the asserting clause derivation. On each iteration of the loop, except possibly the terminating iteration in the case that

a satisfying assignment is found, $\alpha$ is set to a proper prefix of its previous value. The maximum length of decision sequence is $|D| \cdot |\Gamma|$. We assume $D$ is fixed, so the total time spent in the loop is $O(|\Gamma| \cdot (\text{time for handle-conflict}))$.

## 5 Simulating Resolution with Probe-and-Learn

Here, we show that, for any resolution refutation $\Pi$ of MV-CNF or Reg-CNF formula $\Gamma$, there is a sequence of calls to probe-and-learn which refutes $\Gamma$ in time polynomial in the combined size of $\Pi$ and $\Gamma$. We begin by showing that any empowering clause can be absorbed by a sequence of calls to probe-and-learn.

First, suppose that $C = (A \vee a)$ is an $a$-empowering clause for $\Gamma$. Let $\delta$ be a decision sequence consisting of the literals of an element of $\underline{A}$, in any order, f ollowed by a literal which is inconsistent with $a$. If handle-conflict() uses the DECISION learning scheme, then Probe-and-Learn$(\Gamma, \alpha)$ extends $\Gamma$ with at least one clause which is in $\underline{\delta}$, and thus it absorbs $C$.

When probe-and-learn does not use the DECISION learning scheme, it is a little more complicated, because the clauses derived might not include *any* literals based on atoms from $C$. However, with a suitable sequence of calls we can be sure to derive clauses which make $C$ absorbed.

**Lemma 3.** *Suppose $C = (A \vee a)$ is $a$-empowering for $\Gamma$. Then there is a sequence of calls to probe-and-learn which generate an extension $\Gamma'$ of $\Gamma$ such that $C$ is absorbed by $\Gamma'$. Moreover, the total execution time for this sequence of calls is polynomial in the size of $\Gamma$, provided the time for handle-conflict is also.*

*Proof.* Let $\delta$ be a decision sequence from $\underline{A}$, followed by a literal which is inconsistent with $a$. Probe-and-learn must extend $\Gamma = \Gamma_0$ to a set $\Gamma_1$ by adding at least one derived clause, which (because it is an asserting clause) has fewer literals than $\delta$. As long as $C$ remains empowering for $\Gamma_i$, we call probe-and-learn again as follows. On each repetition, since $C$ is still $a$-helpful, no conflict is found until after $a$ is asserted. It follows that each asserting clause derived has a different asserted atom, so the number of repetitions is bounded by the product of $|\Gamma|$ and $|D|$. Moreover, eventually $a$ itself, or some atom stronger than $a$, will be the asserted atom, at which point $C$ is no longer $a$-helpful. If $C$ is still helpful, we repeat for each atom $b$ for which $C$ is still $b$-helpful, after which $C$ is no longer helpful for $\Gamma$. The entire sequence of calls requires time polynomial in $|\Gamma|$.

To see that an appropriate sequence of calls to probe-and-learn can refute $\Gamma$ in time not much longer than the size of some given refutation, we identify a sequence of empowering clauses, and absorb each.

**Lemma 4.** *Let $\Gamma$ be a set of signed clauses, and $\Pi$ a resolution refutation of $\Gamma$. Then, there is a sequence of calls to probe-and-learn, which refutes $\Gamma$ in time polynomial in the combined sizes of $\Gamma$ and $\Pi$, provided handle-conflict is polynomial time.*

*Proof.* We generate a sequence $\Gamma_0 \dots \Gamma_s$ of supersets of $\Gamma$, with $\Gamma_0 = \Gamma$, as follows. If $\Gamma_i \overset{ur}{\vdash} \square$, we are done. Otherwise, let $A$ be the first clause in $\Pi$ that is empowering for $\Gamma_i$. By Lemma 3, there is a sequence of calls to probe-and-learn which generates a superset of $\Gamma_i$ for which $A$ is not empowering. Let this be $\Gamma_{i+1}$. The total execution time is polynomial at most $|\Pi|$ times a polynomial in $|\Gamma|$.

## 6 CDCL with Restarts

The CDCL algorithm with restarts (CDCL-R) can be described in terms of a sequence of calls to Probe-And-Learn. This is illustrated by Algorithm 3.

**Algorithm 3**: Conflict-Directed Clause Learning with Restarts (CDCL-R)

**Input**: finite poset $\mathcal{D}$ and finite set $\Phi$ of signed clauses for $\mathcal{D}$.
**Output**: SAT or UNSAT

```
1  Γ ← Φ   // Clause set, initialized to the input clauses. ;
2  δ ← ⟨⟩   // Decision sequence, initialized to empty. ;
3  repeat
4      Γ, δ ← Probe-and-Learn(D, Γ, δ) ;
5      if UP(δ) ⊨ Γ then
6          return SAT
7      if δ = ⟨⟩ then
8          return UNSAT
9      if Time to Restart then
10         δ ← ⟨⟩
11 end
```

While many details have been abstracted away, Algorithm 3 is essentially the algorithm implemented by most CDCL-based solvers. It begins with the empty decision sequence. In the first call to Probe-and-Learn, the decision sequence is extended until a clause is made false, after which clause learning and back-jumping are carried out (by handle-conflict, in Probe-and-Learn). In subsequent executions of the loop body, the decision sequence resulting from the most recent handle-conflict is extended until either a satisfying assignment is produced, or Probe-and-learn again finds a conflict.

At this level of abstraction, the signed version and classical version are not distinguishable, except for the input parameter $\mathcal{D}$. We make $\mathcal{D}$ argument to Probe-and-Learn to make explicit the fact that Probe-and-Learn (and, in particular, handle-conflict), must be appropriate to $\mathcal{D}$ and the class of formulas in question. Further, if $\mathcal{D}$ is of size 2, then with appropriate choice for Probe-and-Learn, this algorithm is equivalent to the classical CDCL. Our algorithm does not allow for deletion of learned clauses, because the proof does not allow for this, but this can be trivially added.

Since CDCL-R can be viewed simply as a repeated application of Probe-and-Learn, with possible restarts, it is straightforward to see that CDCL-R can be

guided to refute a formula with a resolution refutation $\Pi$ in time polynomial in the size of $\Pi$.

**Definition 3 (Extended Decision Sequence for CDCL-R)** *An extended decision sequence for CDCL-R on input $\Gamma$, is a finite sequence of symbols satisfying:*

1. *each symbol is either a literal of $\Gamma$ or the distinguished symbol* `R`*, (for "restart"),*
2. *each maximal sub-sequence without an* `R` *is a decision sequence for $\Gamma$.*

We may take two views on extended decision sequences. On one view, we may take it as a record or witness of an actual execution of CDCL-R. On the other, we may view it as a string to control an intended execution of CDCL-R.

**Lemma 5.** *If $\Gamma$ has a resolution refutation of size $s$, then there is an execution of CDCL-R on input $\Gamma$ which refutes $\Gamma$ in time polynomial in $s$.*

*Proof.* (Sketch) The only change to the strategy for repeated Probe-and-Learn given above is that, after each Probe-and-Learn call we restart, so that Probe-and-Learn is always called with the empty decision sequence. Thus, instead of implementing the strategy for probes by setting the arguments to Probe-and-Learn, we do it by having Probe-and-Learn select (on line 1) the decision sequence specified by the strategy.

### 6.1 Proof Complexity and p-Simulation

Propositional proof complexity is the study of the relative power of proof systems for propositional logic, measured by minimum length of proofs for tautological formulas. The abstract definition of propositional proof system introduced in the seminal paper of Cook and Reckow [6], can be trivially adapted to refutation proofs for unsatisfiable signed CNF formulas (or, indeed, any co-NP complete set).

**Definition 4** *A refutation proof system for signed CNF formulas for domain $\mathcal{D}$ is a set of strings $L$ (the proofs) with a polytime function $V_L$ (a verifier for $L$) such that $V_L(x) = \Gamma$ if $x$ is an $L$-proof that $\Gamma$ is unsatisfiable, and $V_L(x) = \perp$ otherwise.*

Proof system A p-simulates proof system B if there exists a polynomial function $poly()$, such that for every unsatisfiable formula $\Gamma$ and every B-proof $\Pi_B$ of $\Gamma$, there is an A-proof $\Pi_A$ of $\Gamma$ with $|\,\Pi_A\,| \leq poly(|\,\Pi_B\,|)$.

As a simplifying convention, we require that the minimum size of a proof of $\Gamma$ is $|\Gamma|$. This is not standard in proof complexity, but is necessary for relevance to practical satisfiability algorithms, and is followed also in, e.g., [2, 5, 9, 14]. This is because a formula may be large but have a tiny proof. However, any reasonable satisfiability solver begins by reading the entire formula. Moreover, any reasonable CDCL-R-style solver begins by executing unit propagation, which may visit the entire formula.

To view a satisfiability algorithm as a proof system, we may take any trace of the algorithm on an unsatisfiable clause set $\Gamma$ as a proof of the unsatisfiability of $\Gamma$, provided that the trace reflects the running time of the algorithm, and that we can efficiently verify that the trace corresponds to an execution that reports "unsatisfiable". For present purposes, we may use extended decision sequences as CDCL-R proofs.

**Theorem 1.** *CDCL-R for MV-CNF (resp. Reg-CNF), with polytime handle-conflict, p-simulates signed resolution for MV-CNF (resp. Reg-CNF).*

*Proof.* (Sketch) To show that CDCL-R p-simulates resolution, we show that for any resolution refutation $\Pi$ of clause set $\Gamma$, there is an extended decision sequence $\delta$ s.t., when CDCL-R is executed in accordance with $\delta$ on input $\Gamma$, it runs in time polynomial in the length of $\Pi$, and reports UNSAT. This is given to us by Lemma 5.

**Corollary 1 (Pipatsrisawat & Darwiche)** *CDCL-R p-simulates resolution.*

To see this, it is enough to observe that signed formulas and resolution, when $|\mathcal{D}| = 2$, are equivalent to the classical case. We state this corollary to point out that our result is indeed a generalization of that in [14].

**Theorem 2.** *Signed resolution p-simulates CDCL-R for MV-CNF and Reg-CNF formulas, with 1UIP handle-conflict.*

*Proof.* (Sketch) Consider an execution of CDCL-R that halts and outputs "UNSAT", and let $\sigma$ the extended decision sequence corresponding to this execution. The size of $\sigma$ is certainly polynomial in the length of execution. The set of clauses produced by handle-conflict, together with the input clauses, together with the clauses generated by unit propagation when unsatisfiability is finally determined, constitute a resolution refutation. A symbol of $\sigma$ is consumed for every clause produced by handle-conflict, so the set of learned clauses at the end is of size polynomial in the size of $\sigma$. The set of unit clauses derived by unit propagation at the end is certainly polynomial in the size of the set of learned clauses plus the input set. It follows that the entire refutation is of size polynomial in $|\Gamma| + |\sigma|$.

## 7 Discussion

We have presented a natural generalization of the SAT algorithm known as CDCL with restarts to signed CNF formulas. Adapting the proofs from [14, 1] we showed that the algorithm p-simulates natural forms of binary resolution for these formulas. In particular, our proof applies to general multi-valued CNF formulas, to regular formulas when the truth value set is a lattice, and to regular formulas with complements when the truth value set has a total order. Consideration of implementation is beyond the scope of this paper, but we consider the algorithm to be effectively implementable, and thus a possible basis for practical model-finding or theorem-proving systems.

## References


1. Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res. (JAIR)*, 40:353–373, 2011.
2. Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.
3. B. Beckert, R. Hahnle, and F. Manya. The SAT problem of signed CNF formulas. In Basin ET al, editor, *Labelled Deduction*, chapter 1. Kluwer Academic Applied Logic Series, 2000.
4. Bernhard Beckert, Reiner Hähnle, and Felip Manyà. The 2-sat problem of regular signed cnf formulas. In *Proc. 30th IEEE International Symposium on Multi-Valued Logic (ISMVL 2000)*, pages 331–336, 2000.
5. Samuel R. Buss, Jan Hoffmann, and Jan Johannsen. Resolution trees with lemmas: Resolution refinements that characterize DLL algorithms with clause learning. *Logical Methods in Computer Science*, 4(4), 2008.
6. S. A. Cook and R. A. Reckhow. The relative efficiency of propositional proof systems. *J. Symbolic Logic*, 44(1):23–46, 1979.
7. Reiner Hähnle. Short conjunctive normal forms in finitely valued logics. *J. Log. Comput.*, 4(6):905–927, 1994.
8. Reiner Hähnle. Exploiting data dependencies in many-valued logics. *Journal of Applied Non-Classical Logics*, 6(1), 1996.
9. Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, and Allen Van Gelder. Clause learning can effectively p-simulate general propositional resolution. In *Proc., 23rd National Conference on Artificial Intelligence (AAAI-08), Volume 1*, pages 283–290, 2008.
10. H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies. In *Proc., 19th National Conference on Artificial Intelligence (AAAI-2002)*, pages 674–681, 2002.
11. Cong Liu, Andreas Kuehlmann, and Matthew W. Moskewicz. Cama: A multi-valued satisfiability solver. In *Proc., 2003 Int'l. Conf. on Computer Aided Design (ICCAD-2003)*, pages 326–333, 2003.
12. Joo P. Marques-Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.
13. D.G. Mitchell. A SAT solver primer. *EATCS Bulletin*, 85:112–133, February 2005. The Logic in Computer Science Column.
14. K. Pipatsrisawat and A. Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, 2011.