Backing Backtracking

Sibylle Möhle(✉) and Armin Biere

Johannes Kepler University Linz, Linz, Austria
{sibylle.moehle-rotondi,biere}@jku.at

Abstract. Non-chronological backtracking was considered an important and necessary feature of conflict-driven clause learning (CDCL). However, a SAT solver combining CDCL with chronological backtracking succeeded in the main track of the SAT Competition 2018. In that solver, multiple invariants considered crucial for CDCL were violated. In particular, decision levels of literals on the trail were not necessarily increasing anymore. The corresponding paper presented at SAT 2018 described the algorithm and provided empirical evidence of its correctness, but a formalization and proofs were missing. Our contribution is to fill this gap. We further generalize the approach, discuss implementation details, and empirically confirm its effectiveness in an independent implementation.

1 Introduction

Most state-of-the-art SAT solvers are based on the CDCL framework [8,9]. The performance gain of SAT solvers achieved in the last two decades is to some extent attributed to combining conflict-driven backjumping and learning. It enables the solver to escape regions of the search space with no solution.

Non-chronological backtracking during learning enforces the lowest decision level at which the learned clause becomes unit and then is used as a reason. While backtracking to a higher level still enables propagation of a literal in the learned clause, the resulting propagations might conflict with previous assignments. Resolving these conflicts introduces additional work which is prevented by backtracking non-chronologically to the lowest level [15].

However, in some cases a significant amount of the assignments undone is repeated later in the search [10,16], and a need for methods to save redundant work has been identified. Chronological backtracking avoids redundant work by keeping assignments which otherwise would be repeated at a later stage of the search. As our experiments show, satisfiable instances benefit most from chronological backtracking. Thus this technique should probably also be seen as a method to optimize SAT solving for satisfiable instances similar to [2,14].

The combination of chronological backtracking with CDCL is challenging since invariants classically considered crucial to CDCL cease to hold. Nonetheless, taking appropriate measures preserves the solver's correctness, and the

Supported by Austrian Science Fund (FWF) grant S11408-N23 (RiSE) and by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

M. Janota and I. Lynce (Eds.): SAT 2019, LNCS 11628, pp. 250–266, 2019.
https://doi.org/10.1007/978-3-030-24258-9_18

Chronological and non-chronological CDCL:

Trail:	The assignment trail contains neither complementary pairs of literals nor duplicates.
ConflictLower:	The assignment trail preceding the current decision level does not falsify the formula.

Non-chronological CDCL only:

Propagation:	On every decision level preceding the current decision level all unit clauses are propagated until completion.
LevelOrder:	The literals are ordered on the assignment trail in ascending order with respect to their decision level.
ConflictingClause:	At decision levels greater than zero the conflicting clause contains at least two literals with the current decision level.

Fig. 1. The CDCL invariants listed in the box are usually considered crucial to CDCL. By combining CDCL with chronological backtracking, the last three are violated.

combination of chronological backtracking and CDCL appeared to be a winning strategy: The SAT solver MAPLE_LCM_DIST_CHRONOBT [11] was ranked first in the main track of the SAT Competition 2018.

In Fig. 1 we give invariants classically considered crucial to CDCL which are relevant for the further discussion. Our aim is to demonstrate that although some of them do not hold in [10], the solving procedure remains correct.

Clearly, if upon conflict analysis the solver jumps to a decision level higher than the asserting level, invariant Propagation is violated. Measures to fix potential conflicting assignments were proposed in [10] which in addition violated invariants LevelOrder and ConflictingClause. The algorithm's correctness as well as its efficiency were empirically demonstrated. However, a formal treatment with proofs was not provided.

Our Contribution. Our main contribution consists in providing a generalization of the method presented in [10] together with a formalization. We prove that despite violating some of the invariants given above, the approach is correct. Our experiments confirm the effectiveness of chronological backtracking with an independent implementation in our SAT solver CaDiCaL [4].

2 Preliminaries

Let F be a formula over a set of variables V. A *literal* ℓ is either a variable $v \in V$ or its negation $\neg v$. The variable of ℓ is obtained by $V(\ell)$. We denote by $\bar{\ell}$ the *complement* of ℓ, i.e., $\bar{\ell} = \neg\ell$, and assume $\neg\neg\ell = \ell$. We consider formulae in

conjunctive normal form (CNF) defined as conjunctions of clauses which are disjunctions of literals. We write $C \in F$ if C is a clause in F and $\ell \in C$ for a literal ℓ occurring in C interpreting F as a set of clauses and C as a set of literals. We use set notation for formulae and clauses where convenient.

We call *trail* a sequence of literals with no duplicated variables and write $I = \ell_1 \ldots \ell_n$. We refer to an element ℓ of I by writing $\ell \in I$ interpreting I as a set of literals and denote the set of its variables by $V(I)$. Trails can be concatenated, $I = JK$, assuming $V(J) \cap V(K) = \emptyset$. We denote by $\tau(I, \ell)$ the position of the literal ℓ on the trail I. A *total assignment* is a mapping from V to the truth values 1 and 0. A trail may be interpreted as a *partial assignment* where $I(\ell) = 1$ iff $\ell \in I$. Similarly, $I(C)$ and $I(F)$ are defined.

The *residual* of the formula F under the trail I, denoted by $F|_I$, is obtained by replacing in F the literals ℓ where $V(\ell) \in I$ with their truth value. We define the residual of a clause in an analogous manner. The empty clause and the literal assigned truth value 0 are denoted by $\bot$, the empty formula by $\top$. If $I(F) = \top$, i.e., $F|_I = \top$, we say that I *satisfies* F and call I a *model* of F. If $I(C) = \bot$ for a clause $C \in F$, i.e., $C|_I = \bot$ and hence $F|_I = \bot$, we say that I *falsifies* C (and therefore F) and call C the *conflicting clause*.

We call *unit clause* a clause $\{\ell\}$ containing one single literal ℓ which we refer to as *unit literal*. We denote by $\mathsf{units}(F)$ the sequence of unit literals in F and extend this notion to the residual of F under I by writing $\mathsf{units}(F|_I)$. We write $\ell \in \mathsf{units}(F|_I)$ for referring to the unit literal ℓ in the residual of F under I.

3 Generalizing CDCL with Chronological Backtracking

In classical CDCL SAT solvers based on **non-chronological** backtracking [8] the trail reflects the order in which literals are assigned. The trail is used during conflict analysis to simplify traversal of the implication graph in reverse assignment order and in general during backtracking to undo assignments in the proper reverse assignment order.

In CDCL with non-chronological backtracking, the trail is partitioned into subsequences of literals between decisions in which all literals have the same decision level. Each subsequence starts with a decision literal and extends until the last literal before the next decision. Literals assigned before any decision may form an additional subsequence at decision level zero.

After adding **chronological** backtracking to CDCL as described in [10], the trail is not partitioned in the same way but subsequences of the same decision level are interleaved, while still respecting the assignment order.

Let $\delta \colon V \mapsto \mathbb{N} \cup \{\infty\}$ return the decision level of a variable v in the set of variables V, with $\delta(v) = \infty$ if v is unassigned. This function is updated whenever a variable is either assigned or unassigned. The function δ is extended to literals ℓ, clauses C and trails I by defining $\delta(\ell) = \delta(V(\ell))$, $\delta(C) = \mathsf{max}\{\delta(\ell) \mid \ell \in C\}$ for $C \neq \bot$, and $\delta(I) = \mathsf{max}\{\delta(\ell) \mid \ell \in I\}$. We further define $\delta(\bot) = 0$.

Given a set of literals L, we denote by $\delta(L) = \{\delta(\ell) \mid \ell \in L\}$ the set containing the decision levels of its elements. The function δ updated with decision level d

assigned to $V(\ell)$ is denoted by $\delta[\ell \mapsto d]$. Similarly, $\delta[I \mapsto \infty]$ represents the function δ where all literals on the trail I are unassigned. In the same manner, $\delta[V \mapsto \infty]$ assigns all variables in V to decision level ∞. We may write $\delta \equiv \infty$ as a shortcut. The function δ is left-associative. We write $\delta[L \mapsto \infty][\ell \mapsto b]$ to express that the function δ is updated by first unassigning all literals in a sequence of literals L and then assigning literal ℓ to decision level b.

For the sake of readability, we write $J \leqslant I$ where J is a subsequence of I and the elements in J have the same order as in I and $J < I$ when furthermore $J \neq I$. We denote by $I_{\leqslant b}$ the subsequence of I containing exactly the literals ℓ where $\delta(\ell) \leqslant b$.

Due to the interleaved trail structure we need to define decision literals differently than in CDCL. We refer to the set consisting of all decision literals on I by writing $\mathsf{decs}(I)$ and define a *decision literal* ℓ as

$$\ell \in \mathsf{decs}(I) \quad \text{iff} \quad \ell \in I,\ \delta(\ell) > 0,\ \forall k \in I\ .\ \tau(I,k) < \tau(I,\ell) \Rightarrow \delta(k) < \delta(\ell) \quad (1)$$

Thus, the decision level of a decision literal $\ell \in I$ is strictly higher than the decision level of any literal preceding it on I. If $C|_I = \{\ell\}$ for a literal ℓ, then ℓ is not a decision literal. The set $\mathsf{decs}(I)$ can be restricted to decision literals with decision level lower or equal to i by writing $\mathsf{decs}_{\leqslant i}(I) = \mathsf{decs}(I_{\leqslant i})$.

As in [13] we use an abstract representation of the assignment trail I by writing $I = I_0 \ell_1 I_1 \ldots \ell_n I_n$ where $\{\ell_1, \ldots, \ell_n\} = \mathsf{decs}(I)$. We denote by $\mathsf{slice}(I, i)$ the i-th *slice* of I, i.e., the subsequence of I containing all literals ℓ with the same decision level $\delta(\ell) = i$. The i-th *block*, denoted by $\mathsf{block}(I, i)$, is defined as the subsequence of I starting with the decision literal with decision level i and extending until the last literal before the next decision:

$$\mathsf{slice}(I, i) = I_{=i}$$
$$\mathsf{block}(I, i) = \ell_i I_i$$

Note that in general $I_{=i} \neq I_i$, since I_i (due to the interleaved structure of the trail I) may contain literals with different decision levels, while this is not the case in $I_{=i}$. In particular, there might be literals with a lower decision level than some literal preceding them on the trail. We call these literals *out-of-order literals*. Contrarily to classical CDCL with non-chronological backtracking, upon *backtracking* to a decision level b, blocks must not be discarded as a whole, but only the literals in $\mathsf{slice}(I, i)$ where $i > b$ need to be unassigned.

Consider the trail I on the left hand side of Fig. 2 over variables $\{1, \ldots, 5\}$ (in DIMACS format) where τ represents the position of a literal on I and δ represents its decision level:

Literals 1 and 3 were propagated at decision level zero, literal 5 was propagated at decision level one. The literals 3 and 5 are out-of-order literals: We have $\delta(2) = 1 > 0 = \delta(3)$, whereas $\tau(I, 2) = 1 < 2 = \tau(I, 3)$. In a similar manner, $\delta(4) = 2 > 1 = \delta(5)$, and $\tau(I, 4) = 3 < 4 = \tau(I, 5)$. Moreover, $I_{\leqslant 1} = 1\,2\,3\,5$, $\mathsf{decs}(I) = 2\,4$, $\mathsf{decs}_{\leqslant 1}(I) = 2$, $\mathsf{slice}(I, 1) = 2\,5$, and $\mathsf{block}(I, 1) = 2\,3$.

Upon backtracking to decision level one, the literals in $\mathsf{slice}(I, 2)$ are unassigned. The resulting trail is visualized in the middle of Fig. 2. Note that since

τ	0	1	**2**	3	**4**
I	1	2	**3**	4	**5**
δ	0	1	**0**	2	**1**

τ	0	1	**2**	3
I	1	2	**3**	5
δ	0	1	**0**	1

τ	0	1
I	1	3
δ	0	0

Fig. 2. In the trail I on the left, from the three trails shown, literals 3 and 5 are placed out of order. In fact, their decision level δ is lower than the decision level of a literal preceding them on the trail, i.e., with lower position τ. The trails in the middle and on the right hand side show the results of backtracking to decision levels 1 and 0. When backtracking to the *backtrack level* b, only literals ℓ with $\delta(\ell) > b$ are removed from the trail, while the assignment order is preserved.

the assignment order is preserved, the trail still contains one out-of-order literal, namely 3. Backtracking to decision level zero unassigns all literals in $\mathsf{slice}(I, 2)$ and $\mathsf{slice}(I, 1)$ resulting in the trail in which all literals are placed in order depicted on the right hand side.

4 Calculus

We devise our calculus as a transition system over a set of states S, a transition relation $\rightsquigarrow \subseteq S \times S$ and an initial state s_0. Non-terminal states are described by $(F,\ I,\ \delta)$ where F denotes a formula over variables V, I denotes the current trail and δ refers to the decision level function.

The *initial* state is given by $s_0 = (F,\ \varepsilon,\ \delta_0)$. In this context, F is the original formula, ε denotes the empty trail and $\delta_0 \equiv \infty$. The *terminal* state is either SAT or UNSAT expressing satisfiability or unsatisfiability of F. The transition relation $\rightsquigarrow$ is defined as the union of transition relations $\rightsquigarrow_{\mathsf{R}}$ where R is either True, False, Unit, Jump or Decide. These rules are listed in Fig. 3. We first explain the intuition behind these rules before proving correctness in Sect. 5:

True / False. If $F|_I = \top$, F is satisfiable and the search terminates in the state SAT (rule True). If $F|_I = \bot$, a clause $C \in F$ exists where $I(C) = \bot$. The *conflict level* is $\delta(C) = 0$. Obviously $I_{\leqslant 0}(F) = \bot$ and consequently F is unsatisfiable. Then the procedure terminates in state UNSAT (False).

Unit. Propagated unit literals are assigned the maximum decision level of their reason which may be lower than the current decision level. Requiring that the residual of F under I is conflict-free ensures that invariant ConflictLower holds.

Jump. We have $F|_I = \bot$, i.e., there exists a clause $C \in F$ for which we have $I(C) = \bot$. Since the conflict level is $\delta(C) = c > 0$, there is a decision left on I. We assume to obtain a clause D implied by F (usually through conflict analysis) with $\delta(D) = c > 0$ whose residual is unit, e.g., $\{\ell\}$, at *jump level* $j = \delta(D \setminus \{\ell\})$, the second highest decision level in D. In fact, the residual of D under the trail is unit at any backtrack level b where $j \leqslant b < c \leqslant d$, with $d = \delta(I)$ denoting the current decision level. Using D as a reason, we may backtrack to any of these decision levels. Remember that the decision levels on the trail do not have

True: $(F,\ I,\ \delta) \rightsquigarrow_{\mathsf{True}} \mathsf{SAT}$ if $F|_I = \top$

False: $(F,\ I,\ \delta) \rightsquigarrow_{\mathsf{False}} \mathsf{UNSAT}$ if exists $C \in F$ with $C|_I = \bot$ and $\delta(C) = 0$

Unit: $(F,\ I,\ \delta) \rightsquigarrow_{\mathsf{Unit}} (F,\ I\ell,\ \delta[\ell \mapsto a])$ if $F|_I \neq \top$ and $\bot \notin F|_I$ and exists $C \in F$ with $\{\ell\} = C|_I$ and $a = \delta(C \setminus \{\ell\})$

Jump: $(F,\ I,\ \delta) \rightsquigarrow_{\mathsf{Jump}} (F \wedge D,\ PK\ell,\ \delta[L \mapsto \infty][\ell \mapsto j])$ if exists $C \in F$ with $PQ = I$ and $C|_I = \bot$ such that $c = \delta(C) = \delta(D) > 0$ and $\ell \in D$ and $\ell|_Q = \bot$ and $F \models D$ and $j = \delta(D \setminus \{\ell\})$ and $b = \delta(P)$ and $j \leqslant b < c$ and $K = Q_{\leqslant b}$ and $L = Q_{>b}$

Decide: $(F,\ I,\ \delta) \rightsquigarrow_{\mathsf{Decide}} (F,\ I\ell,\ \delta[\ell \mapsto d])$ if $F|_I \neq \top$ and $\bot \notin F|_I$ and $\mathsf{units}(F|_I) = \emptyset$ and $V(\ell) \in V$ and $\delta(\ell) = \infty$ and $d = \delta(I) + 1$

Fig. 3. In the transition system of our framework non-terminal states $(F,\ I,\ \delta)$ consist of a CNF formula F, the current trail I and the decision level function δ. The rules formalize termination (True and False), backtracking (Jump), unit propagation (Unit) and picking decisions (Decide).

to be sorted in ascending order and that upon backtracking only the literals in the i-th slice with $i > b$ need to be unassigned as discussed in Sect. 3. After backtracking we propagate ℓ and assign it decision level j to obtain $\delta(PK\ell) = b$.

Note. If the conflicting clause C contains exactly one literal ℓ assigned at conflict level c, its residual is unit at decision level $c - 1$. The solver therefore could backtrack to decision level $c-1$ and propagate ℓ. An optimization is to use $D = C$ as reason saving the computational effort of conflict analysis. This corresponds to learning C instead of a new clause D and is a special case of rule Jump. It is also explicitly included in the pseudocode in [10].

Decide. If F is neither satisfied nor falsified and there are no unit literals in $F|_I$, an unassigned variable is assigned. Invariants ConflictLower and Propagation hold.

Example. As pointed out above, the conflicting clause C may contain one single literal ℓ assigned at decision level c. While according to the pseudocode in [10] backtracking is executed to the second highest decision level j in C, the implementation MAPLE_LCM_DIST_CHRONOBT backtracks chronologically to decision level $c - 1$, which may be higher than j. We adopt this strategy in our own solver, but unlike MAPLE_LCM_DIST_CHRONOBT we eagerly propagate ℓ and assign it decision level j, as described in the explanation of rule Jump above.

The authors of [10] focused on unit propagation and backtracking, and an in-depth discussion of the case in which the conflicting clause contains exactly

one literal at conflict level is missing. We fill this gap and explain our calculus in detail by means of an example for this case. We generated this example with our model-based tester Mobical for CaDiCaL based on ideas in [1,12]. Our example is larger and provides a good intuition regarding the occurrence of multiple nested decision levels on the trail as well as its effect on backtracking.

We represent variables by natural numbers and consider a formula F over variables $\{1, \ldots, 48\}$ as above where negative numbers encode negated variables. We further use set notation for representing clauses. Consider the following assignment trail excerpt where the trail I is represented as a sequence of literals, τ denotes the position of a literal on I and δ its decision level:

τ	$\cdots$	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
I	$\cdots$	4	5	30	47	15	18	6	-7	-8	45	9	38	-23	17	44	-16
δ	$\cdots$	3	4	4	4	4	4	5	5	5	5	6	6	6	6	6	6
C		{			-47,										**-17,**	**-44**	}
D		{		-30,	-47,		-18,							**23**			}

Initially, the literals are placed in order on I, i.e., they are sorted in ascending order with respect to their decision level. At this point, a conflict occurs. The conflicting clause is $C = \{-47, -17, -44\}$ depicted below the trail containing two literals at conflict level $c = \delta(C) = 6$, i.e., -17 and -44 depicted in boldface in the above outline. Conflict analysis in our implementation learned the clause $D = \{-30, -47, -18, 23\}$ where $\delta(-30) = \delta(-47) = \delta(-18) = 4$ and $\delta(23) = 6$. Since $\delta(D) = c = 6$ and $j = \delta(D \setminus \{23\}) = 4$, the solver in principle would be free to backtrack to either decision level 4 or 5.

Let the solver backtrack *chronologically* to decision level 5 where D becomes unit, specifically $\{23\}$. The position on the trail the solver backtracks to is marked with a vertical dotted line. Accordingly all literals with decision level higher than 5 are removed from I (literals at positions higher than 13). Then literal 23 is propagated. The jump level is $j = 4$, hence literal 23 is assigned decision level 4 out of order. Literal -38 is propagated due to $\mathsf{reason}(-38) = \{-15, -23, -38\}$ (not shown). Since $\delta(-15) = \delta(-23) = 4$, literal -38 is assigned decision level 4. Then literal -9 is propagated with $\mathsf{reason}(-9) = \{-45, 38, -9\}$ with $\delta(-45) = 5$ and $\delta(38) = 4$. Thus, -9 is assigned decision level 5. The resulting trail is

τ	$\cdots$	4	5	6	7	8	9	10	11	12	13	**14**	**15**	16
I	$\cdots$	4	5	30	47	15	18	6	-7	-8	45	**23**	**-38**	-9
δ	$\cdots$	3	4	4	4	4	4	5	5	5	5	**4**	**4**	5

where the literals 23 and -38 (depicted in boldface) are placed out of order on I. Later in the search we might have the following situation:

τ	$\cdots$	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
I	$\cdots$	18	6	-7	-8	45	23	-38	-9	10	-11	13	16	-17	-25	42	12	-41
δ	$\cdots$	4	5	5	5	5	4	4	5	6	7	5	4	4	4	4	5	5
C		{												17,		-42,	**-12**	}

The first assignment after analyzing the last conflict is placed right after the dashed vertical line. Again, a conflict occurs. Let $C = \{17, -42, -12\}$ be the conflicting clause. The conflict level is $\delta(C) = 5$ and the decision level of I is $\delta(I) = 7$. Clause C contains exactly one literal at conflict level, namely -12 depicted in boldface. The solver backtracks to decision level $c - 1 = 4$ marked with a thick solid line. After removing from I all literals with decision level higher than 4 and propagating literal -12, the resulting trail is

τ	$\cdots$	9	10	11	12	13	14	15	16
I	$\cdots$	18	23	-38	16	-17	-25	42	-12
δ	$\cdots$	4	4	4	4	4	4	4	4

Note that as discussed above we use $D = C \in F$ without actually adding it.

5 Proofs

For proving the correctness of our method, we first show that the system terminates in the correct state which can be done in a straightforward manner. Proving that the system always makes progress is more involved. Last we prove that our procedure terminates by showing that no infinite sequence of states is generated. By $\delta(\mathsf{decs}(I))$ we denote the set consisting of the decision levels of the set of decision literals on I. We start by proving the following invariants:

(1) $\forall k, \ell \in \mathsf{decs}(I) \,.\, \tau(I, k) < \tau(I, \ell) \implies \delta(k) < \delta(\ell)$
(2) $\delta(\mathsf{decs}(I)) = \{1, \ldots, \delta(I)\}$
(3) $\forall n \in \mathbb{N} \,.\, F \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$

Lemma 1 (Invariants). *Invariants* (1) – (3) *hold in non-terminal states.*

Proof. The proof is carried out by induction over the number of rule applications. We assume Inv. (1) – (3) hold in a non-terminal state $(F,\ I,\ \delta)$ and show that they are still met after the transition to another non-terminal state for all rules.

Unit: The trail I is extended by a literal ℓ. We need to show that ℓ is not a decision literal. To this end it is sufficient to consider the case where $a > 0$. There exists a clause $C \in F$ with $\{\ell\} = C|_I$. Since $a = \delta(C \setminus \{\ell\})$, there exists a literal $k \in C$ where $k \neq \ell$ and such that $\delta(k) = a$. Obviously, k was assigned prior to ℓ and $\tau(I, k) < \tau(I, \ell)$. Since $\delta(k) = \delta(\ell)$ and by the definition of decision literal in Eq. (1), ℓ is not a decision literal. The decisions remain unchanged, and Inv. (1) and (2) hold after executing rule Unit.

We have $F \wedge \mathsf{decs}_{\leqslant n}(I) \models \overline{C \setminus \{\ell\}}$ and $F \wedge \mathsf{decs}_{\leqslant n}(I) \models C$, therefore, by modus ponens we get $F \wedge \mathsf{decs}_{\leqslant n}(I) \models \ell$. Since ℓ is not a decision literal, as shown above, $F \wedge \mathsf{decs}_{\leqslant n}(I\ell) \equiv F \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. Hence, $F \wedge \mathsf{decs}_{\leqslant n}(I\ell) \models (I\ell)_{\leqslant n}$, and Inv. (3) holds after executing rule Unit.

Jump: We first show that K contains no decision literal. In fact, the trail I is of the form $I = PQ$, and K is obtained from Q by removing all literals with decision level greater than b. In particular, the order of the remaining (decision) literals remains unaffected, and Inv. (1) still holds. We further have $\delta(K) \leqslant \delta(P) = b$. Since $\forall p \in P, k \in K \;.\; \tau(PK, p) < \tau(PK, k)$ and by the definition of decision literals in Eq. (1), the decision literal with decision level b is contained in P. Therefore, since K contains no (decision) literal with decision level greater than b, it contains no decision literal.

Now we show that ℓ is not a decision literal either. As in the proof for rule Unit, it is sufficient to consider the case where $j > 0$. There exists a clause D where $F \models D$ such that $\delta(D) = c$ and a literal $\ell \in D$ for which $\ell|_Q = \bot$ and $\ell \in Q$. Since $j = \delta(D \setminus \{\ell\})$, $\delta(\ell) = \delta(D) = c > b$, and $\ell \notin K$. Instead, $\ell \in L$, and ℓ is unassigned during backtracking to any decision level smaller than c, i.e., $\ell \notin PK$. Furthermore, there exists a literal $k \in D$ where $k \neq \ell$ and such that $\delta(k) = j$ which precedes ℓ on the trail $PK\ell$. Therefore, following the argument in rule Unit, literal ℓ is not a decision literal, and since the decisions remain unchanged, Inv. (1) and (2) hold after applying rule Jump.

Invariant (3) holds prior to applying rule Jump, i.e., $F \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. We have that $F \models D$, and therefore $F \wedge D \equiv F$. Since $I = PQ$, $PK < I$ and obviously $F \wedge \mathsf{decs}_{\leqslant n}(PK) \implies (PK)_{\leqslant n}$. From $j = \delta(D \setminus \{\ell\})$ we get $D|_{PK} = \{\ell\}$. Repeating the argument in the proof for rule Unit by replacing I by PK and C by D, we have that $F \wedge \mathsf{decs}_{\leqslant n}(PQ\ell) \models (PQ\ell)_{\leqslant n}$, and Inv. (3) is met after executing rule Jump.

Decide: Literal ℓ is a decision literal by the definition of a decision literal in Eq. 1: It is assigned decision level $d = \delta(I) + 1$, and $\forall k \in I \;.\; \delta(k) < \delta(\ell)$. Further, $\forall k \in I\ell \;.\; k \neq \ell \implies \tau(I\ell, k) < \tau(I\ell, \ell)$. Since $\ell \in \mathsf{decs}(I\ell)$, we have $\delta(\mathsf{decs}(I\ell)) = \{1, \ldots, d\}$, and Inv. (1) and (2) hold after applying rule Decide.

Since ℓ is a decision, $F \wedge \mathsf{decs}_{\leqslant n}(I\ell) \equiv F \wedge \mathsf{decs}_{\leqslant n}(I) \wedge \ell_{\leqslant n}$ and since Inv. (3) holds prior to applying Decide, obviously $F \wedge \mathsf{decs}_{\leqslant}(I\ell) \models I_{\leqslant n} \wedge \ell_{\leqslant n} \equiv (I\ell)_{\leqslant n}$, and Inv. (3) is met after applying rule Decide. □

Proposition 1 (Correctness of Terminal State). SEARCH *terminates in the correct state, i.e., if the terminal state is* SAT*, then F is satisfiable, and if the terminal state is* UNSAT*, then F is unsatisfiable.*

Proof. We show that the terminal state is correct for all terminal states.

SAT: We must show that an unsatisfiable formula can not be turned into a satisfiable one by any of the transition rules. Only equivalence-preserving transformations are executed: Rules Unit and Decide do not affect F, and in rule Jump a clause implied by F is added. Therefore, if the system terminates in state SAT, F is indeed satisfiable.

UNSAT: It must be proven that a satisfiable formula can not be made unsatisfiable. Only equivalence-preserving transformations are executed. Rules Unit and Decide do not affect F, and in rule Jump a clause implied by F is added. We need to show that if rule False is applied, the formula F is unsatisfiable. We have to consider Inv. (3) for $n = 0$. There exists a clause $C \in F$ such that $I_{\leqslant 0}(C) = \bot$, which leads to $F \wedge \mathsf{decs}_{\leqslant 0}(F) \equiv F \models I_{\leqslant 0}(C) = \bot$.

□

Proposition 2 (Progress and Termination). SEARCH *makes progress in non-terminal states (a rule is applicable) and always reaches a terminal state.*

Proof. We first prove progress by showing that in every non-terminal state a transition rule is applicable. Then we prove termination by showing that no infinite state sequence is generated.

Progress: We show that in every non-terminal state a transition rule is applicable. The proof is by induction over the number of rule applications. Assume we reached a non-terminal state $(F,\ I,\ \delta)$. We show that one rule is applicable.

If $F|_I = \top$, rule True can be applied. If $F|_I = \bot$, there exists a clause $C \in F$ such that $C|_I = \bot$. The conflict level $\delta(C) = c$ may be either zero or positive. If $c = 0$, rule False is applicable. Now assuming $c > 0$ we obtain with Inv. (3):

$$F \wedge \mathsf{decs}_{\leqslant c}(I) \quad \equiv \quad F \wedge \mathsf{decs}_{\leqslant c}(I) \wedge I_{\leqslant c} \quad \models \quad I_{\leqslant c}.$$

Due to $I_{\leqslant c}(F) \equiv \bot$ we further have $F \wedge I_{\leqslant c} \equiv F \wedge \mathsf{decs}_{\leqslant c}(I) \equiv \bot$. By simply picking $\neg D = \mathsf{decs}_{\leqslant c}(I)$ we obtain $F \wedge \neg D \equiv F \wedge \neg D \wedge I_{\leqslant c} \equiv \bot$, thus $F \models D$. Clause D contains only decision literals and $\delta(D) = c$. From Inv. (1) and (2) we know that D contains exactly one decision literal for each decision level in $\{1, \ldots, c\}$. We choose $\ell \in D$ such that $\delta(\ell) = c$. Then the asserting level is given by $j = \delta(D \setminus \{\ell\})$ and we pick some backtrack level b where $j \leqslant b < c$. Without loss of generalization we assume the trail to be of the form $I = PQ$ where $\delta(P) = b$. After backtracking to decision level b, the trail is equal to $I_{\leqslant b} = PK$ where $K = Q_{\leqslant b}$. Since $D|_{PK} = \{\ell\}$, all conditions of rule Jump hold.

If $F|_I \notin \{\top, \bot\}$, there are still unassigned variables in V. If there exists a clause $C \in F$ where $C|_I = \{\ell\}$, the preconditions of rule Unit are met. If instead $\mathsf{units}(F|_I) = \emptyset$, there exists a literal ℓ with $V(\ell) \in V$ and $\delta(\ell) = \infty$, and the preconditions of rule Decide are satisfied.

In this argument, all possible cases are covered and thus in any non-terminal state a transition rule can be executed, i.e., the system never gets stuck.

Termination: To show termination we follow the arguments in [7,13] or more precisely the one in [5], except that our blocks (as formalized above with the block notion) might contain literals with different decision levels, i.e., subsequences of literals of the same decision level are interleaved as discussed in Sect. 3. This has an impact on the backtracking procedure adopted in rule Jump, where after backtracking to the end of $\mathsf{block}(I, b)$, trail P is extended by $K = Q_{\leqslant b}$. As discussed in the proof of Lemma 1, K contains no decision literals. Apart from that, the same argument applies as in [5], and SEARCH always terminates. □

6 Algorithm

The transition system presented in Sect. 4 can be turned into an algorithm described in Fig. 4 providing a foundation for our implementation. Unlike in [10], we refrain from giving implementation details but provide pseudocode on a higher abstraction level covering exclusively chronological backtracking.

Search: The main function Search takes as input a formula F, a set of variables V, a trail I and a decision level function δ. Initially, I is equal to the empty trail and all variables are assigned decision level ∞.

If all variables are assigned and no conflict occurred, it terminates and returns SAT. Otherwise, unit propagation by means of Propagate is executed until either a conflict occurs or all units are propagated.

If a conflict at decision level zero occurs, Search returns UNSAT, since conflict analysis would yield the empty clause even if the trail contains literals with decision level higher than zero. These literals are irrelevant for conflict analysis (line 7), and they may be removed from I prior to conflict analysis without affecting the computation of the learned clause. The resulting trail contains only propagation literals, and the new (current) decision level is zero upon which the empty clause is learned.

If a conflict at a decision level higher than zero occurs, conflict analysis (function Analyze) is executed. If no conflict occurs and there are still unassigned variables, a decision is taken and a new block started.

Propagate: Unit propagation is carried out until completion. Unlike in CDCL with non-chronological backtracking, the propagated literals may be assigned a decision level lower than the current one (line 3). In this case invariant LevelOrder presented in Sect. 1 does not hold anymore. Propagate returns the empty clause if no conflict occurs and the conflicting clause otherwise.

Analyze: If the conflict level is higher than zero and the conflicting clause C contains exactly one literal ℓ at conflict level c, then C can be used as reason instead of performing conflict analysis (lines 1–3). The idea is to save the computational effort of executing conflict analysis and adding redundant clauses.

Otherwise, a clause D is learned as in CDCL, e.g., the first unique implication point (1st-UIP) containing exactly one literal ℓ at conflict level. Let j be the lowest decision level at which D (or C, if it contains exactly one literal at conflict level) becomes unit. Then according to some heuristics the solver backtracks to a decision level $b \in [j, c-1]$.

This for instance, can be used to retain part of the trail, to avoid redundant work which would repeat the same assignments after backtracking. Remember that the decision levels on the trail may not be in ascending order. When backtracking to b, the solver removes all literals with decision level higher than b from I, i.e., all i-th slices with $i > b$.

Input: formula F, set of variables V, trail I, decision level function δ
Output: SAT iff F is satisfiable, UNSAT otherwise

```
Search ( F )
 1  V := V(F)
 2  I := ε
 3  δ := ∞
 4  while there are unassigned variables in V do
 5    C := Propagate ( F, I, δ )
 6    if C ≠ ⊥ then
 7      c := δ(C)
 8      if c = 0 then return UNSAT
 9      Analyze ( F, I, C, c )
10    else
11      Decide ( I, δ )
12  return SAT

Propagate ( F, I, δ )
 1  while some C ∈ F is unit {ℓ} under I do
 2    I := Iℓ
 3    δ(ℓ) := δ(C \ {ℓ})
 4    for all clauses D ∈ F containing ¬ℓ do
 5      if I(D) = ⊥ then return D
 6  return ⊥

Analyze ( F, I, C, c )
 1  if C contains exactly one literal at decision level c then
 2    ℓ := literal in C at decision level c
 3    j := δ(C \ {ℓ})
 4  else
 5    D := Learn ( I, C )
 6    F := F ∧ D
 7    ℓ := literal in D at decision level c
 8    j := δ(D \ {ℓ})
 9  pick b ∈ [ j, c − 1 ]
10  for all literals k ∈ I with decision level > b do
11    assign k decision level ∞
12    remove k from I
13  I := Iℓ
14  assign ℓ decision level j
```

Fig. 4. This is the algorithm for CDCL with chronological backtracking, which differs from its non-chronological backtracking version as follows: Propagated literals ℓ are assigned a decision level which may be lower than the current one (line 3 in Propagate). The conflict level may be lower than the current decision level (line 7 in Search). If the conflicting clause contains only one literal at conflict level, it is used as reason and no conflict analysis is performed (lines 1–3 in Analyze). Picking the backtracking level is usually non-deterministic (line 9 in Analyze). Backtracking involves removing from the trail I all (literals in) $\mathsf{slice}(I, i)$ with $i > b$ (line 12 in Analyze).

7 Implementation

We added chronological backtracking to our SAT solver CaDiCaL [4] based on the rules presented in Sect. 4, in essence implementing the algorithm presented in Fig. 4, on top of a classical CDCL solver. This required the following four changes, similar to those described in [10] and implemented in the source code which was submitted by the authors to the SAT 2018 competition [11]. This list is meant to be comprehensive and confirms that the changes are indeed local.

Asserting Level. During unit propagation the decision level $a = \delta(C\backslash\{\ell\})$, also called asserting level, of propagated literals ℓ needs to be computed based on the decision level of all the falsified literals in the reason clause C. This is factored out in a new function called `assignment_level`[1], which needs to be called during assignment of a variable if chronological backtracking is enabled.

Conflict Level. At the beginning of the conflict analysis the current conflict level c is computed in a function called `find_conflict_level` (see footnote 1). This function also determines if the conflicting clause has one or more falsified literals on the conflict level. In the former case we can simply backtrack to backtrack level $b = c - 1$ and use the conflicting clause as reason for assigning that single literal on the conflict level. Even though not described in [10] but implemented in their code, it is also necessary to update watched literals of the conflict. Otherwise important low-level invariants are violated and propagation might miss falsified clauses later. In order to restrict the changes to the conflict analysis code to a minimum, it is then best to backtrack to the conflict level, if it happens to be smaller than the current decision level. The procedure for deriving the learned clause D can then remain unchanged (minimizing the 1st-UIP clause).

Backtrack Level. Then we select the backtrack level b with $j \leqslant b < c$, where j is the minimum backjump level j (the second largest decision level in D) in the function `determine_actual_backtrack_level` (see footnote 1). By default we adopted the heuristic from the original paper [10] to always force chronological backtracking ($b = c - 1$) if $c - j > 100$ (`T` in [10]) but in our implementation we do not prohibit chronological backtracking initially (`C` in [10]). Beside that we adopted a variant of reusing the trail [16] as follows. Among the literals on the trail assigned after and including the decision at level $j + 1$ we find the literal k with the largest variable score and backtrack to b with $b + 1 = \delta(k)$.

Flushing. Finally, the last required change was to flush literals from the trail with decision level larger than b but keep those smaller or equal than b. Instead of using an extra data structure (queue) as proposed in [10] we simply traverse the trail starting from block $b + 1$, flushing out literals with decision level larger than b. It is important to make sure that all the kept literals are propagated again (resetting the `propagated` (see footnote 1) level).

[1] Please refer to the source code of CaDiCaL provided at http://fmv.jku.at/chrono.

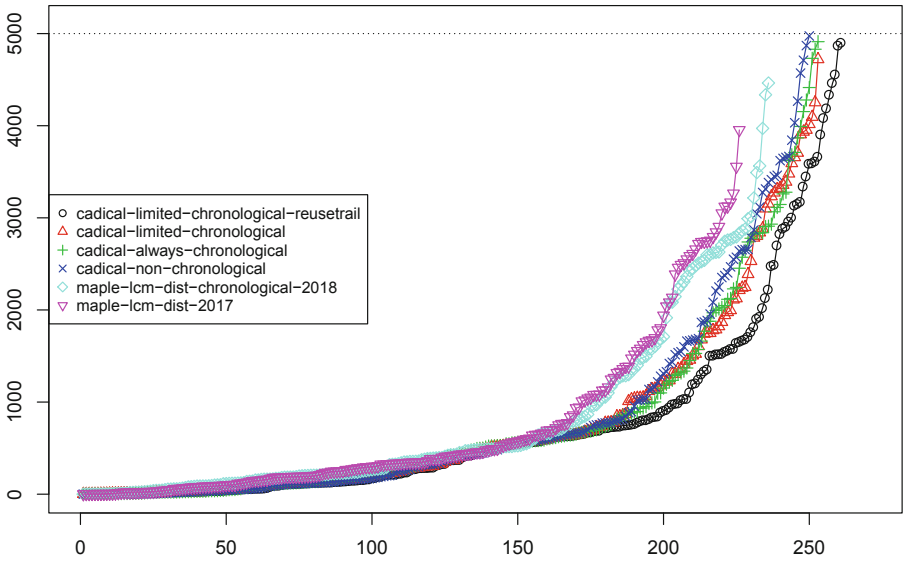

Fig. 5. Cactus plot for benchmarks of the main track of the SAT Competition 2018.

8 Experiments

We evaluated our implementation on the benchmarks from the main track of the SAT Competition 2018 and compare four configurations of CaDiCaL [4]. We also consider maple-lcm-dist-2017 [17], also called Maple_LCM_Dist, which won the main track of the SAT Competition 2017, on which maple-lcm-dist-chronological-2018 [11], also called Maple_LCM_Dist_ChronoBT, is based. We consider the latter as reference implementation for [10]. It won the main track of the SAT Competition 2018 on the considered benchmark set.

The experiments were executed on our cluster where each compute node has two Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz with turbo-mode disabled. Time limit was set to 3600 seconds and memory limit to 7 GB. We used version "0nd" of CaDiCaL. Compared to the SAT Competition 2018 version [4] it incorporates new phase saving heuristics and cleaner separation between stabilizing and non-stabilizing phases [14]. This gave substantial performance improvements on satisfiable formulae [3]. Nevertheless adding chronological backtracking improves performance even further as the cactus plot in Fig. 5 and Table 1 show.

The default version cadical-limited-chronological-reusetrail is best (preliminary experiments with CaDiCaL optimized for SAT Race 2019 did not confirm this result though). It uses the limit $C = 100$ to chronologically backtrack if $c - j > 100$ and further reuses the trail as explained in the previous section. For cadical-limited-chronological reusing the trail is disabled and less instances are solved. Quite remarkable is that configuration cadical-always-chronological ranks third, even though it always enforces chronological backtracking ($b = \mathrm{c}{-}1$).

Table 1. Solved instances of the main track of the SAT Competition 2018.

Solver configurations	Solved instances		
	Total	SAT	UNSAT
cadical-limited-chronological-reusetrail	261	155	106
cadical-limited-chronological	253	147	106
cadical-always-chronological	253	148	105
cadical-non-chronological	250	144	106
maple-lcm-dist-chronological-2018	236	134	102
maple-lcm-dist-2017	226	126	100

On these benchmarks there is no disadvantage in always backtracking chronologically! The original classical CDCL variant cadical-non-chronological comes next followed by the reference implementation for chronological backtracking maple-lcm-dist-chronological-2018 and then maple-lcm-dist-2017 last, confirming the previous results in [10]. Source code and experimental data can be found at http://fmv.jku.at/chrono.

9 Conclusion

The success of MAPLE_LCM_DIST_CHRONOBT [11] is quite remarkable in the SAT Competition 2018, since the solver violates various invariants previously considered crucial for CDCL solvers (summarized in Fig. 1). The corresponding paper [10] however was lacking proofs. In this paper we described and formalized a framework for combining CDCL with chronological backtracking. Understanding precisely which invariants are crucial and which are redundant was the main motivation for this paper. Another goal was to empirically confirm the effectiveness of chronological backtracking within an independendent implementation.

Our main contribution is to precisely define the concepts introduced in [10]. The rules of our framework simplify and generalize chronological backtracking. We may relax even more CDCL invariants without compromising the procedure's correctness. For instance first experiments show that during the application of the Unit rule it is not necessary to require that the formula is not falsified by the trail. Similarly, requiring the formula not to be falsified appears to be sufficient for rule Decide (no need to require that there are no units).

Our experiments confirm that combining chronological backtracking and CDCL has a positive impact on solver performance. We have further explored reusing the trail [16] during backjumping, which requires a limited form of chronological backtracking, too. Our experiments also show that performing chronological backtracking exclusively does not degrade performance much and thus for instance has potential to be used in propositional model counting. Furthermore, besides counting, possible applications may be found in SMT and QBF. We further plan to investigate the combination of these ideas with total assignments following [6].

References

1. Artho, C., Biere, A., Seidl, M.: Model-based testing for verification back-ends. In: Veanes, M., Viganò, L. (eds.) TAP 2013. LNCS, vol. 7942, pp. 39–55. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38916-0_3
2. Audemard, G., Simon, L.: Refining restarts strategies for SAT and UNSAT. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 118–126. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33558-7_11
3. Biere, A.: CaDiCaL at the SAT Race 2019. In: Proceedings of SAT Race (2019, Submitted)
4. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT competition 2018. In: Proceedings of the SAT Competition 2018 - Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2018-1, pp. 13–14. University of Helsinki (2018)
5. Blanchette, J.C., Fleury, M., Weidenbach, C.: A verified SAT solver framework with learn, forget, restart, and incrementality. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 25–44. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_4
6. Goultiaeva, A., Bacchus, F.: Off the trail: re-examining the CDCL algorithm. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 30–43. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_4
7. Marić, F., Janičić, P.: Formalization of abstract state transition systems for SAT. Logical Methods Comput. Sci. **7**(3), 1–37 (2011)
8. Marques-Silva, J.P., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 131–153. IOS Press (2009)
9. Marques-Silva, J.P., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: Proceedings of ICCAD 1996, pp. 220–227 (1996)
10. Nadel, A., Ryvchin, V.: Chronological backtracking. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 111–121. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_7
11. Nadel, A., Ryvchin, V.: Maple LCM dist ChronoBT: featuring chronological backtracking. In: Proceedings of SAT Competition 2018 - Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2018-1, p. 29. University of Helsinki (2018)
12. Niemetz, A., Preiner, M., Biere, A.: Model-based API testing for SMT solvers. In: Proceedings of SMT 2017, Affiliated with CAV 2017, p. 10 (2017)
13. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). J. ACM **53**(6), 937–977 (2006)
14. Oh, C.: Between SAT and UNSAT: the fundamental difference in CDCL SAT. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 307–323. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_23
15. Oh, C.: Improving SAT solvers by exploiting empirical characteristics of CDCL. Ph.D. Thesis, New York University, Department of Computer Science (2016)

16. van der Tak, P., Ramos, A., Heule, M.: Reusing the assignment trail in CDCL solvers. JSAT **7**(4), 133–138 (2011)
17. Xiao, F., Luo, M., Li, C.M., Manyà, F., Lü, Z.: MapleLRB_LCM, Maple_LCM, Maple_LCM_Dist, MapleLRB_LCMoccRestart, and Glucose-3.0+width in SAT Competition 2017. In: Proceedings of SAT Competition 2017 - Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2017-1, pp. 22–23. University of Helsinki (2017)