

1 Lazy Reimplication in Chronological Backtracking

2 **Robin Coutelier** ✉ 

3 TU Wien, Vienna, Austria

4 **Mathias Fleury** ✉ 

5 University Freiburg, Freiburg, Germany

6 **Laura Kovács** ✉ 

7 TU Wien, Vienna, Austria

8 — Abstract —

9 Chronological backtracking is an interesting SAT solving technique within CDCL reasoning, as it
10 backtracks less aggressively upon conflicts. However, chronological backtracking is more difficult
11 to maintain due to its weaker SAT solving invariants. This paper introduces a lazy reimplication
12 procedure for missed lower implications in chronological backtracking. Our method saves propaga-
13 tions by reimplicating literals on demand, rather than eagerly. Due to its modularity, our work can
14 be replicated in other solvers, as shown by our results in the solvers CADICAL and GLUCOSE.

15 **2012 ACM Subject Classification** Theory of computation → Constraint and logic programming;
16 Theory of computation → Automated reasoning

17 **Keywords and phrases** Chronological Backtracking, CDCL, Invariants, Watcher Lists

18 **Digital Object Identifier** 10.4230/LIPIcs.SAT.2024.24

19 **Supplementary Material** *CADICAL* (Source Code): [https://github.com/arminbiere/cadical/
20 tree/strong-backtrack](https://github.com/arminbiere/cadical/tree/strong-backtrack)

21 *NAPSAT* (Source Code *GitLab*): <https://gitlab.uliege.be/smt-modules/sat-library>

22 *NAPSAT* (Source Code *GitHub*): <https://github.com/RobCoutel/NapSAT>

23 *GLUCOSE* (Source Code): <https://github.com/m-fleury/glucose>

24 **Acknowledgements** The authors acknowledge support from the ERC Consolidator Grant ARTIST
25 101002685; the TU Wien Doctoral College TrustACPS; the FWF SpyCoDe SFB projects F8504; the
26 WWTF Grant ForSmart 10.47379/ICT22007; the and the Amazon Research Award 2023 QuAT; the
27 bwHPC of the state of Baden-Württemberg; the German Research Foundation (DFG) through grant
28 INST 35/1597-1 FUGG; and a gift from Intel Corporation. The first author’s Master thesis [10]
29 conducted at the University of Liège under the supervision of Pascal Fontaine laid the foundations
30 of this work. We thank him for his insights.

31 **1** Introduction

32 In the past few years, chronological backtracking in CDCL-based SAT solving attracted re-
33 newed interest as it implements less aggressive procedures when backtracking upon conflicts,
34 particularly for undoing literal assignments stored in the assignment stack. Chronologi-
35 cal backtracking has been proven sound and complete, while also empirically improving
36 performance on SAT competition problems [15, 17, 18].

37 Without chronological backtracking in SAT solving, the truth value of each literal is set as
38 early as possible in the solving process. With chronological backtracking, there are, however,
39 missed lower implications (MLI), i.e., clauses that could have set a literal at a lower SAT
40 decision level. As a remedy to MLI, IntelSAT [17] and CADICAL-1.9.4 [19] fix the level
41 of the assignments. Modifying levels impacts solving performance and significantly clutters
42 the code; for example, reimplication techniques for detecting MLI have been removed in
43 CADICAL-1.9.5 [4] due to the increased code complexity.



© Robin Coutelier, Mathias Fleury, and Laura Kovács;
licensed under Creative Commons License CC-BY 4.0

27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024).

Editors: Supratik Chakraborty and Jie-Hong Roland Jiang; Article No. 24; pp. 24:1–24:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Invariant properties for CDCL algorithms (Section 3)	

Invariant 1– <i>Weak watched literals</i> : No conflict is missed.	

Invariants on implications, native for NCB and WCB (Section 3.1)	

Invariant 2– <i>Implied literals</i> :	Literals are decisions or implied by a clause C that is made unit by the partial assignment.
Invariant 3– <i>Topological order</i> :	The partial SAT assignment follows a topological order of the implication graph.

Strong invariant, non-trivial for CDCL with CB and native in NCB (Section 3.1)	

Invariant 4– <i>Strong watched literals</i> : No implication nor conflict can be missed.	

■ **Figure 1** Invariant properties for CDCL-based SAT solving and maintained by the different chronological backtracking (CB) strategies, particularly by non-chronological backtracking (NCB) and weak chronological backtracking (WCB).

44 In this paper, we introduce a *lazy reimplication procedure for resolving missed lower*
45 *implications in chronological backtracking, while also ensuring efficiency in SAT solving.*
46 Doing so, in Figure 1, we state the invariant properties to be maintained during CDCL
47 and highlight differences between relevant backtracking approaches in SAT solving. In
48 particular, we consider and adjust variants of *non-chronological backtracking* (NCB) [21] and
49 strong chronological backtracking (SCB) [17]. A formal presentation of these invariants and
50 backtracking variants is given in Section 3. Using the invariants of Figure 1, in Section 4
51 we introduce a lazy reimplication procedure to handle missed lower implications, with a
52 particular focus on handling unit implications after backtracking. We also adjust and enhance
53 the first unique implication point (UIP) algorithm [16] with the knowledge of missed lower
54 implications. Our approach is sound (Section 5). We implemented our work in the new solver
55 NAPSAT [20] and present our empirical findings in Section 6. To demonstrate the flexibility
56 of our lazy reimplication techniques, we also implemented the algorithms of Section 4 in
57 CADICAL [5] and GLUCOSE [1], and provide empirical comparisons using these solvers.

58 **Related work.** Within CDCL, the truth values of literals are assigned by guessing (deciding)
59 and propagating them in a trail until a conflict is found. Upon conflict analysis, the trail
60 is adapted by backtracking, i.e. revoking some assignments and swapping the truth value
61 of one variable, called the unique implication point (UIP). The standard approach [21] is
62 to fix the conflict as early as possible with *non-chronological backtracking* (NCB) and all
63 assignments between the current point and the point where the UIP is set are deleted.

64 A different backtracking approach comes with *chronological backtracking* (CB) [15, 18].

65 Here, a less aggressive backtracking scheme is used and some propagations and decisions are
 66 kept. Chronological backtracking may backjump at any level between the UIP and the UIP
 67 falsification point minus one. As a result, chronological backtracking resets a smaller part
 68 of the trail, but it may miss propagations that could have been done earlier if the learned
 69 clause was known beforehand. In this paper, we refer by *weak chronological backtracking*
 70 (WCB) to the CDCL algorithms that use chronological backtracking mechanism and which
 71 do not detect every propagation as early as possible (see Section 3.2)

72 For recovering such missed propagations, we define *strong chronological backtracking*
 73 (SCB). In particular, Nadel [17] introduced a reimplication procedure that eagerly re-assigns
 74 literals detected as missed lower implications to their lowest possible level. We refer to this
 75 SCB technique as *eager strong chronological backtracking* (ESCB). Our work introduces
 76 a new SCB method, *lazy strong chronological backtracking* (LSCB). Unlike ESCB, within
 77 LSCB we reimpl miss implications on demand. As such, our work is stronger than WCB,
 78 as WCB does not perform reimplications at all. In addition, our technique is shown to be
 79 easier and more flexible to implement than ESCB or WCB (Section 6).

80 **Our contributions.** This paper brings the following contributions to chronological back-
 81 tracking in CDCL-based proof search.

- 82 1. We formalize invariant properties that need to be maintained during SAT solving with
 83 chronological backtracking (Section 3). Our invariants incorporate and reason over dif-
 84 ferent backtracking strategies.
- 85 2. We introduce *lazy strong chronological backtracking* (LSCB) for on-demand reimplication
 86 of (conflict) (Sec. 4) and prove soundness of our approach (Section 5).
- 87 3. We implement our work in the new NAPSAT [11, 20] solver (Section 6). To showcase
 88 the flexibility and efficiency of our approach, we integrate LSCB into CADICAL [5] and
 89 GLUCOSE [1], and provide experimental comparisons using these solvers.

90 2 Preliminaries

91 We assume familiarity with propositional logic and CDCL [6], and use the standard log-
 92 ical connectives \neg , \wedge , and \vee . A finite set of elements (e.g. literals) is called *conjunctive*
 93 (respectively, *disjunctive*) to indicate that the set is the conjunction (respectively, disjunc-
 94 tion) of its elements. An *ordered set* is a set \mathcal{S} which defines a bijective function $p_{\mathcal{S}}$ from
 95 elements of \mathcal{S} to naturals, such that $p_{\mathcal{S}}(e)$ is the position of the element e in the ordered
 96 set \mathcal{S} . We consider the first element of \mathcal{S} to have the position 0. Ordered sets are stable
 97 under the removal of elements; that is, for the ordered sets $\mathcal{S}, \mathcal{T}, \mathcal{U}$ with $\mathcal{S} = \mathcal{T} \setminus \mathcal{U}$, we
 98 have $\forall e, e' \in \mathcal{S}. p_{\mathcal{S}}(e) < p_{\mathcal{S}}(e') \Leftrightarrow p_{\mathcal{T}}(e) < p_{\mathcal{T}}(e')$. We denote by \cdot set concatenation; for
 99 simplicity, we use \cdot to also denote appending a sequence with an element. We write $\mathcal{S}[a : b]$
 100 to select the ordered elements e in \mathcal{S} with positions $a \leq p_{\mathcal{S}}(e) \leq b$.

101 We denote by \mathcal{V} a countable set of Boolean variables v . We consider propositional
 102 formulas F in conjunctive normal form (CNF), represented by a conjunctive set of clauses
 103 $\{C_1, C_2, \dots, C_n\}$ over \mathcal{V} . Clauses are disjunctive sets of literals $C = \{c_1, c_2, \dots, c_m\}$, where
 104 a literal c_i is either a Boolean variable v or a negation $\neg v$ of a variable v .

105 To efficiently identify unit propagations, SAT solvers track two literals per clause in the
 106 two-watched literal scheme [16]. We denote the watched literals of a clause C by c_1 and c_2 , and
 107 write $WL(c_1)$ and $WL(c_2)$ for the watched lists of c_1 and c_2 . We have $C \in WL(c_1) \cap WL(c_2)$.

108 During SAT solving, solvers keep track of a *partial assignment*, also called *trail* and
 109 denoted as the conjunctive ordered set $\pi = \tau \cdot \omega$, which is split into two parts: (i) τ is the set
 110 of literals that were already propagated and do not need to be inspected anymore (by checking

111 the watch lists); (ii) ω is the *propagation queue* containing literals that were implied and
 112 waiting to be propagated. The partial assignment π contains the set $\pi^d \subseteq \pi$ of decision literals.
 113 Decisions literals in π^d are arbitrarily chosen literals when unit propagation cannot be further
 114 used and the truth value of a (decision) literal needs to be picked and assigned. We call *unit*,
 115 a clause C containing exactly one unassigned literal ℓ and whose other literals are falsified,
 116 i.e., $\exists \ell \in C. C \setminus \{\ell\}, \pi \models \perp \wedge |\ell| \notin |\pi|$. For conflict analysis, the propagation reasons of literals
 117 are analyzed. Therefore, SAT solvers use a ρ function that maps literals to clauses such that
 118 $\rho(\ell)$ captures the reason for propagating ℓ . The reason for propagating ℓ is the clause C that
 119 implied ℓ under assumption π , that is, $[\ell \in \pi] \wedge [\ell \in \rho(\ell)] \wedge [\rho(\ell) \setminus \{\ell\} \wedge \pi \models \perp]$. Following [15],
 120 we use δ to represent the (decision) *level of* ℓ , i.e., the level when a truth assignment to ℓ
 121 was made. Formally, if ℓ is a decision literal, then the level $\delta(\ell) = \delta(-\ell)$ of ℓ is the number
 122 of decisions preceding and including ℓ , that is $\delta(\ell) = |\pi[0 : p_\pi(\ell)] \cap \pi^d|$. Further, for literals ℓ
 123 implied by $\rho(\ell)$, we have $\delta(\ell) = \max_{\ell' \in \rho(\ell) \setminus \{\ell\}} \delta(\ell')$. Finally, $\delta(\ell) = \infty$ for unassigned literals
 124 ℓ . The definition of δ is extended to clauses and trails, with $\delta(C) = \max_{\ell \in C} \delta(\ell)$; similarly
 125 for $\delta(\pi)$. The level of the empty set is $\delta(\emptyset) = 0$. We write $\delta[\ell \leftarrow d]$ to denote that the level
 126 of ℓ is updated to d . We reserve the special symbol \blacksquare to denote *undefined clauses* during
 127 SAT solving, with $\delta(\blacksquare) = \infty$.

128 In standard CDCL with non-chronological backtracking (NCB) [21], level δ stores the
 129 number of decisions that appear before in the trail, and is always the lowest level possible.
 130 In CDCL with chronological backtracking, the history of propagations and conflicts may,
 131 however, lead to *missed lower implications* (MLI), where a MLI captures the fact that a
 132 clause C is satisfied by a unique literal ℓ at a level *strictly higher* than $\delta(C \setminus \{\ell\})$. Therefore,
 133 in a MLI, the literal ℓ could have been propagated at a lower level in the trail.

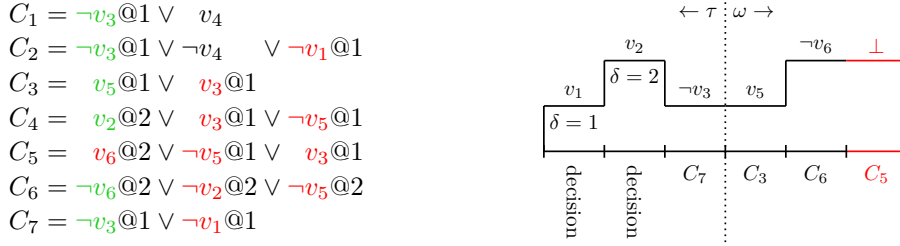
134 **► Example 1 (Missed Lower Implications – MLI).** Figure 2 shows a clause set $\{C_1, \dots, C_7\}$
 135 and a trail $\pi = \tau \cdot \omega$ during CDCL solving with chronological backtracking. The trail diagram
 136 displays, from left to right, the order in which literals are decided and propagated, as well
 137 as the location of the propagation head (symbolized by the dashed line). The propagation
 138 level is symbolized by the height of the step. As a visual aid, literals are colored **green**, **red**
 139 or black, symbolizing respectively satisfied, falsified, and unassigned literals. The watched
 140 literals are the first two in the clause.

141 In the example, the clause set $F_0 = \{C_1, \dots, C_6\}$ was given as input. The decisions
 142 v_1, v_2 and v_3 were made, then the solver found a conflict in C_2 after implying v_4 with reason
 143 $\rho(v_4) = C_1$. The clause $C_7 = \neg v_3 @ 3 \vee \neg v_1 @ 1$ is learned and the solver backtracks to level
 144 $\delta(C_7) - 1 = 2$, continuing its propagations until it reaches the assignment shown on Figure 2.
 145 Figure 2 shows that C_4 is a MLI. Indeed, v_2 is satisfied at level 2, while all other literals are
 146 falsified at level 1. After backtracking to level 1, the implication of v_2 by C_4 is missed since
 147 (i) $\neg v_3$ was already propagated, and (ii) C_4 is watched by v_3 and v_2 .

148 **3 Invariant Properties on CDCL Variants**

149 To properly handle MLI similar to Example 1, in this section we revisit and formalize our
 150 invariants from Figure 1, expressing properties that need to be maintained in (variants of)
 151 CDCL with chronological backtracking.

152 The crux of our invariant properties is captured by watched literals [16]. They reduce the
 153 number of clauses to be checked when propagating a literal. Invariant 1 therefore expresses
 154 that, as long as CDCL does not falsify one of the watched literals c_1, c_2 of a clause C , the
 155 clause C is not a conflict. Therefore, when propagating a watched literal c_i during CDCL,
 156 only checking the clauses watched by $\neg c_i$ is sufficient to not miss any conflict.



■ **Figure 2** C_4 is a MLI while C_5 would be a MLI if v_5 was set to true. We use the notation $v@1$ to indicate that literal v is on level 1.

157 ► **Invariant 1** (Weak watched literals). Let $\pi = \tau \cdot \omega$ be the current trail. For each clause
 158 $C \in F$ watched by the two distinct watched literals c_1, c_2 , we have $\neg c_1 \in \tau \Rightarrow \neg c_2 \notin \tau$.

159 Invariant 1 ensures that conflicts are not missed during CDCL. Indeed, if there is a
 160 conflicting clause C , the conflict is found after propagating all literals of C . After propagation,
 161 no more literal has to be propagated, so $\pi = \tau$. A conflicting clause C thus violates Invariant 1,
 162 and hence the conflict of C is captured during CDCL.

163 3.1 CDCL Invariants on Implications

164 We next ensure the soundness of unit implications. Invariant 2 expresses that literals are
 165 either decisions or implied by a sound implication. Note that an implication can be performed
 166 if there is only one unassigned literal that can satisfy a clause C ; hence, C is a unit clause.
 167 In addition to ensuring that the solver infers correct literals, Invariant 2 is also relevant for
 168 conflict analysis (see proof of Theorem 13).

169 ► **Invariant 2** (Implied literals). If a literal ℓ is in the trail π , then ℓ is either a decision literal
 170 or ℓ is implied by π and its reason $\rho(\ell)$. That is,

$$171 \quad \forall \ell \in \pi. \ell \in \pi^d \vee [\ell \in \rho(\ell) \wedge [\rho(\ell) \setminus \{\ell\} \wedge \pi] \models \perp].$$

172 To perform conflict analysis with the first unique implication point (UIP) [16], CDCL
 173 solving assumes that literals are organized in a topological sort of the implication graph.

174 ► **Invariant 3** (Topological order). Trail π is a topological order of the implication graph:

$$175 \quad \forall \ell \in \pi. \forall \ell' \in \rho(\ell). p_\pi(\neg \ell') \leq p_\pi(\ell),$$

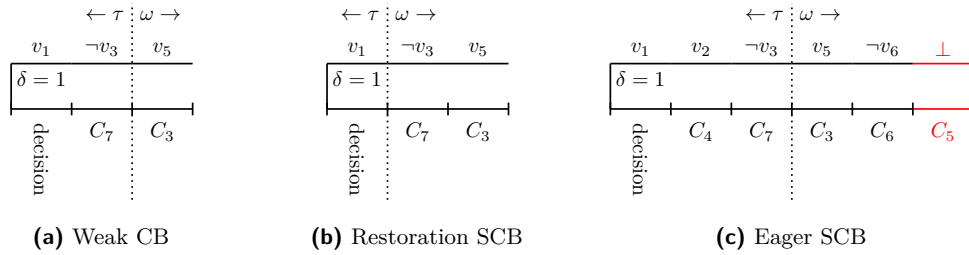
176 where $p_\pi(\ell)$ and $p_\pi(\neg \ell')$ are respectively the positions of ℓ and $\neg \ell'$ in π .

177 Invariant 3 holds by construction in CDCL with non-chronological backtracking (NCB) and
 178 chronological backtracking without reimplication. However, Invariant 3 is crucial in any
 179 setting of reimplicating literals.

180 Finally, we impose that Boolean constraint propagation (BCP) in CDCL does not miss
 181 unit implication during proof search. Invariant 4 therefore formalizes that CDCL cannot
 182 have one propagated falsified watched literal without the clause being satisfied.

183 ► **Invariant 4** (Strong watched literals). Consider the trail $\pi = \tau \cdot \omega$. For each clause $C \in F$
 184 watched by the two distinct watched literals c_1, c_2 , we have $\neg c_1 \in \tau \Rightarrow c_2 \in \pi$.

185 Invariant 4 strengthens Invariant 1. When a conflicting clause C is detected while
 186 propagating ℓ , the literal ℓ cannot be added to τ without violating Invariant 4. As such, by
 187 imposing Invariant 4, the conflict of C is resolved and the trail is adapted.



■ **Figure 3** Different CB ways of handling the missed lower implications of Figure 2.

188 3.2 Chronological Backtracking

189 Invariant 4 holds for CDCL with NCB, since the trail contains monotonically increasing
 190 decision levels. Therefore, within NCB, literals are unassigned in the reverse order of
 191 propagation. In particular, if a literal ℓ is satisfied in a clause C when propagating another
 192 literal ℓ' , the literal ℓ remains satisfied at least until ℓ' is backtracked.

193 **Weak chronological backtracking (WCB).** When considering (variants of) chronological
 194 backtracking in CDCL, Invariant 4 becomes critical, as detailed next. The core idea is to
 195 save parts of the trail without repropagating unlike [12].

196 ► **Example 5.** Let us revisit the example of Figure 2. Figure 3a shows the trail after
 197 backtracking to level 1. Literal v_3 is already propagated ($v_3 \in \tau$), and C_4 is still watched by
 198 v_2 and v_3 . Therefore, the implication of v_2 is missed, even though Invariant 1 is not violated.

199 To circumvent the problem of missing implications similar to Example 5, we distinguish
 200 a *weak chronological backtracking (WCB)* variant of CDCL with chronological backtracking.
 201 Within WCB, Invariant 4 is not necessarily satisfied, as unit implications at lower levels
 202 can be missed. To recover Invariant 4 in variants of CDCL with CB, we adjust and label
 203 two existing solutions in SAT solving: (i) *restoration* [18], for repairing the trail p after
 204 backtracking; and (ii) *prophylaxis* [17], for forcing literals at the lowest possible level.

205 **Restoration.** We call *restoration* the approach in which the trail π is repaired by pushing
 206 back the propagation head when propagating [18]. Out-of-order literals are repropagated
 207 whenever they are moved in the trail during backtracking. For example, in Figure 3b, v_3 was
 208 the first literal that changed position during backtracking, so this is where the propagation
 209 head is set. When backtracking to level δ , the propagation head is set to $p_\pi(\pi^d[\delta])$. When
 210 v_3 is repropagated, v_2 is reimplied. We, therefore, restore Invariant 4 by repropagating
 211 the out-of-order literals. We call this approach *restoring strong chronological backtracking*
 212 (RSCB), allowing to restore the trail π by propagating more. It is also used in CADICAL [5].

213 **Prophylaxis.** We name *prophylaxis*¹ the approach in which missed lower implications are
 214 prevented from becoming missed unit implications [17]. Prophylaxis uses an eager reimpli-
 215 cation procedure and imposes the validity of a compatibility invariant; we formalize this
 216 property in Invariant 6. That is, when a clause C is detected to be a missed lower implica-
 217 tion of ℓ , then ℓ is reimplied at level $\delta(C \setminus \{\ell\})$ and its reason for propagation is updated.
 218 Prophylaxis thus enforces our *backtrack compatible* Invariant 6 by ensuring that no clause
 219 can become unit after backtracking. Furthermore, Invariant 6 guarantees that literals are
 220 always propagated at the lowest level, and conflicts are detected at the lowest level.

¹ “Prophylaxis” is a chess term referring to a move that deals with a threat before it becomes a problem.

■ **Table 1** CB variants in CDCL, together with their invariant properties.

	Inv. 1	Inv. 4	Inv. 6	Inv. 9	Solvers
NCB	✓	✓	✓	✓	Most CDCL solvers
WCB	✓	✗	✗	✗	Our work – NAPSAT
RSCB	✓	✓	✗	✗	MAPLE_LCM_DIST [18], CADICAL
ESCB	✓	✓	✓	✓	IntelSAT and CADICAL 1.9.4
LSCB	✓	✓	✗	✓	Our work – NAPSAT, now in CADICAL

221 ► **Invariant 6** (Backward compatible watched literals). *For each clause $C \in F$ watched by the*
 222 *two distinct watched literals c_1, c_2 , we have $\neg c_1 \in \tau \Rightarrow [c_2 \in \pi \wedge \delta(c_2) \leq \delta(c_1)]$.*

223 ► **Example 7.** Figure 3c shows the trail after v_2 is reimplied. In this case, v_2 was a decision,
 224 and $\neg v_6$ has to be reimplied to level 1 as well. All literals are propagated at the lowest
 225 possible level. Thus, using Invariant 6, the conflict C_5 is properly detected at level 1, instead
 226 of level 2. Figure 3c also shows that the trail π no longer follows a topological order of the
 227 implication graph. These issues have to be addressed.

228 Based on Invariant 6, *eager strong chronological backtracking (ESCB)* is used in [17, 19],
 229 yielding a CDCL method with chronological backtracking that satisfies Invariant 6 by eager
 230 reimplication of missed lower implications. In Table 1 we summarize backtracking strategies
 231 in CDCL, also listing our solution in this respect: *lazy reimplication in strong chronological*
 232 *backtracking (LSCB)*. Our LSCB approach maintains Invariant 1 and Invariant 4, while
 233 weakening Invariant 6 via Invariant 9, as described next in Section 4 and implemented in
 234 Algorithm 1.

235 4 Adapting CDCL with Lazy Reimplications

236 Embedding the prophylaxis approach of Section 3.2 in existing CDCL data structures is
 237 highly non-trivial, due to the rigid and entangled data structures [17, 19], see e.g. [10]. In
 238 addition, reimplifying literals [17, 19] changes the implication graph, and hence the trail π is
 239 no longer a topological sort of the implications; as such, Invariant 3 must be restored.

240 While the restoration approach of Section 3.2 offers a practically simpler solution, restora-
 241 tion might require the re-propagation of a large part of π and thus can be computationally
 242 very expensive. For example, while in Figure 3b only one literal had to be re-propagated,
 243 re-propagation could be applied on an arbitrary number of literals.

244 **Our solution: Lazy reimplication in CDCL.** To overcome inefficiencies of restoration
 245 and pure prophylaxis, our work advocates a *lazy reimplication* technique for CDCL with
 246 strong chronological backtracking. To ensure Invariant 4, we reimplify literals after backtrack-
 247 ing. That is, *we detect missed lower implications eagerly but reimplify them lazily.*

248 Our lazy reimplication approach for CDCL-based solving is summarized in Algorithm 1. In
 249 what follows, we describe the key ingredients of Algorithm 1 and revise the CDCL invariants
 250 of Section 3, adjusted to Algorithm 1. To this end, we introduce a lazy reimplication vector λ
 251 to store missed lower implications, where λ is a function from literals to clauses. Intuitively,
 252 the lazy reimplication vector λ stores the lowest

253 detected missed lower implication for each literal ℓ . The clause $\lambda(\ell) \neq \blacksquare$ is an alternative
 254 reason that would propagate ℓ in trail π , lower than the reason $\rho(\ell)$. Initially, no clause is
 255 assigned, and $\forall \ell. \lambda(\ell) = \blacksquare$ (that is, the undefined clause). Invariant 8 is asserted to hold
 256 during proof search.

24:8 Lazy Reimplication in Chronological Backtracking

257 ► **Invariant 8** (Lazy reimplication). *If the lazy reimplication reason $\lambda(\ell)$ of literal ℓ is defined,*
 258 *then the clause $\lambda(\ell)$ is a missed lower implication of ℓ . That is,*

$$\begin{aligned}
 259 \quad \lambda(\ell) \neq \blacksquare &\Rightarrow \quad \ell \in \pi \wedge \ell \in \lambda(\ell) \\
 260 &\quad \wedge (\lambda(\ell) \setminus \{\ell\} \wedge \pi) \models \perp \\
 261 &\quad \wedge \delta(\lambda(\ell) \setminus \{\ell\}) < \delta(\ell)
 \end{aligned}$$

262 When a missed lower implication for ℓ is detected, then ℓ is not reimplied directly. Rather,
 263 we store the MLI in λ until ℓ is unassigned during backtracking. For example, if a literal ℓ
 264 is assigned at level 3 and a missed lower implication C for ℓ is detected with $\delta(C \setminus \{\ell\}) = 1$,
 265 then backtracking to level 2 will reassign ℓ from level 3 to level 1 by C .

266 Using our lazy reimplication vector λ , we weaken Invariant 6 into Invariant 9 such
 267 that, during backtracking, we identify missed lower implications without requiring the re-
 268 propagation of out-of-order literals.

269 ► **Invariant 9** (Lazy backtrack compatible watched literals). *Consider the trail $\pi = \tau \cdot \omega$. For*
 270 *each clause $C \in F$, if one watched literal c_1 of C is falsified by τ , then the other c_2 must be*
 271 *satisfied at a lower level, or a missed lower implication lower than c_1 is set in λ .*

$$272 \quad \neg c_1 \in \tau \Rightarrow \left(c_2 \in \pi \wedge (\delta(c_2) \leq \delta(c_1) \vee \delta(\lambda(c_2) \setminus \{c_2\}) \leq \delta(c_1)) \right)$$

273 **Lazy reimplication for strong chronological backtracking – LSCB.** Guided by the
 274 reimplication and backtracking properties of Invariant 8 and Invariant 9, Algorithm 1 shows
 275 our LSCB algorithm for CDCL with chronological backtracking, as a slight refactoring of
 276 weak chronological backtracking (WCB). In the following algorithms, particularities of LSCB
 277 are highlighted in blue.

278 An important detail should be noted upon Algorithm 1: in our abstract representation, it
 279 is not explicitly checked whether the learned clause D is different from the conflicting clause
 280 C ; such a check, however, should be performed when implementing Algorithm 1. Indeed, as
 281 pointed out in RSCB [15], it is possible that a conflicting clause C does not require conflict
 282 analysis since C might already be a UIP. However, if the highest literal ℓ in C is a MLI,
 283 then the clause might be conflicting again after backtracking (see Algorithm 4).

284 ► **Example 10.** Consider the example of Figure 2. Here, the conflicting clause C_5 only has
 285 one literal at the highest level, and, as such, it qualifies as a UIP. Therefore no conflict
 286 analysis is required, we only backtrack to level 1, and then C_5 implies v_6 at level 1. However,
 287 if $\neg v_6$ was a missed lower implication, then backtracking to level 1 would reimpl $\neg v_6$, with
 288 C_5 conflicting again; this time, however, C_5 would require conflict analysis.

289 **Propagation in LSCB.** When falsifying a watched literal, Algorithm 1 might need to find
 290 a replacement candidate to become the new watched literal (line 7 of Algorithm 2). We
 291 define the property of the candidate literal with information about its level as below.

292 ► **Definition 11** (Candidate literal). *Let clause C be watched by the literals c_1 and c_2 . with*
 293 *$\neg c_1 \in \omega$. Then, $\text{SEARCHREPLACEMENT}(C, c_1, c_2)$ from Algorithm 2 returns a candidate*
 294 *literal r for which one of the following holds:*

- 295 ■ *Invariant 6 is satisfied on C after $\neg c_1$ is added to τ , i.e.*
 296 $\neg r \in (\tau \cdot \neg c_1) \Rightarrow c_2 \in \pi \wedge \delta(c_2) \leq \delta(r)$;
- 297 ■ *C is conflicting, propagating, or a MLI for c_2 . As such, $C \setminus \{c_2\}$ is unsatisfiable with the*
 298 *current assignment, and r is at the highest decision level in $C \setminus \{c_2\}$, that is*
 299 $(C \setminus \{c_2\} \wedge \pi) \models \perp \quad \wedge \quad \delta(r) = \delta(C \setminus \{c_2\})$

■ **Algorithm 1** Lazy Reimplication in CDCL with CB

```

1:  $\pi = \tau = \omega = \pi^d = \emptyset$ 
2:  $\forall \ell. \delta(\ell) = \infty$ 
3:  $\forall \ell. \text{WL}(\ell) = \emptyset$ 
4:  $\forall \ell. \rho(\ell) = \lambda(\ell) = \blacksquare$ 
5: procedure CDCL( $F$ )
6:   for  $C \in F$  do                                ▷ Fill the watcher lists
7:      $c_1, c_2 \leftarrow$  two literals in  $C$ 
8:      $\text{WL} \leftarrow \text{WL}[c_1 \leftarrow \text{WL}(c_1) \cup \{C\}][c_2 \leftarrow \text{WL}(c_2) \cup \{C\}]$ 
9:   while  $\top$  do
10:     $C \leftarrow \text{BCP}()$                                 ▷ Algorithm 2
11:    if  $C = \top$  then
12:      if  $|\pi| = |\mathcal{V}|$  then                            ▷ All variables are assigned
13:        return SAT
14:       $\ell \leftarrow \text{DECIDE}()$ 
15:       $\omega \leftarrow \omega \cdot \ell, \pi^d \leftarrow \pi^d \cdot \ell, \delta \leftarrow \delta[\ell \leftarrow |\pi^d|]$ 
16:      continue
17:     $D \leftarrow \text{ANALYZE}(C)$                             ▷ Algorithm 4
18:    if  $\delta(D) = 0$  then
19:      return UNSAT
20:     $d \leftarrow$  any level between  $\delta(D) - 1$  and the second highest level of  $D$ 
21:     $\text{BACKTRACK}(d)$                                     ▷ Algorithm 3
22:     $\ell \leftarrow$  the unassigned literals in  $D$ 
23:     $c_2 \leftarrow$  the second highest literal in  $D$ 
24:     $\omega \leftarrow \omega \cdot \ell, \delta \leftarrow \delta[\ell \leftarrow \delta(C \setminus \{\ell\}), \rho \leftarrow \rho[\ell \leftarrow D]$ 
25:     $F \leftarrow F \cup \{D\}$                             ▷ Does nothing if  $C = D$ 
26:     $\text{WL} \leftarrow \text{WL}[\ell \leftarrow \text{WL}(\ell) \cup \{D\}][c_2 \leftarrow \text{WL}(c_2) \cup \{D\}]$ 

```

300 Concretely, the $\text{SEARCHREPLACEMENT}(C, c_1, c_2)$ procedure iterates over literals of $C \setminus$
301 $\{c_2\}$ and stops when it finds a literal r that would satisfy Invariant 6 if c_1 was replaced by
302 r . In case of failure, it returns the highest literal in $C \setminus \{c_2\}$. The knowledge of the highest
303 literal in $C \setminus \{c_2\}$ is enough to determine the nature and level of the clause.

304 Algorithm 2 shows our Boolean constraint propagation (BCP) algorithm adapted to
305 support LSCB. As opposed to standard BCP, Algorithm 2 does not stop when the other
306 watched literal is satisfied. We need the extra guarantee that either c_2 is implied at a level
307 lower than c_1 , or it is registered as a MLI before skipping the clause. Further, when a
308 non-falsified replacement literal cannot be found, Algorithm 2 still changes the watched
309 literal. While this is not always strictly necessary (for example, in conflicts), systematically
310 swapping the highest literal allows checking the level of the clause in constant time and
311 provides cheap useful properties to the clause.

312 **Backtracking in LSCB.** When backtracking, our LSCB approach has the information of
313 whether a clause C violates Invariant 4. Therefore, Algorithm 3 can directly imply those
314 missed lower implications (line 15 of Algorithm 3).

315 The order in which literals are reimplicated in Algorithm 3 is not important, as shown later
316 in Theorem 17. It is, however, unclear whether a specific order would impact performance
317 in problems where the stability of literal position in the trail is important. In such cases,
318 ordering the reimplications in increasing levels might be beneficial.

Algorithm 2 Boolean Constraint Propagation in LSCB

```

1: procedure PROPAGATELITERAL( $\ell$ )
2:    $c_1 \leftarrow \neg \ell$ 
3:   for  $C \in \text{WL}[c_1]$  do
4:      $c_2 \leftarrow$  the other watched literal in  $C$ 
5:     if  $c_2 \in \pi \wedge [\delta(c_2) \leq \delta(c_1) \vee \delta(\lambda(c_2) \setminus \{c_2\}) \leq \delta(c_1)]$  then
6:       continue
7:      $r \leftarrow \text{SEARCHREPLACEMENT}(C, c_1, c_2)$ 
8:      $\text{WL} \leftarrow \text{WL}[c_1 \leftarrow \text{WL}(c_1) \setminus \{C\}][r \leftarrow \text{WL}(r) \cup \{C\}]$ 
9:     if  $\neg r \notin \pi$  then
10:      continue
11:     if  $\neg c_2 \in \pi$  then ▷ Conflict
12:      return  $C$ 
13:     if  $c_2 \in \pi$  then
14:       if  $\delta(c_2) > \delta(r) \wedge \delta(\lambda(c_2) \setminus \{c_2\}) > \delta(r)$  then
15:          $\lambda \leftarrow \lambda[c_2 \leftarrow C]$  ▷ New or improved MLI
16:       continue
17:      $\omega \leftarrow \omega \cdot c_2, \rho \leftarrow \rho[c_2 \leftarrow C], \delta \leftarrow \delta[c_2 \leftarrow \delta(r)]$ 
18:   return  $\top$ 

1: procedure BCP
2:   while  $\omega \neq \emptyset$  do
3:      $\ell \leftarrow \text{FIRST}(\omega)$ 
4:      $C \leftarrow \text{PROPAGATELITERAL}(\ell)$ 
5:     if  $C \neq \top$  then
6:       return  $C$ 
7:      $\omega \leftarrow \omega \setminus \{\ell\}, \tau \leftarrow \tau \cdot \ell$ 
8:   return  $\top$ 

```

319 **Conflict analysis with MLI.** As opposed to traditional backtracking, Algorithm 1 does
320 not guarantee that, once it backtracks to a level lower than the level of the learned clause D ,
321 the clause D will be propagating. Indeed, let the falsified learned clause $D = \{c_1, c_2, \dots, c_m\}$
322 with c_1 a unique literal at level $\delta(D)$. If we backtrack to level $\delta(D) - 1$, c_1 might be reimplicated
323 at a lower level, and D would still be a conflict. In response to this, we propose the following
324 two solutions:

325 **(Analyze-1)** we analyze the conflict and backtrack again until we get a unit clause;

326 **(Analyze-2)** we perform conflict analysis with the knowledge of missed lower implications.
327 In Algorithm 4 we chose option 2. Option 1 will generate the same clause in the end, but
328 might create some unnecessary ones in the process. We empirically check our intuition in
329 Section 6 and demonstrate that option 2 indeed works better. We refer to $D \otimes_{\ell} C'$ as the
330 result of binary resolution applied to the clauses C and D over the literal ℓ .

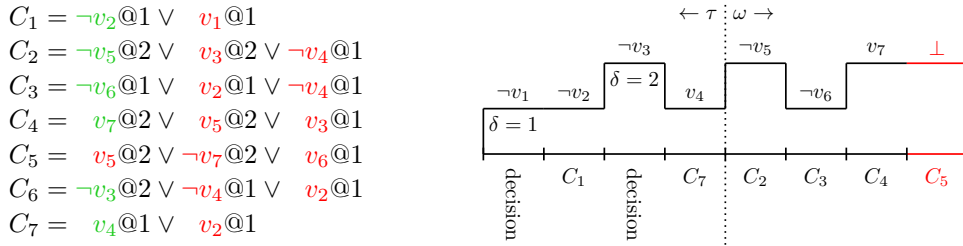
331 In Algorithm 4, when possible, we use the lazy reimplication reason $\lambda(\ell)$ instead of the
332 real reason $\rho(\ell)$ during conflict analysis. The lazy reason $\lambda(\ell)$ is guaranteed to introduce
333 literals at a level lower than $\delta(C)$, making it converge to a UIP faster. Once a UIP is obtained,
334 Algorithm 4 does not stop if there exists a missed lower implication for the last literal at
335 the conflict level. Furthermore, we adapted the learnt clause minimization approach [22],
336 adjusted to Algorithm 4 so that both reasons are checked if the literal can be removed.

■ **Algorithm 3** Backtracking and Reimplication.

```

1: procedure BACKTRACK( $d$ )
2:    $\Lambda \leftarrow \emptyset$                                 ▷  $\Lambda$  is the set that will be reimplied
3:    $\pi = \tau \cdot \omega$ 
4:   for  $\ell \in \pi$  do
5:     if  $\delta(\ell) > d$  then
6:       if  $\delta(\lambda(\ell) \setminus \{\ell\}) \leq d$  then
7:          $\Lambda \leftarrow \Lambda \cup \{\lambda(\ell)\}$           ▷ Store the MLI for later
8:          $\pi \leftarrow \pi \setminus \{\ell\}$                 ▷ Unassign  $\ell$ 
9:          $\delta \leftarrow \delta[\ell \leftarrow \infty], \rho \leftarrow \rho[\ell \leftarrow \blacksquare]$ 
10:         $\lambda \leftarrow \lambda[\ell \leftarrow \blacksquare]$         ▷  $\lambda(\ell)$  is either used, or no longer valid
11:    $\pi^d \leftarrow \pi \cap \pi^d$                         ▷ Remove the unassigned literals
12:    $\tau \leftarrow \pi \cap \tau$ 
13:    $\omega \leftarrow \pi \setminus \tau$ 
14:   for  $C \in \Lambda$  do                                ▷ Reimplying the MLI
15:      $\ell \leftarrow$  the unassigned literal in  $C$ 
16:      $\omega \leftarrow \omega \cdot \ell, \rho \leftarrow \rho[\ell \leftarrow C], \delta \leftarrow \delta[\ell \leftarrow \delta(C \setminus \{\ell\})]$ 

```



■ **Figure 4** The clause $\neg v_3 @2 \vee \neg v_4 @1 \vee v_2 @1$ is a missed lower implication in this example. v_2 and $\neg v_4$ are falsified at level 1, whereas $\neg v_3$ is only satisfied at level 2.

337 ▶ **Example 12.** Figure 4 shows a conflict after Algorithm 2 detected a missed lower implication
338 C_6 . From Invariant 9, we have $\lambda(\neg v_3) = C_6$. Algorithm 1 will then trigger Algorithm 4
339 to analyse the conflict on C_5 . During conflict analysis with Algorithm 4, we start from the
340 conflicting clause $D = \neg v_7 @2 \vee v_5 @2 \vee v_6 @1$ and apply the resolution $D \leftarrow D \otimes_{\neg v_7} C_4$ to
341 obtain $D = v_5 @2 \vee v_3 @2 \vee v_6 @1$. We once again apply resolution and have $D \leftarrow D \otimes_{v_5} C_2$,
342 yielding the clause $D = v_3 @2 \vee v_6 @1 \vee \neg v_4 @1$. As this D is a UIP, most CDCL approaches
343 would stop conflict analysis here. However, in our LSCB approach we know that v_3 can
344 be reimplied at level 1. Therefore, after backtracking to level 1 and reimplying $\neg v_3$ with
345 Algorithm 3, the clause D would still be conflicting and conflict analysis would need to be
346 triggered again. Instead we apply the resolution $D \leftarrow D \otimes_{v_3} C_6$ to get a clause at level 1,
347 namely clause $D = v_6 @1 \vee \neg v_4 @1 \vee v_2 @1$. We then continue until the procedure at level 1
348 and obtain the final clause $D = v_2 @1$.

24:12 Lazy Reimplication in Chronological Backtracking

Algorithm 4 Conflict Analysis.

```

1: procedure ANALYZE( $C$ )
2:    $\pi \leftarrow \tau \cdot \omega$  ▷ Array version of the trail.
3:    $D \leftarrow C$  ▷ Current learned clause.
4:    $n \leftarrow |\{\ell : \ell \in D \wedge \delta(\ell) = \delta(D)\}|$  ▷ Number of literals at the highest level.
5:   while  $\top$  do
6:      $\ell \leftarrow$  the last literal in  $\pi$  falsified in  $D$  at level  $\delta(D)$ 
7:     if  $n = 1 \wedge \lambda(\ell) = \blacksquare$  then
8:       return  $D$ 
9:      $C' \leftarrow \rho(\ell)$ 
10:    if  $\lambda(\ell) \neq \blacksquare$  then
11:       $C' \leftarrow \lambda(\ell)$ 
12:     $D \leftarrow D \otimes_{\ell} C'$ 
13:     $n \leftarrow |\{\ell : \ell \in D \wedge \delta(\ell) = \delta(D)\}|$ 

```

5 Soundness of Lazy Reimplication

This section proves the soundness and completeness² of our LSCB approach given in Algorithm 1. We note that Algorithm 1 implements strong chronological backtracking and does not miss any implication; as such, Invariant 4 holds.

► **Theorem 13** (Soundness of conflict analysis). *Let $C \in F$ be a conflicting clause with the partial assignment π . Then, conflict analysis in ANALYZE(C) from Algorithm 1 returns a conflicting clause that is implied by the clause set.*

Proof. The starting clause $D \leftarrow C$ is conflicting. At each step, D is resolved with a clause C' such that $C' = \rho(\ell)$ or $C' = \lambda(\ell)$, with $\ell \in C'$ and $\neg\ell \in D$. From the definition of ρ and λ , we have $(C' \setminus \{\ell\} \wedge \pi) \models \perp$. Therefore, the clause $D \leftarrow D \otimes_{\ell} C'$ is conflicting, and implied by F , since $C' \in F$. ◀

► **Theorem 14** (No missed unit implication). *Algorithm 1 satisfies Invariant 9. As such, our LSCB method in Algorithm 1 does not miss unit implications.*

Proof. We prove that Invariant 9 holds for each building block of Algorithm 1.

BCP. Invariant 9 trivially holds at the starting state, where $\pi = \emptyset$. Further, during the propagation of one literal, Algorithm 2 ensures that for each clause $C \in F$ watched by c_1 and c_2 , the following Hoare triple holds

$$\{P\} \text{PROPAGATELITERAL}(\ell) \{Q\},$$

where

$$P \equiv \neg c_1 \in \tau \Rightarrow [c_2 \in \pi \wedge [\delta(c_2) \leq \delta(c_1) \vee \delta(\lambda(c_2) \setminus \{c_2\}) \leq \delta(c_1)]]$$

$$Q \equiv \neg c_1 \in (\tau \cdot \ell) \Rightarrow [c_2 \in \pi \wedge [\delta(c_2) \leq \delta(c_1) \vee \delta(\lambda(c_2) \setminus \{c_2\}) \leq \delta(c_1)]]$$

By structural induction over the statements of Algorithm 2, we conclude that Invariant 9 is maintained by BCP.

² with details also in the code base of NAPSAT

368 **Backtracking.** During backtracking in Algorithm 3, each literal c_i is inspected: c_i is either
 369 removed from the trail π or c_i is kept. Violating Invariant 9 means that a literal c_2 from the
 370 trail is removed such that $\neg c_1 \in \tau \wedge c_2 \notin \pi$ for some clause $C = \{c_1, c_2, \dots, c_m\}$ (since the
 371 levels are not altered). However, this case is rectified, since either $\delta(c_1) \leq \delta(c_2)$ (and then
 372 $\neg c_1$ would be removed from τ), or $\delta(\lambda(c_2) \setminus \{c_2\}) \leq \delta(c_1)$ (and then c_2 would be reimplied at
 373 level $\delta(\lambda(c_2) \setminus \{c_2\})$ and $c_2 \in \pi \wedge \delta(c_2) \leq \delta(c_1)$ would be true), or $\delta(\lambda(c_2) \setminus \{c_2\}) > \delta(c_1)$ (and
 374 then $\neg c_1$ is also backtracked). As such, backtracking in Algorithm 3 preserves Invariant 9.

375 **Analysis.** Within conflict analysis in Algorithm 4, the state of CDCL is not modified, only
 376 read. Therefore, any invariant that held before Algorithm 4 also holds after Algorithm 4.

377 **CDCL.** We finally ensure that Invariant 9 is maintained by Algorithm 1 also during its
 378 decision step and while adding a clause to the formula F . First, deciding still preserves
 379 Invariant 9, since it merely adds a non-assigned literal to the propagation queue ω . Second,
 380 after backtracking in Algorithm 1, we know by construction that the learned clause will
 381 have a single literal ℓ that is unassigned. This literal ℓ is then implied at level $\delta(D \setminus \{\ell\})$,
 382 satisfying Invariant 9 since the second watched literal c_2 is falsified at level $\delta(D \setminus \{\ell\})$. ◀

383 ▶ **Corollary 15** (No missed conflict/implication). *Our LSCB method from Algorithm 1 pre-*
 384 *serves the strong watched literal property of Invariant 4.*

385 Based on the results above, we conclude the soundness and completeness of LSCB.

386 ▶ **Theorem 16** (LSCB soundness and completeness). *Lazy reimplication with strong chrono-*
 387 *logical backtracking from Algorithm 1 is sound and complete.*

388 **Proof.** Theorem 13 implies that clauses added to the clause set are implied by F . By
 389 induction, if ϕ is the original CNF, then if $\phi \models F$ and $F \models C$, then $\phi \models F \cup \{C\}$.
 390 Furthermore, from Corollary 15 we conclude that Invariant 4 holds.

391 Algorithm 1 returns unsat iff there exists a conflict at level 0; that is, there exists a set
 392 of clauses $F' \subseteq F$ such that $F' \models \perp$. As $\phi \models F$, then $\phi \models \perp$, and thus ϕ is unsatisfiable.
 393 Otherwise, Algorithm 1 returns SAT if a model τ exists such that every variable has been
 394 assigned and propagated ($\pi = \tau$). Based on Invariant 4, no conflict is possible and $\phi \models \tau$. ◀

395 ▶ **Theorem 17** (Topological order in LSCB). *The literals reimplied by the backtracking proce-*
 396 *cedure of Algorithm 3 respect the topological order of the implication graph.*

397 **Proof.** The reimplied literals cannot depend on each other. Indeed, if they are reimplied,
 398 their implication level before backtracking was higher than d . Therefore, if a literal ℓ depends
 399 on a literal ℓ' in the implication graph, then $\delta(\ell) \geq \delta(\ell')$. If the missed lower implication
 400 $\lambda(\ell)$ has a level lower than d , then all literals in $\lambda(\ell) \setminus \{\ell\}$ are lower than d , and therefore
 401 were not backtracked. Therefore, since all literals are independent, they can be reimplied in
 402 an arbitrary order at the end of the trail, and still respect the topological order. ◀

403 **6 Empirical Analysis**

404 In this section, we discuss the implementation of Algorithm 1 in our new SAT solver NAPSAT.
 405 We also integrated it in CADICAL and GLUCOSE, and present our empirical results using
 406 NAPSAT, CADICAL, and GLUCOSE.

407 **6.1 NapSAT for Lazy Reimplication in CDCL**

408 We implemented our LSCB method from Algorithm 1 in the new SAT solver NAPSAT. Our
 409 NAPSAT tool is a CDCL solver using the watcher list scheme [16] with blocker literals [9].
 410 NAPSAT supports the backtracking variants of NCB, WCB, RSCB, and LSCB at runtime.
 411 In chronological backtracking, the backtracking scheme is purely chronological, that is,
 412 NAPSAT always backtracks to one level before the conflict (unlike CADICAL). NAPSAT
 413 uses the VSIDS decision heuristic [16] with the agility restart strategy [3] and root-level
 414 clause elimination [7]. NAPSAT is available at <https://github.com/RobCoutel/NapSAT>
 415 and consists in a total of ~ 5.800 loc, among which the core of the solver represents ~ 1.500 loc.

416 **Blocker literals in NapSAT.** Blocker literals are useful to reduce the number of pointer
 417 dereferencing of the literal pointer [9]. If the blocker b is assigned at a level higher than the
 418 literal ℓ being falsified, then it might get backtracked before ℓ and a conflict might be missed.
 419 Invariant 9 can therefore be weakened, while still ensuring that no unit implication is missed.
 420

421 **► Invariant 18** (Lazy backtrack compatible watched literals with blocker literals). *For each*
 422 *clause $C \in F$ watched by the two distinct literals c_1, c_2 and with blocker b , we have*

$$423 \quad \neg c_1 \in \tau \Rightarrow \left(c_2 \in \pi \wedge (\delta(c_2) \leq \delta(c_1) \vee \delta(\lambda(c_2) \setminus \{c_2\}) \leq \delta(c_1)) \right) \\
 424 \quad \vee \left(b \in \pi \wedge \delta(b) \leq \delta(c_1) \right)$$

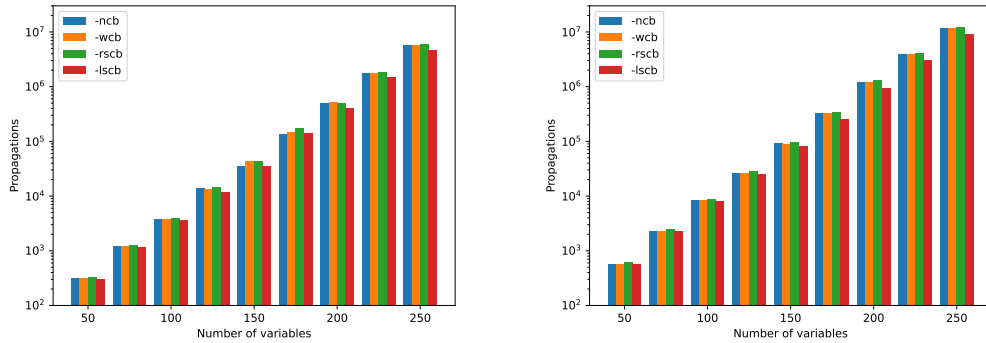
425 The eager update of blocking literals is in essence similar to strategies that aggressively
 426 update watched literals during BCP [14].

427 **Experiments.** Figure 5 shows the average total number of propagations of NAPSAT on the
 428 3-SAT uniform random problems from SATLIB [13]. Our LSCB method from Algorithm 1,
 429 indicated via `-lscb`, performs better than the other backtracking versions of NAPSAT, both
 430 for satisfiable and unsatisfiable instances. Figure 6 shows more details. In particular, it
 431 shows the total number of propagations of each unsatisfiable problem with 250 variables. It
 432 shows that LSCB consistently has fewer propagations than NCB, WCB, and RSCB.

433 We acknowledge that the number of propagations alone is not always representative of the
 434 real performance of a SAT solver, since propagation in LSCB is slightly more expensive than
 435 in NCB or WCB. However, the number of propagations in NAPSAT indicates the impact of
 436 missed lower implications. For example, Figure 5 shows that restoring the trail with RSCB
 437 might not be worth finding the missed lower implications in the random 3-SAT benchmarks;
 438 yet, reimplying literals lazily significantly reduces the total number of propagations.

439 **6.2 Integration of LSCB in CaDiCaL and Glucose**

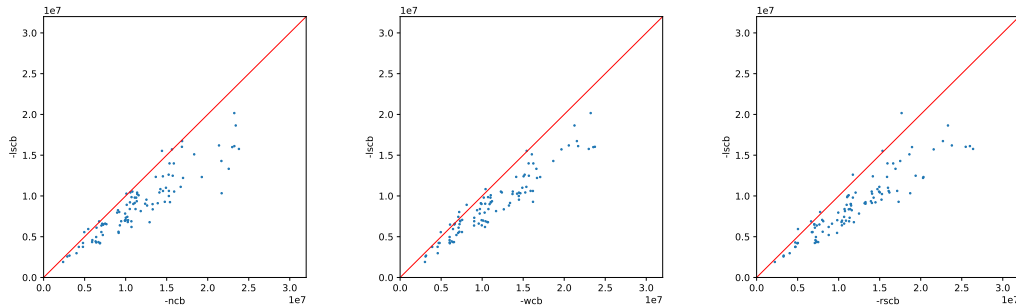
440 **LSCB in CaDiCaL.** We implemented our LSCB approach from Algorithm 1 in CADICAL
 441 [4], the baseline solver of the hack track of the SAT Competition. Thanks to the built-in
 442 model-based tester MOBICAL, the most effort came with ensuring that we have implemented
 443 correctly Invariant 4: CADICAL does not require watching the literals of two highest levels



(a) Satisfiable instances

(b) Unsatisfiable instances

■ **Figure 5** Average total number of propagations performed by NAPSAT on the SATLIB 3-SAT random problem, clustered by the number of variables, and backtracking technique employed.



(a) LSCB vs. NCB

(b) LSCB vs. WCB

(c) LSCB vs. RSCB

■ **Figure 6** Total number of propagations of NAPSAT for each unsatisfiable Uniform Random 3-SAT problem from SATLIB. The red line is the equality line. Marks under the equality line favour our new approach.

444 when the clause is propagating. This, however, requires iterating over the clause to find the
 445 propagation level, which we do not need.

446 We remark that we did not change the default backjumping policy of CADICAL: when
 447 backjumping from more than 100 levels occurs (following the value implemented in CADICAL),
 448 we resort to backtracking (going one level back). Otherwise, an algorithm similar to
 449 trail reuse for restarts is used to decide how many levels should be kept. Unlike the version
 450 implemented in NAPSAT, in CADICAL we store the missed level instead of checking the
 451 level of the MLI each time we need the level.

452 We tested various configurations, as summarized in Table 2, on the bwForCluster Helix with
 453 AMD Milan EPYC 7513 CPUs, using a memory limit of 16 GB RAM on the problems from
 454 the SAT Competition. Overall, we can see there is little difference between the considered
 455 configurations. In particular, the performance difference between WCB and NCB is limited,
 456 making it unclear if chronological backtracking is important. However, similar to the original
 457 CADICAL implementation [15], on the benchmarks from the SAT Competition 2018, there
 458 is an improvement from WCB over NCB. Our intuition is that chronological backtracking
 459 is especially useful when the decision heuristic is picking the wrong literals finding conflicts

24:16 Lazy Reimplication in Chronological Backtracking

■ **Table 2** Number of solved instances by different variants of strong backtracking on the SC2023 competition, using a 5.000s timeout

CADICAL version	solved	PAR-2 ($\times 10^3$)
base-CADICAL = RSCB	248	4.09
LSCB, Analyze-2 and minimization	246	4.16
ESCB	245	4.16
LSCB and Analyze-2	246	4.19
NCB	247	4.19
LSCB and Analyze-1	242	4.24

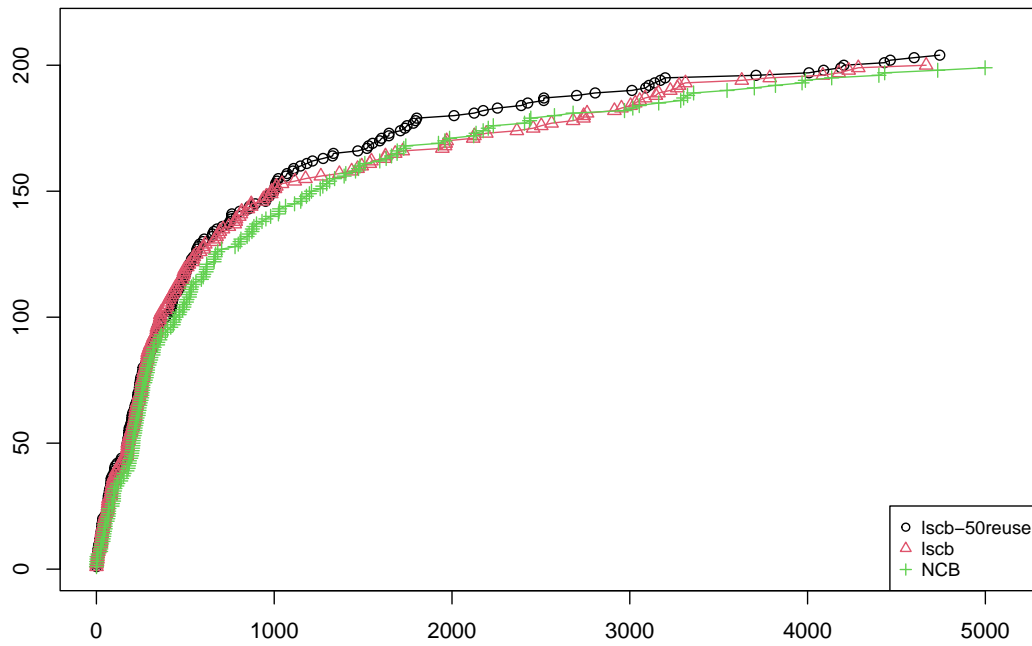
460 late instead of early (like finding a new unit at level 500 instead of level 1). The decision
461 heuristics seem to perform worse on the 2018 benchmarks, while this did not seem to have
462 happened since.

463 While the results in NAPSAT seem to indicate a large decrease in the number of
464 propagations, three factors mitigate this effect in CADICAL, as follows. (i) Propagating a
465 literal ℓ a second time as in RSCB is cheaper than propagating it for the first time. Most
466 clauses remaining in the watch list of ℓ will already be satisfied and are faster to check. (ii)
467 RSCB allows to use of blocking literals more loosely. There is no need to compare the level
468 of the blocking literal and the propagated one, making them more potent. (iii) Searching for
469 a replacement literal is slightly more expensive in LSCB since we need to record the highest
470 literal in the clause.

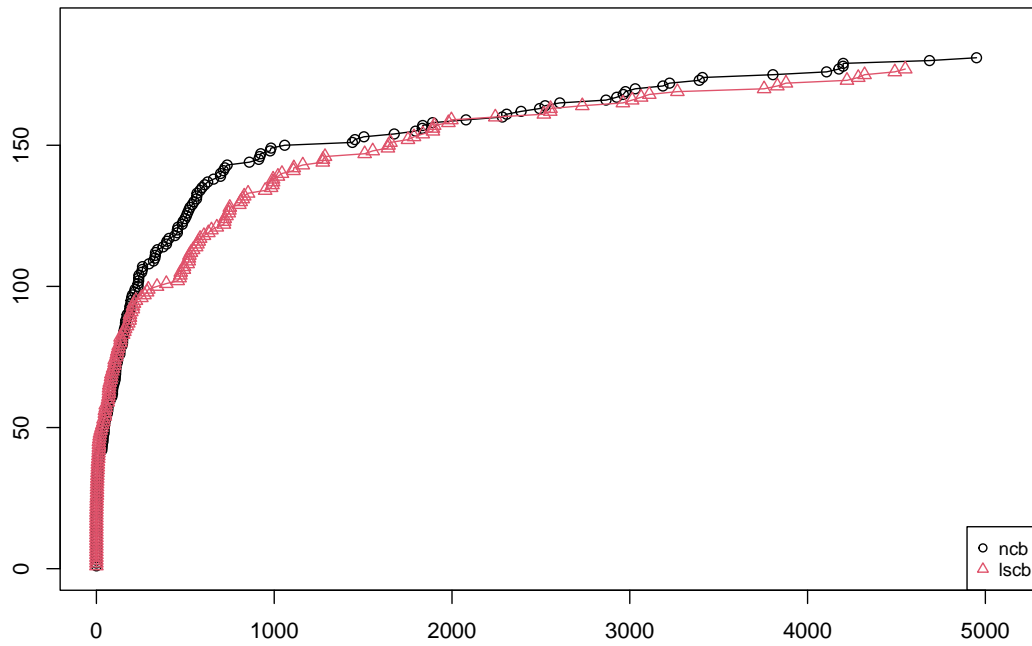
471 In a context where propagations are more expensive, such as SMT or user-propagators [8],
472 these considerations might weigh less on the overall performance of the solver. We will
473 investigate these applications for future work.

474 **LSCB in Glucose.** We also implemented our Algorithm 1 for LSCB into the latest version
475 of GLUCOSE [1], the SAT solver that pioneered the LBD heuristic for the usefulness of
476 clauses (only without the minimization part). This is the only solver where we implemented
477 the LSCB without any existing CB in the code. The entire diff (including new logging
478 information and more assertions) is less than 1.000 lines. Our actual implementation of
479 LSCB in GLUCOSE is very close to our abstract Algorithm 1, because the blocker literal is
480 always exactly the other watched literal. We use the simple heuristic to backtrack one level
481 if jumping back more than 100, otherwise use the normal backjumping. We did not change
482 the heuristic to block restarts [2], which is based on the trail length.

483 While running GLUCOSE with LSCB on the SAT Competition 2023 (Fig. 7b), we ob-
484 served worse performance. Interestingly, this is mostly due to one family of benchmarks,
485 SC23_Timetable, that perform much worse with strong backtracking (but are solved even-
486 tually). On the 2018 benchmarks again (Fig. 7a), we observed a slight performance improve-
487 ment when using GLUCOSE with LSCB and it seems to be better to trigger chronological
488 backtracking more often.



(a) GLUCOSE variants in the SAT Competition 2018



(b) GLUCOSE variants in the SAT Competition 2023

■ **Figure 7** CDF of the different GLUCOSE (without strategy adapting) versions. The constant indicates when chronological backtracking is triggered instead of backjumping: We apply chronological backtracking when NCB would require jumping back more than 100 levels by default, like in CADICAL. In the SAT Competition 2018, the version that triggers chronological backtracking for more than 50 levels performs best.

489 **7 Conclusion**

490 We introduce a lazy reimplication procedure to be used in CDCL with (variants of) chronologi-
 491 cal backtracking. We particularly focus on the definitions of weak chronological backtracking
 492 (WCB), restoring strong chronological backtracking (RSCB), eager strong chronological
 493 backtracking (ESCB), and lazy strong chronological backtracking (LSCB). Our invariant
 494 properties on these backtracking variants exploit watched literals. We prove that our ap-
 495 proach of lazy reimplication in strong chronological backtracking (LSCB) yields a sound
 496 and complete SAT solving method. Our implementation in NAPSAT, and its integration
 497 with CADICAL and GLUCOSE, gives practical evidence that LSCB is significantly easier
 498 to implement than ESCB, while also propagating fewer literals than RSCB, and providing
 499 better guarantees than WCB.

500 In the future, we intend to extend our LSCB method to reason over virtual literal levels,
 501 that is, levels of missed lower implications if such a clause is detected. We believe such an
 502 extension would allow to converge closer to the guarantees of ESCB, while mitigating both
 503 algorithmic complexities and reimplication costs. Further, we will explore the integration of
 504 chronological backtracking variants in the context of SMT, as a robust approach to handling
 505 arbitrary incremental clauses and expensive theory propagations.

506 **References**

-
- 507 1 Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT
 508 Solvers. In *IJCAI*, pages 399–404, 2009. URL: [http://ijcai.org/Proceedings/09/Papers/
 509 074.pdf](http://ijcai.org/Proceedings/09/Papers/074.pdf).
- 510 2 Gilles Audemard and Laurent Simon. On the Glucose SAT solver. *Int. J. Artif. Intell.*
 511 *Tools*, 27(1):1840001:1–1840001:25, 2018. doi:10.1142/S0218213018400018.
- 512 3 Armin Biere. Adaptive Restart Strategies for Conflict Driven SAT Solvers. In *SAT*, volume
 513 4996 of *LNCS*, pages 28–33. Springer, 2008. doi:10.1007/978-3-540-79719-7_4.
- 514 4 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt.
 515 CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editor, *Computer Aided Verification -
 516 36th International Conference, CAV 2024, Paris, France, July 24-27, 2024*, LNCS. Springer,
 517 2024. To appear.
- 518 5 Armin Biere, Mathias Fleury, and Florian Pollitt. CaDiCaL_vivinst, IsaSAT, Gimsatul,
 519 Kissat, and TabularaSAT entering the SAT competition 2023. In *SAT Competition 2023
 520 – Solver and Benchmark Descriptions*, volume B-2023-1 of *Department of Computer Science
 521 Report Series B*, pages 14–15. University of Helsinki, 2023.
- 522 6 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satis-
 523 fiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*.
 524 IOS Press, 2021. doi:10.3233/FAIA336.
- 525 7 Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in SAT solving. In Armin
 526 Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satis-
 527 fiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 391 – 435.
 528 IOS Press, 2nd edition edition, 2021.
- 529 8 Nikolaj S. Bjørner, Clemens Eisenhofer, and Laura Kovács. User-propagators for custom
 530 theories in SMT solving. In David Déharbe and Antti E. J. Hyvärinen, editors, *Pro-
 531 ceedings of the 20th Internal Workshop on Satisfiability Modulo Theories co-located with
 532 the 11th International Joint Conference on Automated Reasoning (IJCAR 2022) part of
 533 the 8th Federated Logic Conference (FLoC 2022), Haifa, Israel, August 11-12, 2022*, vol-
 534 ume 3185 of *CEUR Workshop Proceedings*, pages 71–79. CEUR-WS.org, 2022. URL:
 535 <https://ceur-ws.org/Vol-3185/extended6630.pdf>.

- 536 9 Geoffrey Chu, Aaron Harwood, and Peter J. Stuckey. Cache Conscious Data Structures for
537 Boolean Satisfiability Solvers. *JSAT*, 6(1-3):99–120, 2009. URL: [https://satassociation.
538 org/jsat/index.php/jsat/article/view/71](https://satassociation.org/jsat/index.php/jsat/article/view/71).
- 539 10 Robin Coutelier et al. Chronological vs. Non-Chronological Backtracking in Satisfiability
540 Modulo Theories. Master’s thesis, Université de Liège, Liège, Belgique, 2023.
- 541 11 Robin Coutelier Pascal Fontaine. ModularIT solver. Accessed March 2024. URL: [https:
542 //gitlab.uliege.be/smt-modules/](https://gitlab.uliege.be/smt-modules/).
- 543 12 Randy Hickey and Fahiem Bacchus. Trail saving on backtrack. In Luca Pulina and Martina
544 Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International
545 Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in
546 Computer Science*, pages 46–61. Springer, 2020. doi:10.1007/978-3-030-51825-7_4.
- 547 13 Holger H Hoos and Thomas Stützle. SATLIB: An Online Resource for Research on SAT.
548 *Sat*, 2000:283–292, 2000.
- 549 14 Norbert Manthey. Watch Sat and LTO for CaDiCaL. In *SAT Competition 2023 – Solver and
550 Benchmark Descriptions*, volume B-2023-1 of *Department of Computer Science Report Series
551 B*, pages 10–11. University of Helsinki, 2023.
- 552 15 Sibylle Möhle and Armin Biere. Backing Backtracking. In *SAT*, volume 11628 of *LNCS*,
553 pages 250–266. Springer, 2019. doi:10.1007/978-3-030-24258-9_18.
- 554 16 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
555 Chaff: Engineering an Efficient SAT Solver. In *DAC*, pages 530–535. ACM, 2001. doi:
556 10.1145/378239.379017.
- 557 17 Alexander Nadel. Introducing Intel(R) SAT Solver. In *SAT*, volume 236 of *LIPICs*, pages
558 8:1–8:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.
559 SAT.2022.8.
- 560 18 Alexander Nadel and Vadim Ryvchin. Chronological Backtracking. In *SAT*, volume 10929 of
561 *LNCS*, pages 111–121. Springer, 2018. doi:10.1007/978-3-319-94144-8_7.
- 562 19 Florian Pollitt. Cadical 1.9.4. Accessed March 2024. URL: [https://github.com/
563 arminbiere/cadical/tree/reimply-branch](https://github.com/arminbiere/cadical/tree/reimply-branch).
- 564 20 Coutelier Robin. NapSAT solver. Accessed March 2024. URL: [https://github.com/
565 RobCoutel/NapSAT](https://github.com/RobCoutel/NapSAT).
- 566 21 João P. Marques Silva and Karem A. Sakallah. GRASP: A Search Algorithm for Propositional
567 Satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999. doi:10.1109/12.769433.
- 568 22 Niklas Sörensson and Armin Biere. Minimizing Learned Clauses. In *SAT*, volume 5584 of
569 *LNCS*, pages 237–243. Springer, 2009. doi:10.1007/978-3-642-02777-2_23.