Tuning SAT solvers for LTL Model Checking

Anissa Kheireddine
EPITA, LRE, Kremlin-Bicêtre
Sorbonne Université, CNRS, LIP6, Paris
France, anissa.kheireddine@lrde.epita.fr

Etienne Renault
EPITA, LRE, Kremlin-Bicêtre
France, renault@lrde.epita.fr

Souheib Baarir
Sorbonne Université, CNRS, LIP6, Paris
Université Paris Nanterre, Nanterre
Now at EPITA, LRE, Le Kremlin-Bicêtre France,
souheib.baarir@lip6.fr

***Abstract*—Bounded model checking (BMC) aims at checking whether a model satisfies a property. Most of the existing SAT-based BMC approaches rely on generic strategies, which are supposed to work for any SAT problem. The key idea defended in this paper is to tune SAT solvers algorithm using: (1) a static classification based on the variables used to encode the BMC into a Boolean formula; (2) and use the hierarchy of Manna&Pnueli [33] that classifies any property expressed through Linear-time Temporal Logic (LTL). By combining these two information with the classical Literal Block Distance (LBD) measure [46], we designed a new heuristic, well suited for solving BMC problems. In particular, our work identifies and exploits a new set of relevant (learnt) clauses. We experiment with these ideas by developing a tool dedicated for SAT-based LTL BMC solvers, called BSaLTic. Our experiments over a large database of BMC problems, show promising results. In particular, BSaLTic provides good performance on UNSAT problems. This work highlights the importance of considering the structure of the underlying problem in SAT procedures.**

***Index Terms*—Bounded model-checking, SAT, Structural information, Linear Temporal Logic, Optimization**

I. Introduction

Model checking [12] is an automated procedure that establishes the correctness of hardware and software systems ranging from the simple program that runs a microwave to the very complex software driving a nuclear power plant, passing by our smartphones and cars. Model checking is therefore very useful for eliminating bugs and increasing confidence in hardware designs and software products.

Usually, the program (model) at hand is expressed in a formal language (e.g., SMV [35], Verilog [37], Promela [25], etc.), while the property is expressed as a temporal logic formula (e.g., LTL [41], CTL [13], PSL [20], etc.). A property is said to be *verified* if no execution of the model can invalidate it, otherwise it is *violated*. To achieve this verification, two approaches have been considered: explicit model checking [26] and symbolic model checking [8], [10]. In explicit model checking, the behaviour of the system as well as the property are represented as automata: a Kripke structure for the system [2] and a Büchi automaton for the LTL property [49]. Then, a synchronous product is performed between the Kripke and the automaton of the (negated) property. If the product is empty, the property is verified, otherwise it is violated. The main drawback of this technique is the state-space explosion problem [14], i.e., the size of the system state-space grows exponentially. This problem can be tackled using symbolic model checking, that represents states implicitly using Boolean functions (e.g., BDD [8], SAT formula [5]).

Regardless of the used technique, properties are expressed using temporal logic. In this paper, we focus on the Linear-time Temporal Logic (LTL) since it is capable of expressing many properties of interest and has been extensively studied [1], [33], [39]. For instance, Manna&Pnueli [33] established a full hierarchy of the properties that can be expressed using LTL. This hierarchy has been exploited for tuning the performance of some model-checkers [18], [21], [40].

The objective of the paper is to improve the solving of BMC problems, as far as SAT techniques are concerned. This is realized through the exploitation of the structure of the BMC problem. Indeed, we rely on: (1) a characterization of the variables encoding the BMC problem as a Boolean (SAT) formula (Section III); (2) a categorization of the LTL properties w.r.t. to the hierarchy of Manna&Pnueli (Section IV). Sections V and VI give a preliminary study on the characteristics of BMC problems in the SAT context. This latter section introduces the framework BSaLTic[1] we developed to achieve this study. Section VII explains how the usage of these information can improve the global performance of modern SAT solvers. Experiments through BSaLTic are presented in Section VIII.

II. Related work

Most existing work focuses on building generic approaches for tuning SAT solver [23], [28], [36], [46], [52] and rarely exploits the information derived from the original problem. In this paper, we focus our study on BMC problems offering two main characteristics: the classification of clauses and the specificity of the studied LTL property. To our knowledge, none of these two elements have been studied together in this context. Our previous work [29] can be considered at the corner stone of this presented work. Regardless of the LTL specificity, this prior study [29] suggests a generic clause partitioning methodology and applied it in the case of BMC problems. This partitioning leads to new optimizations for the SAT solving to identify and protect interesting learnt clauses.

The work of [3] investigates the origin of each variables w.r.t. its unrolling depth. The authors found a correlation between these time steps and the communities found in the

[1]For a description of our setup, detailed results and code, see https://akheireddine.github.io/

CNF encoding. The study also demonstrates that the Literal Block Distance [46] measure used in almost all modern SAT solvers to identify good learnt clauses, is related to the unrolling steps measure. The authors tried a number of hacks in order to exploit the unrolling iterations of variables. The proposed heuristics were not fruitful such as: forcing variable elimination according to their unrolling iterations, shifting variables score, protecting clauses using a different metric than LBD. Still, this experimental study can complement our work and point out some unsuccessful results of the authors. Other BMC-based researches [44], [50], [51] present a variety of optimizations in the context of SAT-based BMC, such as: variable ordering heuristics, branching heuristics, studying the symmetry structure of the BMC formula. These works could be refined using the information exploited in this present paper.

Finally, the hierarchy of Manna&Pnueli [33] has been used to tune explicit model-checkers [18], [21], [40]. These studies suggested a decomposition of the input automaton or propose optimizations for specific classes of the hierarchy.

To the best of our knowledge, this paper is the first covering all the classes of this hierarchy in the context of SAT-based BMC. Most of the literature gives attention to safety or guarantee properties [43], [47] only, such as IC3 [7] or PDR [19] procedures. There exist some rigorous methods to convert liveness properties into safety properties [4] and thus IC3/PDR approaches can be applied. In this paper, we go further by specializing the verification procedure for each class of the hierarchy.

III. SAT-based Bounded Model Checking

Existing model-checking approaches [8], [10] suffer from the state-space explosion problem [14]. This issue is tackled using SAT-based BMC methods [6], [11], [22] which has a low memory footprint thanks to (1) the restrictions of the verification to sequences bounded by some integer k, and (2) the use of Boolean Satisfiability (SAT) techniques. Given a model M, an LTL property p (*after negation*), and a bound k, the SAT-based BMC approach builds a propositional formula representing the combination $(M \otimes p)$ of M and p, both unrolled up to k steps. The formula is said to be satisfiable iff there exists a violation of the property of maximum length k. Otherwise, it is unsatisfiable and the property is verified up to length k. Equation 1 depicts the encoding of the synchronized product into a Boolean formula.

$$\underbrace{\overbrace{I(s_0)}^{Initial\ stats} \overbrace{\bigwedge T(s_0,s_1) \bigwedge \cdots \bigwedge T(s_{k-1},s_k)}^{Transitions} \bigwedge}_{Model} \underbrace{P_k}_{Property} \quad (1)$$

This encoding [27], [48] translates a graph $(M \otimes p)$ to a set of constraints (*clauses*) that are a disjunction of variables. The final Boolean formula is represented in the form of a conjunction of clauses (CNF) and can be solved using state-of-the-art SAT solvers. Those solvers based on the Conflict Driven Clause Learning algorithm [34], [45], [52] are of particular interest in this work.

Conflict Driven Clause Learning (CDCL) [52]. This algorithm is one of the main methods used to solve Satisfiability problems and is an enhancement of the DPLL algorithm [15], [16]. CDCL algorithm performs a backtracking search, selecting at each node of the search tree, a decision variable that is set to a Boolean value. Each decision, called branching, is followed by an inference step that deduce and propagates forced variable assignments (a procedure called *unit-propagation*). This branching process is repeated until a model is found or a conflict is reached. In the first case, the formula is said to be *satisfiable*, and the model is reported. When a conflict is reached while no branching is active, the formula is *unsatisfiable*. Otherwise, a **learnt clause** is generated thanks to a procedure called *conflict-analysis* [34], [45].

Learnt clauses will avoid repeating the same mistake, and therefore allow faster deductions (during *conflicts analysis* and *unit-propagation* steps). Since the number of conflicts is huge (avg. 5000/s [46]), controlling the size of the database storing learnt clauses is a challenging task. It can dramatically affect the solver's performance. Many strategies have been proposed to manage the cleaning of the stored clauses. One of the state-of-the-art strategies uses the Literal Block Distance (LBD) measure [46].

IV. Manna & Pnueli Hierarchy

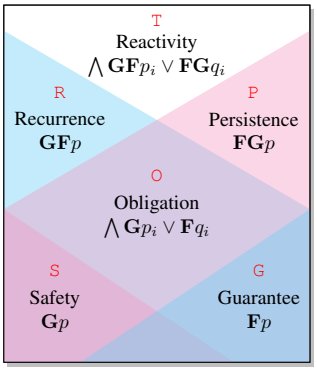


Fig. 1: Hierarchy of Manna&Pnueli with S ∪ G ⊆ O ⊆ R ∪ P ⊆ T

LTL is a temporal logic that specifies (infinite) sequences describing system behaviors. Lamport [30] partitioned the properties expressed by LTL into two classes: safety (*some bad thing never happens*) and liveness (*some good thing eventually happens*). In these classes, two main temporal operators are used: **F** (**F***inally something happens*) and **G** (**G***lobally something holds*). For example, one can specify some atomic proposition "$a = 42$" (e.g., variable a is equal to 42) that must holds at every point in time (i.e., **G** "$a = 42$") or eventually holds at some future point in time (i.e., **F** "$a = 42$").

Based on the combination of the two precedent operators, Manna&Pnueli [33] refined the Lamport's partitioning into six categories (depicted in Fig.**1**):

- *Safety* (S): similar to the one described in the Lamport [30] classification.

- *Guarantee* (G): some good thing happens at least once in the future.
- *Obligation* (O): combines safety and guarantee properties. This enforces more restrictions on the sequences leading to some good thing.
- *Persistence* (P): at some point, a good thing will happen and hold forever.
- *Recurrence* (R): some good things will appear infinitely often.
- *Reactivity* (T): combines recurrence and persistence properties. This enforces more restrictions on the sequences between good things.

Each one has inherent characteristics that can be exploited to tune the performance of model-checkers. To the best of our knowledge, only common classes (safety and guarantee) have been exploited to improve SAT-based BMC [43]. *In this work, we studied the entire hierarchy (S, G, O, P, R and T) in order to fine tune the SAT heuristics w.r.t the handled property.*

V. A PEEK INSIDE SAT-BASED BMC PROBLEMS

The main goal of this paper is to identify relevant information that can be used to enhance CDCL-based SAT solving of BMC problems. We identify two sources of information, *static information* and *dynamic information*:

- **Variable-based classification *(static)*:** when translating the high-level description of the problem into a SAT problem, we can partition the obtained variables into three **disjoint** sets: $\mathcal{M}$, $\mathcal{J}$ and $\mathcal{P}$ where $\mathcal{M}$ (model) is a Boolean representation of original variables of the system at hand, $\mathcal{J}$ (auxiliary variables) is a set of fresh variables used to finalize the conversion into a SAT formula, and $\mathcal{P}$ (property) the set of variables used to translate the Büchi automaton of the (negated) property.
  Hence, any clause in $\mathcal{C}$ (the set of clauses of the problem) can be classified according to the variables it handles: $C_X = \{\omega \in \mathcal{C} \,|\, \forall v \in \mathbf{Var}(\omega), v \in X\}$ where $\mathbf{Var}(\omega)$ is the set of variables forming the clause ω, X is either $\mathcal{P}$ (the property), $\mathcal{M}$ (the model), $\mathcal{J}$ (auxiliary variables), $\mathcal{PJ}$ (property and auxiliary variables), $\mathcal{PM}$ (property and model variables), $\mathcal{MJ}$ (model and auxiliary variables) or $\mathcal{PMJ}$ (property, model and junction variables). Note that the intersection between pairs of the seven classes C_X is empty.
- **LTL hierarchy *(static)*:** when solving a BMC problem, the LTL formula is known *a priori*. We can apply the syntactic characterization of Manna&Pnueli in order to compute its corresponding class in the hierarchy. This allows to propose a parametrization specific to each category of the hierarchy.
- **LBD score [46] *(dynamic)*:** It is a positive integer, used as a learnt clause quality metric (the less the better) in almost all competitive CDCL-like SAT solvers. The LBD of a clause can change over time and can be recomputed each time the clause is fully assigned.

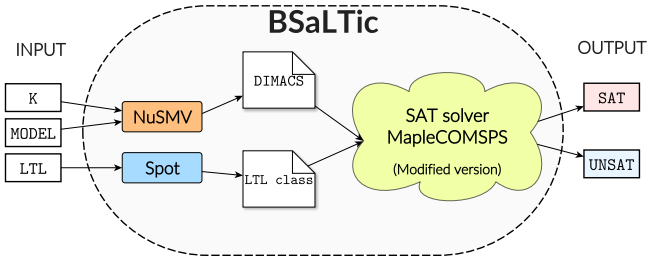


Fig. 2: BSaLTic's Framework. The dashed box represents the BSaLTic tool

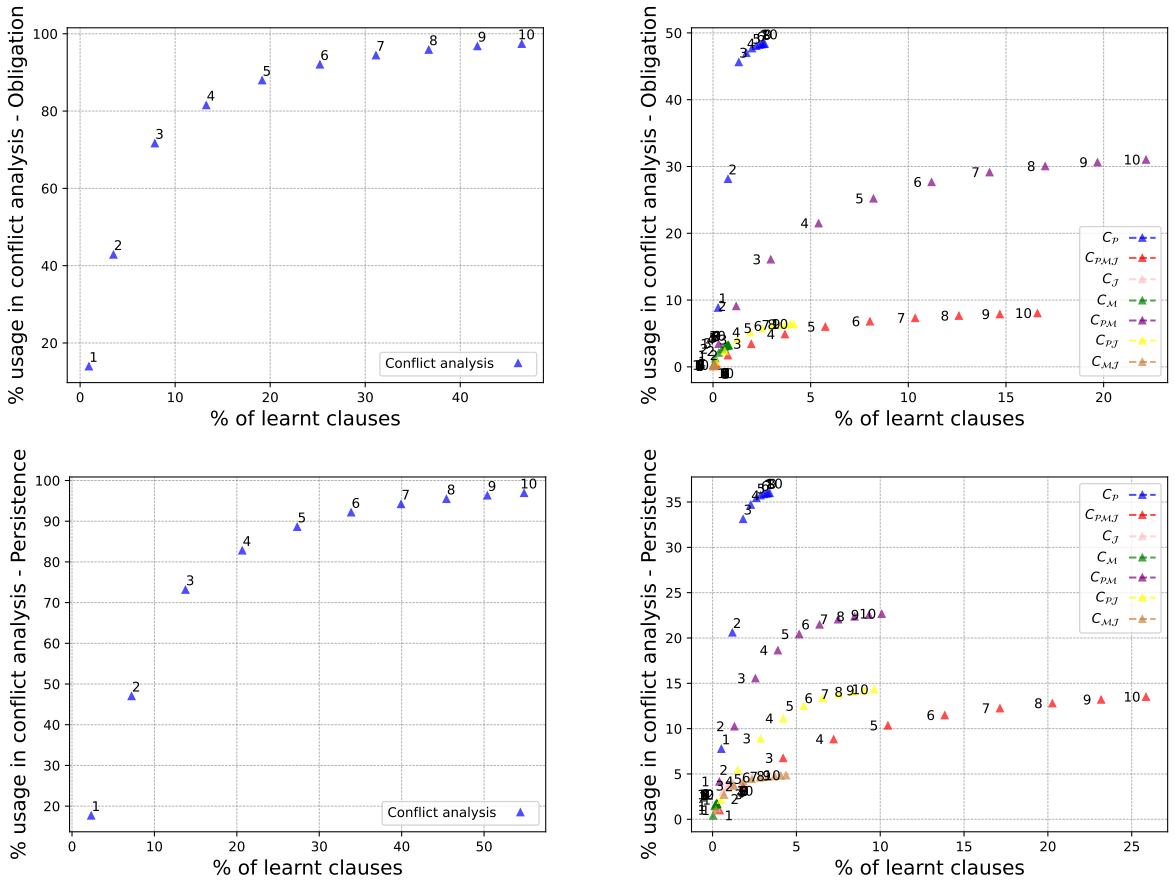
- **Learnt clauses database *(dynamic)*:** during CDCL solving, every time a new clause is generated, its LBD is computed. According to this value, the usefulness of the clause is guessed and a storing strategy is then applied. When considering MapleCOMSPS [32], the winner of the main track of the SAT competition 2016 and used as core engine for the best solvers in the last 5 years, it will promote the clause at hand into one of its three databases: ***core*** (LBD$\leq$3) for the really important clauses (never deleted), ***tier-2*** (LBD$\leq$6) for not-yet-decided clauses, and ***local*** database for the remaining clauses. Clauses in ***tier-2*** can be promoted to the ***core*** or downgraded to the ***local*** database whereas those of ***local*** database can either be promoted to ***tier-2*** or permanently deleted. Similarly, Kissat-MAB [42] solver, the winner of the SAT competition 2021, processes the same way by promoting clauses in three databases: ***core*** (LBD$\leq$2), ***tier-2*** (LBD$\leq$6) and ***local*** for the remain clauses.

In this paper we propose to combine and exploit the aforementioned static information and the LBD dynamic score in order to build a new heuristic that sharply adjusts the identification of interesting learnt clauses to protect (in the databases) for the SAT-based BMC solving.

VI. A STUDY OF BMC PROBLEMS USING BSALTIC

This section aims to analyze the relation between static BMC-based information and Dynamic CDCL-based information. The derived information will be exploited further in Section VII. We start by introducing the BSaLTic framework developed for our experiments. Then, we detail the benchmark setup before performing the analysis phase through BSaLTic.

Framework. To study the characteristics of BMC problems, we developed a tool called BSaLTic[1]. Fig.**2** shows the architecture of the framework which involves NuSMV [9], Spot [17] and MapleCOMSPS SAT solver [32]. BSaLTic takes three parameters as input: (1) an SMV program, (2) an LTL property (*not negated*), (3) and a bound k, required for any BMC problem. Spot is used to identify (syntactically) the class of LTL formula it belongs to (according to the hierarchy of Manna&Pnueli). In the other hand, NuSMV produces a file, representing the SAT encoding of the BMC problem at hand [27]. These two information are then processed by (our



(a) without classification of the clauses

(b) with classification of the clauses

Fig. 3: Measures on the *training benchmark* showing learnt clauses usage in *conflict-analysis* for Obligation problems (top) and Persistence problems (bottom). Each class is colored and annotated by its LBD value.

modified version of) MapleCOMSPS solver. Note that the SAT solver component is encapsulated inside an aggregator of solvers [31]. Thus it will be easy to integrate (later) other SAT solvers into the framework.

Benchmark setup. Our benchmark is composed of 1320 problems. All SMV instances (with their respective LTL properties) come from a variety of benchmarks: the HWMC Competition (2017[2] and 2020[3]), the hardware verification problems [9], the BEEM database [38], and from the RERS Challenge[4]. Some LTL properties have been generated using Spot [17] such that, each category of the Manna&Pnueli hierarchy would be equivalently represented (220 formula per category). We used various bound k for each SMV problem $k = \{20, 40, 60, ..., 4000, 6000\}$. We omitted trivial instances that runs less than 1 second on MapleCOMSPS solver.

Measures. To evaluate the importance of the information presented in Section V, we built a *training benchmark* for each category of LTL property, composed of 27% of the associated benchmark (60 problems per category). For each instance of this *training benchmark*, we logged the information related to each learnt clause when used in the *unit-propagation* and *conflict-analysis* phases with its corresponding LBD value (the training phase is done once and took 220 hours). Thus, the usage rate of each of the classes during these two procedures are computed by accumulating all the clauses of the same class and the same LBD value.

Fig.**3** depict these rates for Obligation and Persistence properties (the plots of the other classes are omitted for place constraints[1]). The x-axis reports the cumulative mean percentage of generated learnt clauses on the *training benchmark* and the y-axis corresponds to the cumulative mean usage percentage on *conflict-analysis* (for space constraints, we omitted the plots of *unit-propagation* step but it has similar patterns as the *conflict-analysis* one). Each point represents the used percentage of learnt clauses of a certain LBD (we only display LBD from 1 up to 10 to simplify the reading, but we went up to 22 in practice). Fig.**3b** displays the various classes of clauses. For example, on Obligation benchmark (top figure), the purple triangle with left annotation 3 shows that 2.9% of generated learnt clauses of class $\mathcal{PM}$, have an LBD$\leq$3 and

[2]http://fmv.jku.at/hwmcc17/
[3]http://fmv.jku.at/hwmcc20/
[4]RERS models translated in NuSMV: https://tinyurl.com/29a4jcme

are used in 16.0% during the *conflict-analysis* time.

From these figures, we can observe that $C_{\mathcal{P}}$ always stands out. Nonetheless, the importance of the remaining classes seem to vary according to the group of property that is checked.

VII. Contribution

The above study reveals that LBD could hide interesting information. For instance, the LBD value used to define the *core* database (LBD≤3) represents 8.1% of learnt clauses for 76.4% of usage. A closer look to Fig.**3** reveals that half of this usage came from the $C_{\mathcal{P}}$ class and represents only 1.3% of the learnt clauses.

The idea defended in this paper is: even if the generic characterization of learnt clauses with the LBD measure works efficiently for a majority of SAT problems, it can be sharply adjusted with structural information in the case of BMC problems. Our contribution offers an opening toward new ideas on the identification of good **selector** (a specific LBD value for each class of clauses according to the handled LTL property).

This selector identifies new sets of clauses that are relevant during the solving process. Typically, a selector would be of the form: protect clauses $C_{\mathcal{P}}$ with LBD≤8, $C_{\mathcal{J}}$ with LBD≤4, etc.

We present a new heuristic, called $\mathrm{H}_{\mathcal{F}}$ (computed thanks to linear programming system) for building selectors of interest, depending on the information collected during the training phase. The proposed approach does not rely on a specific SAT solver and can be implemented on top of any CDCL-based SAT solvers. The clauses identified by this selector can be protected in different manners. In this work, we propose two protections strategies (see Sections VII-B and VII-C).

A. Frequency-based heuristic $\mathrm{H}_{\mathcal{F}}$

This section introduces a way of finding interesting selectors based on the usage frequency of a class $i \in X$ (see Section VI), namely $f = \frac{\%\text{usage of } C_i}{\%\text{generated learnt } C_i}$.

The optimization problem. The idea here is to derive a selector that does not overpass a given threshold on the number of generated learnt clauses, while proposing a better usage frequency. Thus, we define a linear program $\mathbf{M}(b,\epsilon)$, that looks for a selector that maximizes the total usage frequency w.r.t. 3 major constraints:

- **(1)** A constraint to ensures that the total number of learnt clauses is *at most* equal to that corresponding to LBD≤b, with a precision $\epsilon \in [0,2]$. We observed that beyond 2% margin, the number of additional clauses becomes harmful for the solver.
- **(2)** A constraint which ensures that the total utilisation rate is equal to that corresponding to LBD≤b, *at worst*.
- **(3)** Constraints that reflect the importance of each class of clauses. Indeed, Fig.**3b** shows that some classes are more relevant than others (for example, class $C_{\mathcal{P}}$). This must have an influence on the resolution of the linear system. Therefore, we constraint the linear program to protect much more the relevant classes of clauses by giving them an LBD≥b. To do so, we associate a probability to each class $i \in X$ according to their relevance in Fig.**3b**. The probability is calculated by taking the total frequency of use of the clauses in a given class, and then normalizing over the 7 classes. Thus, for a set of classes $S \subseteq X$, having a probability higher than a given threshold[5], we force the system to generate a solution where the LBD score of the class $i \in S$ is at least equal to b (LBD≥b).

The formal description of the linear system needs the introduction of the following variables and notations:

- x_i^j: the decision variable of class i with LBD≤j.
- L_i^j: the percentage of generated learnt clauses (x-axis) of class i with an LBD≤j.
- U_i^j: the usage rate of class i (y-axis) with an LBD≤j.
- b: LBD value of reference used in the constraints.
- ϵ: the precision gap.

Hence, our optimization problem, $\mathbf{M}(b,\epsilon)$, is:

$$\mathbf{M}(b,\epsilon) = \quad \textbf{maximize} \sum_{i\in X} \mathbf{f}_i = \quad \sum_{i\in X}\sum_{j\geq 1} \frac{\overbrace{x_i^j U_i^j}^{\%\text{ usage}}}{\underbrace{x_i^j L_i^j}_{\%\text{ learnt}}}$$

$$\text{s.t.} \begin{cases} \sum_{i\in X}\sum_{j\geq 1} x_i^j L_i^j \;\leq (\sum_{i\in X}\sum_{j=1}^{b} L_i^j) + \epsilon & \textbf{(1)} \\ \sum_{i\in X}\sum_{j\geq 1} x_i^j U_i^j \;\geq \sum_{i\in X}\sum_{j=1}^{b} U_i^j & \textbf{(2)} \\ \sum_{j=1}^{b-1} x_i^j = 0 \qquad \forall i \in S & \textbf{(3)} \\ \sum_{j\geq 1} x_i^j = 1 \qquad \forall i \in X \\ x_i^j \in \{0,1\} \quad \forall i \in X, \quad \forall j, 0 < j \leq 22 \end{cases}$$

The system is applied on both *conflict-analysis* and *unit-propagation* procedures. We then obtain two selectors (the optimal solution for each of the two procedures). Naturally, we will opt for the solution with the best frequency usage in both.

Instantiation of the optimization problem. State-of-the-art solvers like MapleCOMSPS [32] identifies two particulars LBD values of interest: LBD≤3 and LBD≤6. Clauses having the former LBD value belong to the ***core*** database and are never deleted. Those clauses having the latter LBD value belong to the ***tier-2*** database and are treated with a finer deletion strategy. The Kissat-MAB [42] solver fellow the same reasoning but changing the LBD≤3 to LBD≤2 for the ***core*** database. Hence, we propose to derive our selectors according to these LBD values: S_3=$\mathbf{M}(3,\epsilon)$ (for LBD≤3) and S_6=$\mathbf{M}(6,\epsilon)$ (for LBD≤6). We recall here that we have defined these selectors for each category of the hierarchy of Manna&Pnueli. Each linear program was solved in less than 5 seconds using the Gurobi Optimizer [24].

[5]According to the information we collected in SectionV, we determined that classes with probability higher than 0.2 are interesting.

Selectors	Manna&Pnueli	$C_{\mathcal{J}}$	$C_{\mathcal{M}}$	$C_{\mathcal{MJ}}$	$C_{\mathcal{P}}$	$C_{\mathcal{PJ}}$	$C_{\mathcal{PMJ}}$	$C_{\mathcal{PM}}$
Default *core*	-	3	3	3	3	3	3	3
$\mathbf{S_3}$	S safety	3	3	3	3	3	2	4
	G guarantee	2	3	2	4	2	2	4
	O obligation	2	4	2	4	3	2	3
	P persistence	2	4	2	3	3	2	4
	R recurrence	2	3	2	4	3	2	4
	T reactivity	2	3	2	4	2	2	4
Default *tier-2*	-	6	6	6	6	6	6	6
$\mathbf{S_6}$	S safety	2	6	4	6	7	5	7
	G guarantee	6	6	4	5	8	5	10
	O obligation	3	19	3	11	7	5	6
	P persistence	22	4	5	9	7	5	7
	R recurrence	6	9	5	5	7	5	8
	T reactivity	22	6	3	6	6	5	7

TABLE I: Selectors computed using $H_{\mathcal{F}}$ on the training benchmark of each LTL property

Table-**I** presents the selectors S_3 and S_6 for each LTL property. A selector is described as a set of values, each of these corresponds to a class of clauses C_X. For example, value 19 of the column $C_{\mathcal{M}}$ means that S_6 identifies $C_{\mathcal{M}}$ clauses with an LBD$\leq$19, as good clauses for Obligation specifications.

At first sight, our strategies produce selectors that sharply contrast the state-of-the-art selectors. Indeed, the generated LBD values can go up to 22 while the default selectors are constants.

For instance, the S_6 selector on Obligation properties gives a high LBD score for $C_{\mathcal{P}}$ class (LBD$\leq$11) which correlates with the previous Fig. **3b**: $C_{\mathcal{P}}$ clauses are frequently used with fewer involved learnt clauses.

A comparison of the S_3 and S_6 selectors to the references show a better usage frequency. For instance, when observing the obligation properties, the usage frequency of S_3 is 9.97% (w.r.t. 7.20% of the generated learnt clauses) and 9.11% (w.r.t. 7.86% of the generated learnt clauses) for the reference. In the other hand, the usage frequency of S_6 is 1.25% (w.r.t. 16.12% of the generated learnt clauses) against 1.17% (w.r.t. 17.37% of the generated learnt clauses) for the reference. These number are taken from the *conflict-analysis* procedure.

The next section presents two approaches to protect those clauses identified by the selectors S_3 and S_6: for S_3, we use a permanent protection by redefining the ***core*** database (Section VII-B). For S_6, we soften the protection and act around the purge of ***tier-2*** and ***local*** databases (Section VII-C).

B. *Permanent protection* (**C**)

As the total size of clauses identified by S_3 is almost the same than the size for the default strategy (LBD$\leq$3), we propose here to apply a permanent protection strategy for the clauses of S_3. Indeed, we redefine the ***core*** database according to S_3: no deletion strategy is applied on this database, and all clauses identified by S_3 are kept forever. The remaining clauses not recognized by the selector follow the default storage technique: clauses with LBD$\leq$6 are kept in the ***tier-2*** and clauses with LBD$>$6 are added in the ***local*** database.

C. *Database Reduction* (**T**)

In the case of S_6, the number of identified clauses is too large to consider a permanent storage approach. The solver will be quickly flooded by too many clauses to manage. Therefore, we propose to act on the deletion strategies.

By default, in MapleCOMSPS, clauses with LBD$>$3 are stored in either the ***tier-2*** or ***local*** databases. Clauses in ***tier-2*** can be moved to ***local*** and conversely. Clauses in ***local*** may be permanently deleted if not used. Clauses in ***tier-2*** are kept if they are used at least once every 30000 conflicts, otherwise they are dropped to ***local***. Half of the clauses in this last database are permanently deleted.

We propose to modify the above strategy as follows: clauses identified by S_6 and belonging to the databases ***tier-2*** or ***local*** are kept "alive" for some additional time. They are not dropped if they are used at least once every $30000 \times r$ conflicts (we experimented on a subset of the benchmark different values of $r \in \{2, 4, 6, 8\}$ and the value 4 gave the best results). In our experiments, we set r to 4. *This way, the solver is given a chance to remove clauses that are totally useless.*

VIII. Benchmark

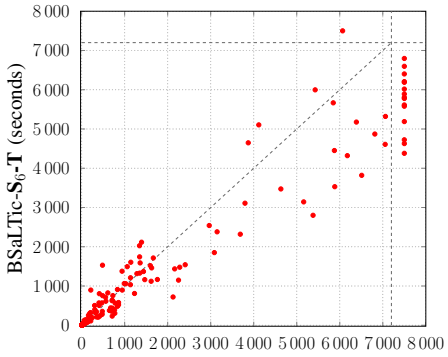
This section presents the evaluation of the heuristic $H_{\mathcal{F}}$ on the full benchmark presented in Section VI (omitting the training benchmark and the pre-processing time). All the experiments are conducted on an Intel Xeon machine with a time limit of 2 hours using BSaLTic tool.

A. *Comparison with state-of-the-art*

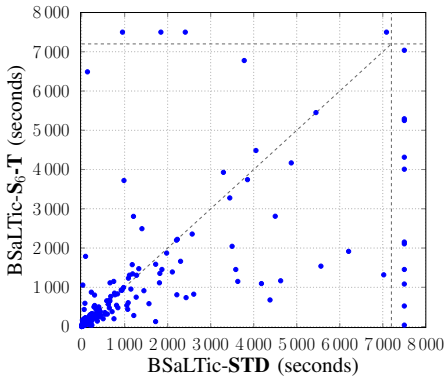
We evaluate our S_3 and S_6 selectors and their associated protections (resp. **C** and **T**), with state-of-the-art techniques: BSaLTic-**STD** (the MapleCOMSPS solver integrated into our tool BSaLTic), and the best approach presented in the paper [29], namely, BSaLTic-$\mathbf{H_{LP}}$. It relies on protecting good learnt clauses as the permanent protection strategy (**C**) but uses a different linear program to compute the selector. Moreover, BSaLTic-$\mathbf{H_{LP}}$ does not exploit the information provided by the studied LTL specification.

Solver	Manna&Pnueli	UNSAT	SAT	TOTAL	PAR-2
BSaLTic-**STD**	S safety	52	30	82	360h04
	G guarantee	72	56	128	170h35
	O obligation	25	107	132	135h15
	P persistence	46	66	112	213h34
	R recurrence	**43**	70	113	217h33
	T reactivity	56	79	135	124h08
	Total	294	408	702	1221h11
BSaLTic-$\mathbf{H_{LP}}$	S safety	52	28	80	364h40
	G guarantee	70	56	126	176h12
	O obligation	29	**109**	138	119h37
	P persistence	49	66	115	205h59
	R recurrence	42	**73**	115	210h50
	T reactivity	55	78	133	128h42
	Total	297	410	707	1206h02
BSaLTic-$\mathbf{S_3}$-**C**	S safety	**55**	**31**	**86**	**350h48**
	G guarantee	71	55	126	178h58
	O obligation	28	108	136	125h01
	P persistence	50	**68**	**118**	**194h43**
	R recurrence	**43**	71	114	213h58
	T reactivity	56	80	136	118h21
	Total	303	413	716	1181h52
BSaLTic-$\mathbf{S_6}$-**T**	S safety	52	29	81	364h03
	G guarantee	**74**	**57**	**131**	**163h51**
	O obligation	**32**	**109**	**141**	**110h06**
	P persistence	**51**	66	117	196h46
	R recurrence	**43**	**73**	**116**	**202h27**
	T reactivity	**58**	**81**	**139**	**109h15**
	Total	**310**	**415**	**725**	**1146h30**

TABLE II: Comparison between state-of-the-art solvers BSaLTic-**STD** and BSaLTic-$\mathbf{H_{LP}}$ with our modified solvers BSaLTic-$\mathbf{S_6}$-**T** and BSaLTic-$\mathbf{S_3}$-**C**.



(a) UNSAT instances



(b) SAT instances

Fig. 4: Scatter-plot of UNSAT and SAT instances comparing BSaLTic-$\mathbf{S_6}$-**T** to BSaLTic-**STD** solver (on O, P, R and T only)

For the sake of clarity we rename our S_3 and S_6 selectors with their associated protecting strategies BSaLTic-$\mathbf{S_3}$-**C** and BSaLTic-$\mathbf{S_6}$-**T**, respectively.

Table-**II** details the results of our experiments. For each class of the hierarchy, we display the number of UNSAT solved instances where no counter-example (of length at most k) was found that displays a violation of the property, the number of SAT solved instances that violate the handled LTL property, the total number of solved instances and the PAR-2 metric used in SAT competitions[6].

First, we observe that BSaLTic-**STD** obtains the worst performance due to its generic approach for preserving and detecting relevant clauses. We also note that, on this benchmark, the results of the approach presented in [29] are reproducible, i.e. tuning the core database according to some metric improves the results.

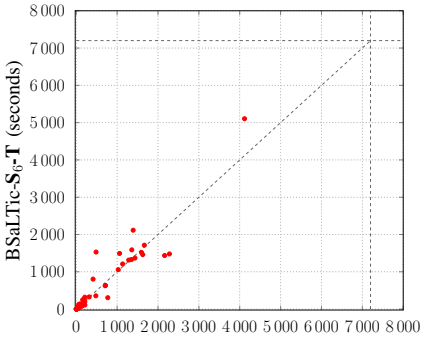
Table-**II** reveals that both the proposed approaches in this paper outperform state-of-the-art techniques. Indeed, BSaLTic-$\mathbf{S_6}$-**T** wins over the reference BSaLTic-**STD** on the SAT and UNSAT problems: it manages to solve 16 more UNSAT and 7 more SAT instances, with a PAR-2 of 74 hours less than BSaLTic-**STD**. It also outperforms BSaLTic-$\mathbf{H_{LP}}$ with a total of 18 instances resolved for 59 hours less. A closer look at the results shows that BSaLTic-$\mathbf{S_6}$-**T**, wins by a large margin over the high-level classes of Manna&Pnueli's hierarchy that are known to be difficult (O, P, R and T). Nonetheless, we remain competitive on safety and guarantee properties (S and G) where BSaLTic-$\mathbf{S_6}$-**T** solves as many UNSAT instances on safety, and 2 UNSAT instances more on guarantee properties than BSaLTic-**STD** approach. The explanation we have behind this, is that the generic tuning of learnt clause databases in SAT procedures already encompasses relevant information when it comes to solve safety or guarantee properties.

The scatter plots of Fig.**4** compares UNSAT and SAT instances (in seconds) between BSaLTic-**STD** and BSaLTic-$\mathbf{S_6}$-**T**, on LTL properties coming from higher parts of the hierarchy (i.e., O, P, R and T). It shows significant improvement on the known difficult UNSAT problems while being competitive on the SAT instances compared to the state-of-the-art approach.

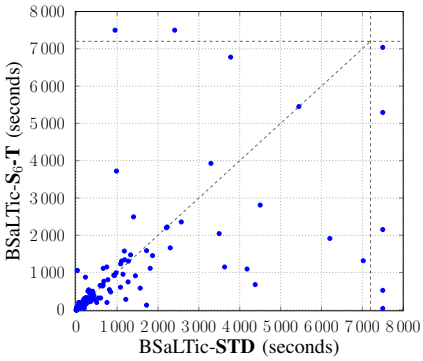
We investigate further the behaviour of BSaLTic-$\mathbf{S_6}$-**T** by splitting the benchmark into ASYNCHRONOUS and SYNCHRONOUS problems. Such problems have different characteristics and will help to decide when our approaches are useful.

The scatter plots in Fig.**5** and Fig.**6** reveal that BSaLTic-$\mathbf{S_6}$-**T** is mostly effective on ASYNCHRONOUS instances, in particular UNSAT ones (see Fig.**6a**). Nevertheless, BSaLTic-$\mathbf{S_6}$-**T** remains competitive on SYNCHRONOUS instances (Fig.**5a** and Fig.**5b**).

[6]PAR-k is the penalised average run time, counting each timeout as k times the running time cutoff.

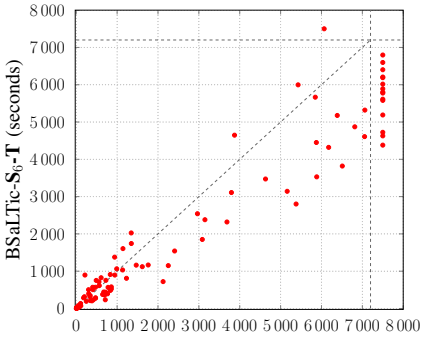


(a) UNSAT instances

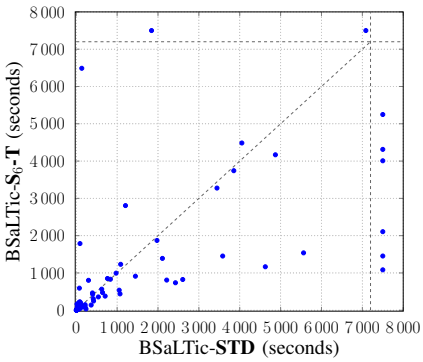


(b) SAT instances

Fig. 5: Scatter-plot of UNSAT and SAT SYNCHRONOUS problems comparing BSaLTic-$\mathbf{S_6}$-**T** to BSaLTic-**STD** solver (on O, P, R and T only)



(a) UNSAT instances



(b) SAT instances

Fig. 6: Scatter-plot of UNSAT and SAT ASYNCHRONOUS problems comparing BSaLTic-$\mathbf{S_6}$-**T** to BSaLTic-**STD** solver (on O, P, R and T only)

Solver	UNSAT	SAT	TOTAL	PAR-2
BSaLTic-$\mathbf{S_6}$-**T**	**310**	**415**	**725**	**1146h30**
BSaLTic-$\mathbf{S_3}$-**C**	303	413	716	1181h52
BSaLTic-$\mathbf{S_3}$-**C**,$\mathbf{S_6}$-**T**	302	411	713	1177h17

TABLE III: Comparison between BSaLTic-$\mathbf{S_6}$-**T**, BSaLTic-$\mathbf{S_3}$-**C** and their combo BSaLTic-$\mathbf{S_3}$-**C**,$\mathbf{S_6}$-**T**

Solver	Manna&Pnueli	UNSAT	SAT	TOTAL	PAR-2
BSaLTic-**nLTL-T**	S safety	51	**32**	**83**	**359h41**
	G guarantee	73	55	128	173h20
	O obligation	26	108	134	130h41
	P persistence	47	**66**	113	210h24
	R recurrence	**43**	71	114	206h20
	T reactivity	57	80	137	119h35
	Total	297	412	709	1200h05
BSaLTic-$\mathbf{H_{\mathcal{F}}}$	S safety	**52**	29	81	364h03
	G guarantee	**74**	**57**	**131**	**163h51**
	O obligation	**32**	**109**	**141**	**110h06**
	P persistence	**51**	**66**	**117**	**196h46**
	R recurrence	**43**	**73**	**116**	**202h27**
	T reactivity	**58**	**81**	**139**	**109h15**
	Total	**310**	**415**	**725**	**1146h30**

TABLE IV: Comparison of (non) LTL-based approaches BSaLTic-$\mathbf{S_6}$-**T** and BSaLTic-**nLTL-T**.

Although it is difficult to analyse the comparison of our approaches when using different back-end engines, we thought it was important to discuss the results we obtained using state-of-the-art solvers. Thus, in BSaLTic we replaced MapleCOMSPS with Kissat-MAB[7] [42]. The derived tool is called BSaLTic-Kissat-MAB where the solver is not tuned, and is used as a black-box. The results of the latter tool show that, although BSaLTic-Kissat-MAB outperforms the presented approaches globally, we manage to solve 26 SAT and 17 UNSAT that BSaLTic-Kissat-MAB could not solve (where it is known that the Kissat-MAB solver performs better on SAT instances).

Therefore, the question that remains open is how to integrate our optimizations in Kissat-MAB which has the same management of the different learnt clause, but a more complex implementation.

B. Variations of the proposed approaches

To have a complete study of the techniques proposed in this paper, we derive and evaluate new variants.

As BSaLTic-$\mathbf{S_3}$-**C** has the best performance on the safety properties, with 5 extra instances compared to the BSaLTic-$\mathbf{S_6}$-**T** approach (see Table-**II**), we experimented the combination of S_3 and S_6. The resulting tool, named BSaLTic-$\mathbf{S_3}$-**C**,$\mathbf{S_6}$-**T** will protect clauses of S_3 in the **core** database via the permanent protection **C**. The remain clauses identified by S_6 are stored using the database reduction **T**.

[7]the winner of the SAT 2021 competition

The aim is to increase the performance of BSaLTic-$\mathbf{S_6}$-**T** on safety problems, while maintaining its contribution on the rest of the problems.

Table-**III** highlights the results. The combined version manages to reduce the PAR-2 with 4 hours less in comparison to BSaLTic-$\mathbf{S_3}$-**C**, but it does not provide any improvement on the number of solved instances compared to the separated configurations. It solves only 713 instances while the least efficient solver (BSaLTic-$\mathbf{S_3}$-**C**) solves 716.

We also experimented a ***non*** LTL-based selector using $H_{\mathcal{F}}$ heuristic: BSaLTic-**nLTL-T**. It does not take into account the information provided by the LTL specification (i.e., the hierarchy of Manna&Pnueli).

Table-**IV** shows that dedicated LTL-based selector (BSaLTic-$\mathbf{S_6}$-**T**) outperforms the ***non*** LTL-based one (BSaLTic-**nLTL-T**), with 16 additional instances and a reduction on the PAR-2 time of 53 hours. As previously observed, (BSaLTic-$\mathbf{S_6}$-**T**) doesn't perform better on safety instances, however, the drawback of the LTL-based tuning only affects SAT instances but we remain competitive on well-known difficult UNSAT problems. *The experiment shown in Table **IV** strengthens the defended idea of building dedicated heuristics.*

IX. Conclusion

In this paper we proposed to exploit the characteristics of BMC problems in order to improve their solving through SAT techniques. Our approach focuses on tuning one of the main components of CDCL-like SAT solvers: learnt clauses databases. Such databases usually use generic metrics to identify and preserve relevant clauses. We proposed a refinement of this metric with structural information extracted from the model and the property at hand. We then presented the $H_{\mathcal{F}}$ heuristic that uses the above information to determine relevant learnt clauses. This latter, experimented over a large set of BMC problems, outperforms the state-of-the-art approaches on both *verified* (UNSAT) properties and *violated* ones (SAT) for a specific bound k. Thereby, building dedicated (problem-specific) selectors for managing the database of clauses is crucial but requires a deep understanding of the structure of the underlying problem.

The next steps of our work is to integrate this idea on the state-of-the-art Kissat-MAB [42] solver by performing the analysis (Section VI) and experimenting the selectors S_3 and S_6 on the Kissat-MAB version.

Another perspective is to experiment our idea in the context of parallel environments. The aim is to derive an efficient procedure for a SAT-based parallel solving of the BMC problem. Actually, one of the main components in SAT-based parallel solving is the sharing of information between different solvers. Hence, we think to adapt our approaches in order to fine-tune the sharing strategies.

References

[1] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distrib. Comput.*, 2(3):117–126, September 1987. doi:10.1007/BF01782772.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[3] Guillaume Baud-Berthier, Jesús Giráldez-Cru, and Laurent Simon. On the community structure of bounded model checking SAT problems. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT'17)*, pages 65–82, 2017.

[4] Armin Biere, Cyrille Artho, and Viktor Schuppan. Liveness checking as safety checking. *Electronic Notes in Theoretical Computer Science*, 66:160–177, 12 2002. doi:10.1016/S1571-0661(04)80410-9.

[5] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In W. Rance Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[6] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking, 12 2003. doi:10.1016/S0065-2458(03)58003-2.

[7] Aaron R. Bradley. Understanding IC3. In *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2012.

[8] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986. doi:10.1109/TC.1986.1676819.

[9] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[10] E. Clarke, K. McMillan, S. Campos, and V. Hartonas-Garmhausen. Symbolic model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, pages 419–422, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[11] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, July 2001. doi:10.1023/A:1011276507260.

[12] Edmund Clarke, E. Emerson, and Joseph Sifakis. Model checking. *Communications of the ACM*, 52, 11 2009. doi:10.1145/1592761.1592781.

[13] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Dexter Kozen, editor, *Logics of Programs*, pages 52–71, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg.

[14] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. *Model Checking and the State Explosion Problem*, pages 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[15] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962. doi:10.1145/368273.368557.

[16] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[17] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016.

[18] Stefan Edelkamp, Stefan Leue, and Alberto Lluch-Lafuente. Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology Transfer*, 5(2–3):247–267, 2004.

[19] Niklas Een, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, FMCAD '11, page 125–134, Austin, Texas, 2011. FMCAD Inc.

[20] Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL (Series on Integrated Circuits and Systems)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[21] Sami Evangelista, Alfons Laarman, Laure Petrucci, and Jaco van de Pol. Improved multi-core nested depth-first search. In *Proceedings of the 10th international conference on Automated technology for verification and analysis (ATVA'12)*, volume 7561 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, 2012.

[22] Malay K. Ganai. Sat-based scalable formal verification solutions. In *Series on Integrated Circuits and Systems, Springer-Verlag New York*, 2007.

[23] Matthew L. Ginsberg and David A. McAllester. Gsat and dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1994.

[24] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021. URL: http://www.gurobi.com.

[25] Gerard Holzmann. *Spin Model Checker, the: Primer and Reference Manual*. Addison-Wesley Professional, first edition, 2003.

[26] Gerard J. Holzmann. Explicit-state model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 153–171, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-10575-8_5.

[27] Paul Jackson and Daniel Sheridan. Clause form conversions for boolean circuits. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing*, pages 183–198, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[28] Sima Jamali and David Mitchell. Simplifying CDCL clause database reduction. In *SAT*, volume 11628 of *Lecture Notes in Computer Science*, pages 183–192. Springer, 2019.

[29] Anissa Kheireddine, Etienne Renault, and Souheib Baarir. Towards Better Heuristics for Solving Bounded Model Checking Problems. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:11, Dagstuhl, Germany, 2021. URL: https://drops.dagstuhl.de/opus/volltexte/2021/15298.

[30] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng.*, 3(2):125–143, mar 1977. doi:10.1109/TSE.1977.229904.

[31] Ludovic Le Frioux, Souheib Baarir, Julien Sopena, and Fabrice Kordon. PaInleSS: a framework for parallel SAT solving. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT'17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 233–250. Springer, Cham, August 2017.

[32] Jia Hui Liang, Chanseok Oh, Vijay Ganesh, Krzysztof Czarnecki, and Pascal Poupart. Maple-comsps, maplecomsps lrb, maplecomsps chb. *Proceedings of SAT Competition*, 2016, 2016.

[33] Z. Manna and A. Pnueli. A hierarchy of temporal properties (invited paper, 1989). In *PODC '90*, 1990.

[34] Joao Marques-Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48:506–521, 1999.

[35] Kenneth L. McMillan. *The SMV System*, pages 61–85. Springer US, Boston, MA, 1993.

[36] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, page 530–535, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/378239.379017.

[37] Samir Palnitkar. *Verilog HDL: A Guide to Digital Design and Synthesis*. Prentice-Hall, Inc., USA, 1996.

[38] Radek Pelánek. Beem: Benchmarks for explicit model checkers. In Dragan Bošnački and Stefan Edelkamp, editors, *Model Checking Software*, pages 263–267, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[39] Doron Peled and Klaus Havelund. *Refining the Safety–Liveness Classification of Temporal Properties According to Monitorability*, pages 218–234. Springer International Publishing, Cham, 2019.

[40] Etienne Renault, Alexandre Duret-Lutz, Fabrice Kordon, and Denis Poitrenaud. Strength-based decomposition of the property büchi automaton for faster model checking. In Nir Piterman and Scott A. Smolka, editors, *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, volume 7795 of *Lecture Notes in Computer Science*, pages 580–593. Springer, 2013.

[41] Kristin Y. Rozier. Survey: Linear temporal logic symbolic model checking. *Comput. Sci. Rev.*, 5(2):163–203, May 2011.

[42] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. Un bandit manchot pour combiner CHB et VSIDS. In *Actes des 16èmes Journées Francophones de Programmation par Contraintes (JFPC)*, Nice, France, June 2021. URL: https://hal-amu.archives-ouvertes.fr/hal-03270931.

[43] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a sat-solver. In Warren A. Hunt and Steven D. Johnson, editors, *Formal Methods in Computer-Aided Design*, pages 127–144, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[44] Ofer Shtrichman. Tuning sat checkers for bounded model checking. In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 480–494, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[45] João P. Marques Silva and Karem A. Sakallah. Grasp—a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '96, page 220–227, USA, 1997. IEEE Computer Society.

[46] Laurent Simon and Gilles Audemard. Predicting Learnt Clauses Quality in Modern SAT Solver. In *Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*, Pasadena, United States, July 2009. URL: https://hal.inria.fr/inria-00433805.

[47] Ofer Strichman. Accelerating bounded model checking of safety properties. *Formal Methods Syst. Des.*, 24(1):5–24, 2004.

[48] G. S. TSEITIN. On the complexity of derivation in propositional calculus. *Structures in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1968. URL: https://ci.nii.ac.jp/naid/10030021172/en/.

[49] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994. doi:https://doi.org/10.1006/inco.1994.1092.

[50] Chao Wang, HoonSang Jin, Gary D. Hachtel, and Fabio Somenzi. Refining the sat decision ordering for bounded model checking. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, page 535–538, New York, NY, USA, 2004. Association for Computing Machinery.

[51] Emmanuel Zarpas. Simple yet efficient improvements of sat based bounded model checking. In Alan J. Hu and Andrew K. Martin, editors, *Formal Methods in Computer-Aided Design*, pages 174–185, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[52] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '01, page 279–285. IEEE Press, 2001.