Boolean Abstraction for Temporal Logic Satisfiability*

Alessandro Cimatti[1], Marco Roveri[1], Viktor Schuppan[1], and Stefano Tonetta[2]

[1] FBK-irst, IT-38050 Trento, Italy
{cimatti,roveri,schuppan}@itc.it
[2] University of Lugano, Faculty of Informatics, CH-6904 Lugano, Switzerland
tonettas@lu.unisi.ch

Abstract. Increasing interest towards property based design calls for effective satisfiability procedures for expressive temporal logics, e.g. the IEEE standard Property Specification Language (PSL).
In this paper, we propose a new approach to the satisfiability of PSL formulae; we follow recent approaches to decision procedures for Satisfiability Modulo Theory, typically applied to fragments of First Order Logic. The underlying intuition is to combine two interacting search mechanisms: on one side, we search for assignments that satisfy the Boolean abstraction of the problem; on the other, we invoke a solver for temporal satisfiability on the conjunction of temporal formulae corresponding to the assignment. Within this framework, we explore two directions. First, given the fixed polarity of each constraint in the theory solver, aggressive simplifications can be applied. Second, we analyze the idea of conflict reconstruction: whenever a satisfying assignment at the level of the Boolean abstraction results in a temporally unsatisfiable problem, we identify inconsistent subsets that can be used to rule out possibly many other assignments. We propose two methods to extract conflict sets on conjunctions of temporal formulae (one based on BDD-based Model Checking, and one based on SAT-based Simple Bounded Model Checking). We analyze the limits and the merits of the approach with a thorough experimental evaluation.

1 Introduction

The role of properties in the design flow is becoming increasingly important. Properties can be used to describe design intent, document designs, and enable for earlier validation steps (e.g. in requirements analysis, realizability, and even in synthesis). Satisfiability engines for temporal logic formulae can be important backbones of property-based design. They can be used to show that a set of requirements is consistent, or entails some required properties, or is compatible with some desirable behaviors [27].

Given the degree of sophistication of model checking technologies, it would be tempting to reduce temporal logic satisfiability to model checking algorithms. However, model checking and requirements analysis are inherently different, and substantial problems from the user's perspective are open. For example, providing diagnostic information in case of inconsistency of a specification can not be solved by searching for a counterexample trace: the user is working at the level of requirements, and thus the inconsistency should be identified at the same level, e.g. as a subset of inconsistent requirements. Furthermore, this approach may have some limitations: in fact, techniques

* This work has been partly supported by ORCHID, a project sponsored by the Provincia Autonoma di Trento, and by the European Commission under contract 507219 (PROSYD).

and tools for temporal logic model checking are focusing on complexity in the model, and even reductions on the temporal logic formula [30] are oriented to dominating the complexity in the model.

In this paper we propose a novel approach to the satisfiability of temporal logic. The intuition is to combine two forms of search: Boolean enumeration and temporal reasoning. Boolean enumeration is carried out on the propositional abstraction of the specification, where temporal atoms are abstracted into Boolean atoms; once a satisfying assignment is available, temporal reasoning is invoked on the corresponding set of temporal formulae. If a model is found, then the problem is satisfiable, otherwise reasoning theory is used to reconstruct a conflict, and the iteration is restarted.

This approach is mutuated by recent work on Satisfiability Modulo Theories (SMT) [8]. To the best of our knowledge, this is the first time the SMT paradigm, typically used for decidable fragments of First Order Logics, is applied to temporal satisfiability. This choice provides a clear conceptual framework, and suggests several important directions. First, don't cares in the Boolean abstraction of the problem pinpoint temporal formulae that are irrelevant for satisfiability, and can be safely ignored, thus reducing the effort to be carried out in temporal reasoning. Second, fixed polarity constraints are given in input to the theory solver: this enables more aggressive simplifications of the input problem (e.g. pure literal rule). Third, the theory solvers for temporal logic should be extended to provide unsatisfiable cores (or simply unsat cores): these are explanations for unsatisfiability, i.e. inconsistent subsets of the problem in input. This information can be used to rule out all those assignments that satisfy the Boolean abstraction of the problem, but are associated with a superset of an unsatisfiable core. We extend two satisfiability checking algorithms, one based on BDD-based language emptiness [12], and one on SAT-based Simple Bounded Model Checking (SBMC) [20], to return unsat cores. This is in general an interesting aspect, since it enables to provide explanations for unsatisfiability, and ultimately to generalize the idea of unsatisfiable core to the case of temporal logic.

We instantiate our approach on the Property Specification Language (PSL) [1], for its high expressiveness (it captures all ω-regular languages), and its practical interest. We remark however that the approach is general, and independent of the specific temporal logic at hand. The approach has been implemented within the NuSMV model checker [9]. A notable feature at the implementation level is that we use Binary Decision Diagrams as the top level enumeration mechanism. This is in contrast to the current trends in SMT, where DPLL-based enumeration is becoming a de facto standard. A DPLL-based enumeration could have been adopted here, and will be in fact investigated in the future. The BDD-based approach is justified by the fact that for the problems at hand the Boolean splitting is dominated by the temporal one, and as such, BDD-based reasoning turns out not to be a bottleneck. The approach has been experimentally evaluated on a large set of benchmarks, both for BDD-based and SAT-based techniques, and the results are very promising.

This paper is structured as follows. In Section 2, we shortly present the temporal logic PSL. In Section 3, we discuss previous approaches to temporal logic satisfiability. In Section 4, we overview the proposed approach. In Section 5, we discuss the idea of pure literal simplification, and the algorithms for the extraction of unsat cores. In

Section 6, we experimentally evaluate our approach. Finally, in Section 7, we draw some conclusions and outline directions for future work.

2 The Property Specification Language PSL

In this paper, we use PSL [1] as our temporal logic. PSL is a very rich language. Here we consider a subset, which is mostly used in practice, and provides ω-regular expressiveness [3]. The subset combines Linear Temporal Logic [28] (LTL) and Sequential Extended Regular Expressions (SERE) [1]. (SEREs extend classical regular expressions with language intersection, thus allowing for a greater succinctness at a cost of a possible exponential blow-up in the conversion to automata. Moreover, the atoms of SEREs are Boolean expressions enabling efficient determinization of automata.)

In the definition of the PSL syntax, for technical reasons, we introduce the "suffix conjunction" connective as a dual of the suffix implication. Moreover, we consider only the strong version of the temporal operators (the weak operators can be rewritten in terms of the strong ones [1]) and the strong version of the SEREs (though our approach can be easily extended to deal also with the weak semantics).

Definition 1 **(PSL syntax).** *Assume a set $\mathscr{A}$ of atomic propositions. We define the* PSL *formulae, as follows:*

- *if $p \in \mathscr{A}$, p is a* PSL *formula;*
- *if ϕ_1 and ϕ_2 are* PSL *formulae, then $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ are* PSL *formulae;*
- *if ϕ_1 and ϕ_2 are* PSL *formulae, then $\mathbf{X}\phi_1$, $\phi_1\mathbf{U}\phi_2$, $\phi_1\mathbf{R}\phi_2$ are* PSL *formulae;*
- *if r is a* SERE *and ϕ is a* PSL *formulae, then $r \diamond\!\!\rightarrow \phi$ and $r \mapsto \phi$ are* PSL *formulae;*
- *if r is a* SERE*, then r is a* PSL *formula.*

The $\mathbf{X}$ ("next-time"), the $\mathbf{U}$ ("until"), and the $\mathbf{R}$ ("releases") operators are called *temporal operators*. We call the $\diamond\!\!\rightarrow$ ("suffix conjunction"), and the $\mapsto$ ("suffix implication"), *suffix operators*. Notice that, the r not occurring in the left side of a suffix operator is the *strong* version of a SERE ($r!$ in the PSL notation). In the following, we will consider such r as an abbreviation for $r \diamond\!\!\rightarrow True$ [4]. We also use $\mathbf{G}\phi$ as an abbreviation for $\neg(True\ \mathbf{U}\ \neg\phi)$. LTL can be seen as a subset of PSL in which the suffix operators and the SEREs are suppressed.

We refer the reader to [1] for a formal definition of the semantics of PSL, and in particular of the entailment relation $w \models \phi$ for any infinite word w over a given alphabet Σ ($\Sigma = 2^{\mathscr{A}}$) and PSL formula ϕ. Notice that we can build Boolean expressions by means of atomic formulae and Boolean connectives. The language of a PSL formula ϕ over the alphabet Σ is defined as the set $\mathscr{L}(\phi) := \{w \in \Sigma^\omega \mid w \models \phi\}$. The satisfiability problem is to check if $\mathscr{L}(\phi) \neq \emptyset$ for a given PSL formula ϕ.

3 Previous Approaches to PSL Satisfiability

Satisfiability of temporal logics [15] has been widely studied. The seminal work of [32] established the PSPACE-completeness of the satisfiability problem for LTL. Since then, many techniques have been proposed to solve the problem. The first decision procedures

are based on tableau systems [34,24,23]. The tableau rules exploit the connection between the syntax of formulae and the tableau structures. The expansion is terminated by some criteria based either on the recurrence of nodes or on maximal strongly connected components. Temporal resolution has been devoted some attention [16] and has been used as a basis for works based on theorem proving, as well as inspiration for SNF-based LTL bounded model checking. Satisfiability of LTL can also be reduced to check language emptiness of Nondeterministic Büchi automata (NBA) [33] or to check the existence of a winning strategy for focus games [22].

In particular, if we reduce the satisfiability problem to checking the emptiness of the language of an NBA, we can exploit model checking engines: it is possible to check the satisfiability of formula ϕ by model checking the validity of the negation of ϕ on a completely nondeterministic Kripke Structure. This way we can exploit symbolic techniques both for the translation of the formula into Büchi automata and for the emptiness checking [12]. Alternatively, it is possible to use SAT based bounded model checking techniques as in [5].

The satisfiability problem has been extended also to other temporal logics. In particular, [2] studied the satisfiability of richer languages that combine LTL with regular expressions, such as ForSpec [2] and PSL [1]. The satisfiability of subsets of ITL [18] has also been studied in [26].

We now concentrate on recent approaches to dealing with satisfiability of PSL, namely [19,7,10,29,11]. The first step in the so-called monolithic approaches is to convert the PSL problem in a monolithic alternating Büchi automaton (ABA); during the conversion, semantic simplification steps (such as the elimination of unreachable states, and restricted forms of minimization by observational equivalence) are applied. The ABA is then converted into a symbolically represented NBA. In [7], this is done by means of a symbolic encoding of Miyano and Hayashi [25], and can be applied both to BDD-based and SAT-based approaches. In [19], an encoding of the ABA that is specialized for bounded model checking is proposed.

The conversion proposed in [10] is based on the so called Suffix Operator Normal Form (SONF). The idea is to partition the translation, by first converting a PSL formula ϕ into an equi-satisfiable formula in SONF, structured as $\bigwedge_i \phi_i \wedge \bigwedge_j \mathbf{G}(p_I^j \rightarrow (r_j \star\!\!\rightarrow p_F^j))$, where ϕ_i are LTL formulae, r_j are SEREs, p_I^j and p_F^j are propositional atoms, and $\star\!\!\rightarrow$ is either $\mapsto$ or $\Diamond\!\!\rightarrow$. Formulae of the form $\mathbf{G}(p_I^j \rightarrow (r_j \star\!\!\rightarrow p_F^j))$ are called Suffix Operator Subformulae (SOS's). The translation first converts the formula in NNF, and then "lifts out" the occurrences of suffix operators, by introducing fresh variables (intuitively, the p^j in the formula above), together with the corresponding SOS. For lack of space, we omit the details regarding the conversion of SOS into NBA; we only mention that the translation is specialized to exploit the structure of SOS (see [10] for details).

The translation presented in [29] introduces a new variable for every subformula. A difference is that the testers of [29] set the new variable to true *if and only if* the subformula is satisfied, while in SONF the subformula is triggered by an implication, and is, as such, more amenable to the exploitation of don't cares.

In [10], a substantial experimental evaluation is carried out on PSL satisfiability. The SONF based approach results in dramatic improvements in PSL automata compilation

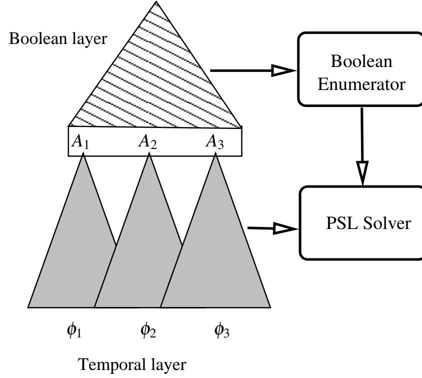

Fig. 1. Satisfiability modulo theory schema for PSL.

```
1:  function PSLSat(Φ)
2:      φ̄ ← Abstract(φ)
3:      Init(ImpIter, φ̄)
4:      while (HasNext(ImpIter)) do
5:          μ ← GetNext(ImpIter)
6:          Φ_μ ← Concretize(μ)
7:          (res, reason) ← IsPslSat(Φ_μ)
8:          if (res = "sat") then
9:              return "sat"
10:         else
11:             Prune(ImpIter, reason)
12:         end if
13:     end while
14:     return "unsat"
15: end function
```

Fig. 2. The PSL satisfiability algorithm.

time. However, on those problems where the ABA construction succeed to build an automaton within the time limit, the search time is typically in favor of the monolithic approach. This is mainly due to the fact that in certain examples the semantic simplifications are extremely effective. In [11], additional improvements over [10] are obtained by applying cheap syntactic simplification rules, that result in additional savings not only in search but also in construction time.

4 Boolean Abstraction for Temporal Satisfiability

Consider the temporal satisfiability problem $\Phi \doteq (\phi_1 \leftrightarrow (\phi_2 \vee \phi_3))$. If it is possible to show that the set $\{\phi_1, \phi_3\}$ is temporally satisfied by a word w, then ϕ_2 is irrelevant: intuitively, the truth value of ϕ_2 over w can not affect the truth of Φ. However, all the automata-based approaches presented in the previous section are going to compile the formula *statically*: this means, for instance, that they will generate and search an automaton taking into account each ϕ_i. Information could be potentially disregarded because of the Boolean structure of the formula is in fact taken into account.

In this paper we propose a new approach that tries to overcome this problem. The idea, depicted in Fig. 1, is to decouple the search for a temporal model in two interacting, hierarchically connected phases: in the first, we look for a propositionally satisfying assignment (an implicant) to the Boolean abstraction of the problem; in the second, we check whether the set of temporal formulae corresponding to the implicant is temporally satisfiable.

We see a temporal property Φ as a Boolean combination $BoolComb(\phi_1, \dots, \phi_n)$, where ϕ_i are distinct temporal formulae. The Boolean abstraction of Φ is $\overline{\phi} \doteq BoolComb(A_1, \dots, A_n)$, where the A_i are distinct Boolean variables (called the Boolean abstraction of ϕ_i); in the example above, $\overline{\phi}$ is $(A_1 \leftrightarrow (A_2 \vee A_3))$. We define an assignment μ for $\overline{\phi}$ as a mapping from each A_i to $\{True, False, X\}$. We call μ_t the atoms assigned to $True$, μ_f the atoms assigned to $False$, and μ_x the atoms assigned to X(for don't care). We say that μ propositionally satisfies $\overline{\phi}$ iff the formula obtained by replacing each occurrence of $A_i \in \mu_t$ with $True$, and each $A_i \in \mu_f$ with $False$, evaluates to $True$. A temporal model for Φ can be seen as an assignment μ satisfying $\overline{\phi}$, plus a temporal model for the conjunction of the required temporal formulae.

Theorem 1. *Φ is satisfiable iff there exists a truth assignment μ for $\overline{\phi}$ such that Φ_μ is satisfiable, where*

$$\Phi_\mu \doteq \bigwedge_{i.A_i \in \mu_t} \phi_i \wedge \bigwedge_{j.A_j \in \mu_f} \neg \phi_j$$

This theorem suggests an algorithm to check the satisfiability of a PSL formula Φ, and in general of any temporal formula. The (disjunctive) Boolean structure of Φ, expressible as a disjunction of Φ_μ, can be used to obtain several (hopefully) smaller automata, that can then be analyzed individually, with standard language emptiness checks (or other techniques).

Figure 2 reports the algorithm. The function PSLSAT( ) takes in input a PSL property Φ and returns "sat" iff Φ is satisfiable, otherwise it returns "unsat". ABSTRACT( ) builds the Boolean abstraction for Φ. *ImpIter* enables enumeration of the implicants (satisfying assignments) of $\overline{\phi}$. HASNEXT( ) returns *True* iff there is at least one (yet unexplored) implicant left. GETNEXT( ) returns the next such implicant. If there is none, then the PSL formula is unsatisfiable and "unsat" is returned (line 14). Otherwise, we iterate for each implicant (lines 4–13). From μ the function CONCRETIZE(μ) builds the formula Φ_μ corresponding to the implicant μ. ISPSLSAT( ) is a function that takes a PSL property and returns "sat" if the property is unsatisfiable, otherwise it possibly returns a reason for the unsatisfiability. This function can simply be any of the functions reported in Sect. 3. If the Φ_μ is temporally satisfiable, then we are done and the top level function returns "sat". Otherwise, at line 11, the result is analyzed by PRUNE( ), removing all remaining prime implicants that can be inferred to be unsatisfiable from the obtained *reason*. In our implementation *reason* is a set of implicants corresponding to a set of unsatisfiable cores of Φ_μ. Note that the unsatisfiability of Φ_μ establishes Φ_μ itself as an unsatisfiable core.

Relations to Satisfiability Modulo Theories. The high level schema presented above is largely inspired by the standard approaches to decision procedures for Satisfiability Modulo Theories (SMT), implemented in a number of systems and for a number of theories. In SMT, the enumeration of satisfying assignments is often carried out by a DPLL-based solver, that incrementally constructs an assignment for the Boolean abstraction of the formula. A typical technique is *early pruning*, where the theory solvers are called on the concretization of the assignment while this is being constructed. The advantage of early pruning is that it can prune a partial assignment as soon as its concretization becomes theory-unsatisfiable.

Some SMT solvers do attempt to extract don't care information on the Boolean abstraction; the combination with early pruning, however, appears to be nontrivial. Here we take a different perspective: we do not rely on early pruning, and try to exploit the presence of don't cares as much as possible, by enumerating prime implicants. This enables us to limit the number of theory constraints (PSL properties) sent to the theory solver. We base our Boolean enumeration on a BDD package, that provides primitives for on-the-fly extraction of one prime implicant. This choice is mostly motivated by the fact that the complexity of the temporal reasoning often dominates the problem, and thus BDD-based enumeration of prime implicants is not a bottleneck; in the future we also plan to experiment with DPLL-based enumeration. Another interesting feature

is that “essential literals”, i.e. literals that are common to all prime implicants, can be extracted at a reasonable cost. Notice that the set of essential literals includes all literals that can be obtained by standard SAT-based unit propagation and potentially more.

A key issue with SMT is the ability to avoid the same mistake in theory reasoning: more precisely, we don't want to try a prime implicant, if its intersection with a previously disproved one concretizes to an inconsistent set of temporal formulae. This problem is addressed by requiring that theory solvers should return a conflict set, i.e. an inconsistent subset of the problem it was given in input. In DPLL-based SMT, theory solvers are able to express conflicts in form of *conflict clauses*, that can be easily integrated with the conflict analysis and back-jumping mechanism. In the next section, we discuss how to address this problem in the setting of temporal satisfiability.

5 A Theory Solver for Temporal Logic

We now discuss how to design a theory solver. First, we exploit the fact that the input problem is a conjunction of temporal constraints with fixed polarity. This opens up to many optimizations. A particularly interesting simplification, given the fixed polarity of the constraints, is based on the notion of pure literal for PSL (Sect. 5.1). Then, we propose two new methods for the extraction of unsatisfiable cores (conflict sets) from the standard PSL satisfiability algorithms, one based on the use of BDD techniques (Sect. 5.2), and the second based on the use of SAT techniques (Sect. 5.3).

5.1 Pure Literal Simplification for PSL

First, we extend the notion of positive/negative occurrence of a proposition (the notion of positive/negative occurrence of a proposition in a Boolean expression is assumed to be known), and then we extend the notion to PSL formulae.

Definition 2. *If an occurrence of p in a Boolean expression b is positive [resp., negative] and b occurs in a SERE r, then that occurrence of p is positive [resp., negative] in r too.*

Let ϕ be a PSL formula and p a proposition. We define if an occurrence of p in ϕ is positive [resp., negative] recursively on the syntax of PSL formulae:

- *p is a positive occurrence of p in p*
- *every positive [resp., negative] occurrence of p in ϕ is a negative [resp., positive] occurrence in $\neg\phi$*
- *every positive [resp., negative] occurrence of p in ϕ is a positive [resp., negative] occurrence in $X\phi$, $\psi \wedge \phi$, $\phi \wedge \psi$, $\psi U \phi$, $\phi U \psi$, $\phi R \psi$, $\psi R \phi$, $r \mapsto \phi$, and $r \diamond\!\!\rightarrow \phi$*
- *every positive [resp., negative] occurrence of p in r is a positive [resp., negative] occurrence in $r \diamond\!\!\rightarrow \phi$*
- *every positive [resp., negative] occurrence of p in r is a negative [resp., positive] occurrence in $r \mapsto \phi$*

We now define when a proposition is pure. Intuitively, if the proposition is pure, we can substitute every occurrence with either true or false, depending on the polarity, without affecting the satisfiability.

Definition 3. *Let ϕ be a PSL formula and p a proposition. p is pure positive [pure negative, resp.] in ϕ iff all the occurrences of p are positive [negative, resp.].*

Theorem 2. *If p is pure positive [pure negative, resp.] in ϕ, then ϕ is satisfiable iff $\phi[\top/p]$ [resp., $\phi[\bot/p]$] is satisfiable.*

5.2 BDD-based Inconsistency Analysis

The first inconsistency analysis technique exploits a BDD-based computation of the fair states, i.e. those states that are the starting point of an accepting path. The standard symbolic procedure to check language emptiness (LE) [12] builds an automaton for the input formula Φ, computes the set of fair states and intersects it with the initial states: the resulting set (denoted with $[[\Phi]]$) contains all states that are the starting point of some path that accepts Φ.

Let $\phi_0,\ldots,\phi_n$ be temporal formulae with a top-level temporal operator over a set of atomic propositions AP. For each temporal formula ϕ_i, we introduce an activation variable. Let $A_0,\ldots,A_n$ be atomic propositions not in AP. We define a formula Ψ as

$$\Psi = \bigwedge_i A_i \rightarrow \phi_i(x)$$

The set $[[\Psi]]$ resulting from applying LE to Ψ is conditioned by the activation variables: it contains tuples of state variables from the automata of the ϕ_i together with the activation variables A_i. In order to obtain the sets of temporal formulae ϕ_i which are inconsistent, we look at those tuples of activation variables that do not have any corresponding state in $[[\Psi]]$.

Formally, suppose that M_Ψ is an automaton represented with a set V of state variables and that M_Ψ encodes the formula Ψ so that a set $[[\Psi]]$ of states is defined in such a way that:

1. all states in $[[\Psi]]$ are the starting point of some path accepting Ψ;
2. all words satisfying Ψ are accepted by some path starting from $[[\Psi]]$.

Suppose that V contains a variable v_{A_i} for every activation variable A_i such that a state s assigns v_{A_i} to true iff all paths starting from s accept the propositional formula A_i. Let $V_A = \{v_{A_0}\ldots v_{A_n}\}$ and $V' = V_\psi \setminus V_A$. [3]

Theorem 3. *Let UC be a subset of $\{0,\ldots,k\}$. Then, there exists a state s in $[[\Psi]]$ such that $s \models \bigwedge_{i\in UC} v_{A_i}$ iff $\bigwedge_{i\in UC}\phi_i$ is satisfiable.*

Corollary 1.

$$\exists V'([[\Psi]]) = \{s \in 2^{V_A} \,|\, \textstyle\bigwedge_{i.s\models v_{A_i}} \phi_i \textit{ is sat}\}$$
$$\neg\exists V'([[\Psi]]) = \{s \in 2^{V_A} \,|\, \textstyle\bigwedge_{i.s\models v_{A_i}} \phi_i \textit{ is unsat}\}$$

Thus, the set $\neg\exists V'([[\Psi]])$ encodes all the possible subset of the implicant $\phi_0,\ldots,\phi_n$ that are inconsistent. When using SAT-based enumeration of implicants, we should not add all the clauses corresponding to the above set as blocking clauses. Instead, techniques to minimize the configurations should be employed. In our BDD-based setting, the information can be directly fed back to the main search (by a simple conjunction within the prime implicants enumeration routine) in order to prevent the next iterations of Boolean enumeration from producing PSL-unsatisfiable configurations.

[3] Note that the LTL compilation of [12] and the PSL compilation discussed in Section 3 satisfy all these assumptions.

5.3 SAT-based Inconsistency Analysis

Standard incremental SAT-based bounded model checkers with completeness, such as [19], can be used off-the-shelf to determine language emptiness for LTL formulae. These approaches can be extended to extract an unsatisfiable core from a conjunction of temporal constraints.

Intuitively, the extraction relies on the ability of a Boolean SAT solver such as MiniSat [14] to check satisfiability of a Boolean formula f under a set of assumed literals $\{l_i\}$, i.e., $(\bigwedge_i l_i) \wedge f$. If that turns out to be unsatisfiable, the SAT solver returns a subset $UC \subseteq \{l_i\}$ such that $UC \wedge f$ is still unsatisfiable. Given a prime implicant Φ_μ we prefix the formulae ϕ_i with activation variables A_i as in the previous section. We then supply the literals corresponding to the value $True$ for the activation variables at the initial time step as assumptions to the SAT solver. When a subset of these literals is reported to cause a conflict, it is straightforward to obtain the corresponding unsatisfiable core of Φ_μ. This SAT approach, differently from the BDD-based approach, computes only a single rather than the set of all unsatisfiable cores for Φ_μ. In the following we formalize that intuition.

SAT-based bounded model checking [5] represents a finite path π of length k over a set of variables V as the valuations of a set of variables $V[0,k]$, where $V[0,k]$ contains one variable $v[i]$ for each $v \in V$ and $0 \leq i \leq k$.

Given a set of variables V, a linear temporal logic formula ϕ, and a natural number k, a SAT-based bounded model checker following the approach [6] in Fig. 3 generates the following Boolean formulae:[4]

1. a *witness formula* $|[V,\phi,k]|$ over (a superset of) $V[0,k]$. The set of satisfying assignments of $|[V,\phi,k]|$ corresponds exactly to the set of paths $\pi[0,k]$ such that π represents a lasso-shaped path that satisfies ϕ.
2. a *completeness formula* $\langle\langle V,\phi,k\rangle\rangle$ over (a superset of) $V[0,k]$. If $\langle\langle V,\phi,k\rangle\rangle$ is unsatisfiable, then ϕ is unsatisfiable.

Let Φ_μ be a prime implicant with atoms $\{\phi_0,\ldots,\phi_n\}$ and activation variables $\{A_0,\ldots,A_n\}$, and let ψ be Φ_μ prefixed with activation variables as in the previous section. Then we have

Lemma 1. $(\bigwedge_i v_{A_i}[0]) \wedge |[V \cup V_A, \psi, k]|$ *is satisfiable iff there is a lasso-shaped witness* $\pi[0,l-1] \circ \pi[l,k]^\omega$ *of* Φ_μ.

Lemma 2. *Let* $UC \subseteq \{0,\ldots,n\}$. *If* $\langle\langle V \cup V_A, \psi, k\rangle\rangle$ *is unsatisfiable under assumptions* $\{v_{A_i}[0] \mid i \in UC\}$ *then* $\bigwedge_{i \in UC} \phi_i$ *is unsatisfiable.*

Theorem 4. *The algorithm in Fig. 3 returns Sat iff* Φ_μ *is satisfiable. If it returns* $(Unsat, UC)$*, then UC is an unsatisfiable core of* Φ_μ.

[4] Model checking typically involves both, a model and a temporal logic formula. As we are only concerned with satisfiability of linear temporal logic formulae, we disregard the model part. To simplify the presentation we also disregard that (1) the witness formula typically allows to detect finite violating prefixes of safety properties [21] and (2) guaranteeing termination requires additional constraints [31,6]. Our implementation handles both.

1: **function** LE_SAT(Φ_μ)
2: $k \leftarrow 0$
3: **while** $(True)$ **do**
4: $(res, UC) \leftarrow$ SAT_ASSUME($\langle\langle V \cup V_A, \psi, k \rangle\rangle$, $\{v_A[0] \mid v_A \in V_A\}$)
5: **if** $(res = Unsat)$ **then** **return** $(Unsat, \{\phi_i \mid v_{A_i}[0] \in UC\})$
6: $res \leftarrow$ SAT($(\bigwedge_{v_A \in V_A} v_A[0]) \wedge |[V \cup V_A, \psi, k]|$)
7: **if** $(res = Sat)$ **then** **return** Sat
8: $k \leftarrow k+1$
9: **end while**
10: **end function**

Fig. 3. SAT-based language emptiness with unsatisfiable cores.

6 Experiments

The algorithms described in previous sections have been implemented within the NUSMV model checker [9]. To show the effectiveness of the proposed approach, we carried out an experimental evaluation, based on the benchmarks proposed in [10,11], in [17], and also on some benchmarks collected from the web. The benchmarks from [10,11] are random properties obtained by applying to randomly generated SEREs typical patterns extracted from industrial case studies [13]. The benchmarks are either randomly generated Boolean combinations of such typical properties, or implications/bi-implications between large conjunctions of such typical properties. The latter cases model refinement and equivalence among specifications, as is often seen in requirements engineering. The benchmarks from [17] are properties coming from a requirements engineering domain, and model whether a given property is implied by a big conjunction of other properties.

We evaluate the Boolean abstraction approach using BDD-based theory solving, and SAT-based SBMC theory solving, both based on the SONF algorithm for PSL satisfiability presented in [11]. The same approach was also chosen as a base line for the evaluation of performance improvements. (We also considered the possibility to include in our comparison other tools, e.g. [22,26], at least for pure LTL problems. However, some preliminary experiments on moderate-sized problems clearly indicated that the satisfiability based on model checking is vastly superior, at least in terms of the currently available implementations.) In the experiments, we evaluate the impact of Boolean abstraction, pure literal simplification, and feedback.

All experiments were run on a 3 GHz Intel CPU equipped with 4 GB of memory running Linux; for each run, we used a timeout of 120 seconds and a memory limit of 768 MB. For all methods, we used the settings that turned out to provide better results in [10,11]. For BDD-based methods, dynamic variable reordering was used, and forward reachability simplification was enabled on the tableau automata. For SAT-based methods, we used MiniSAT [14], and we enabled the completeness check based on the simple path constraint. The complete test suite and an extended version of this paper can be found at `http://sra.itc.it/people/roveri/cav07-bapsl/`.

In the following we use method descriptors consisting of up to five parts to describe an approach. (1) If ba is present, Boolean abstraction is used. (2) sbmc or bdd indicates whether SAT- or BDD-based solvers are used. (3) Presence of fb indicates that feedback is used. Finally, (4) pt and (5) ppi stand for pure literal simplification applied

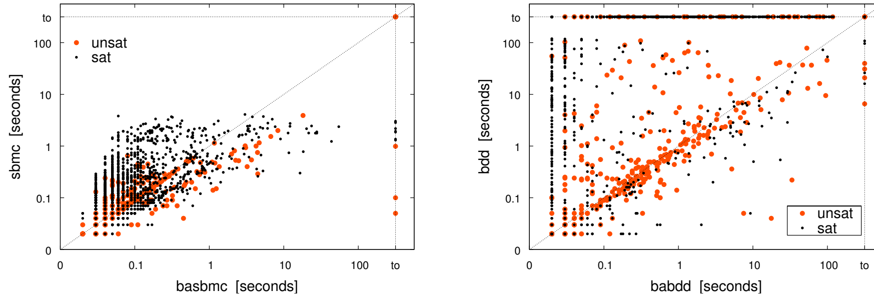

Fig. 4. Solving time of approaches with and without Boolean abstraction.

at the top and prime-implicant levels, respectively. As an example, babddptppi stands for BDD-based solver with Boolean abstraction and with the pure literal simplification applied both to the the top-level formula and to each prime implicant, but without using feedback. bdd and sbmc mark the respective base line approaches.

In Fig. 4, we report the scatter plots comparing the Boolean abstraction approaches (no pure literal simplification and no feedback) against the corresponding base line without Boolean abstraction. The plots show that the Boolean enumeration approach may lead to advantages in the case of SAT, and is vastly superior in the case of BDDs.

In Fig. 5, we compare Boolean abstraction with and without pure literal simplification. The plots show that the pure literal simplification dramatically reduces search time, both when applied at the prime implicant (row 1) and at the top level (row 2). Row 3 demonstrates that the application on the prime implicant level can gain an additional advantage even after the application on the top level. For our set of examples, the reverse is not true, see row 4. Rather, there seems to be a small penalty induced by the overhead of pure literal simplification at the top. In all cases the impact with BDD-based solvers turns out to be much stronger than with SAT-based solvers.

We now analyze the impact of feedback. In Fig. 6, upper row, we compare basbmc and babdd with the corresponding configurations with feedback activated. The plots show that enabling conflict extraction sometimes pays off, but most often it degrades the overall performance. However, the degraded performance can be explained with the fact that the current implementation of the feedback is rather naïve, and uses the theory solvers as off-the-shelf. Interestingly enough, the generation of conflicts sets can dramatically reduce the search space, by avoiding to reconsider implicants that proved to be inconsistent in previous calls. This is clear if we plot the number of prime implicants analyzed by the algorithms with and without feedback before determining a result (see plots of the second row).

For more results see the extended version of this paper, including pure literal simplification for sbmc and bdd, a comparison between SAT- and BDD-based approaches, and splitting some results of Figs. 4, 6 by the number of prime implicants examined.

7 Conclusion and Future Work

We proposed a novel paradigm for satisfiability of temporal logics, where model enumeration applied to the propositional abstraction of the problem interacts with a solver able to decide conjunctions of temporal formulae. A thorough experimental evaluation

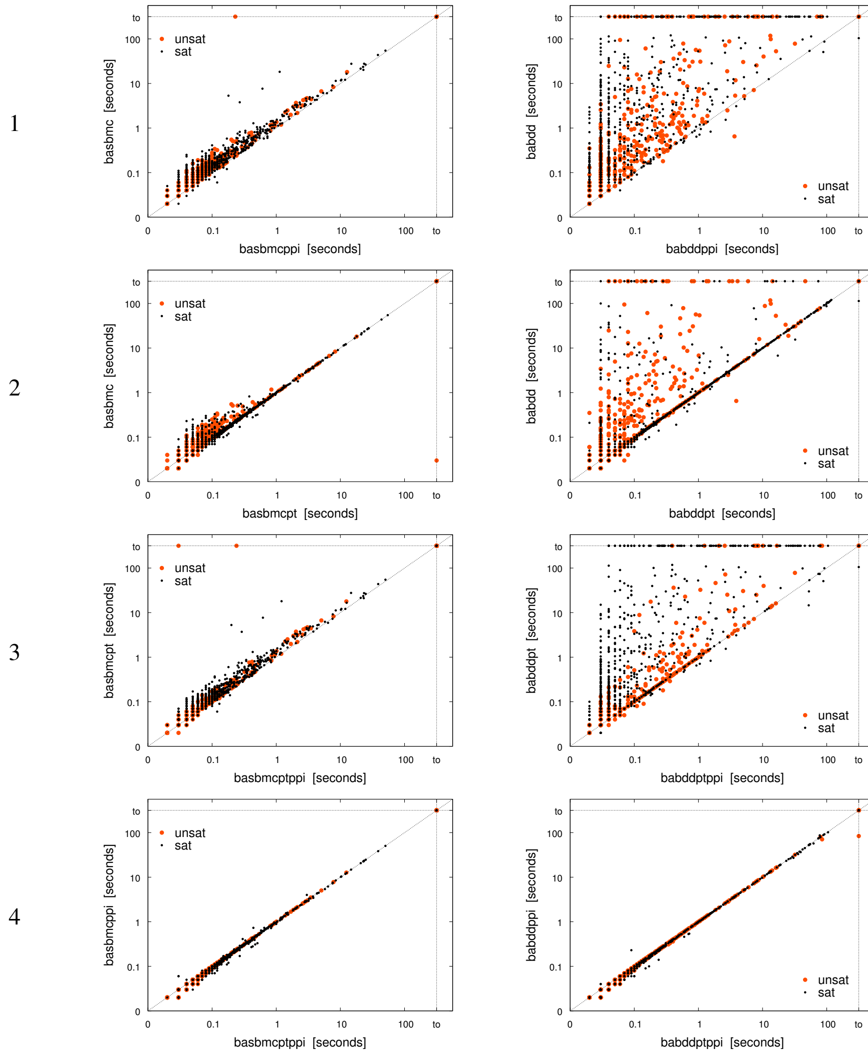

Fig. 5. Solving time of approaches with Boolean abstraction and combinations of pure literal simplification at the top- and/or prime implicant levels.

shows that the approach may result in substantial advantages, especially in the case of BDD-based reasoning, and the advantage mostly leverages on a generalization of the pure literal simplification rule to PSL. We also defined ways for computing unsatisfiable cores from the temporal solvers, and showed that their use may indeed reduce the search space, currently at the price of a penalty in performance.

In the future, a short term activity is to optimize the computation of conflict sets. In the longer term, the adoption of an SMT framework for temporal satisfiability suggests different research directions. First, we will investigate how to enable early pruning by means of incremental theory reasoning. Second, we will work on ways to combine

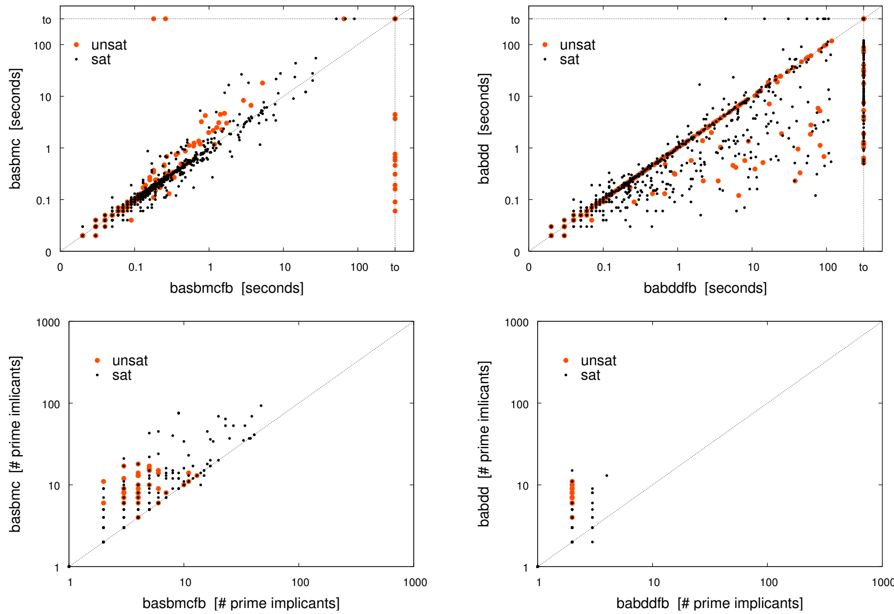

Fig. 6. The impact of feedback.

BDD-based and SAT-based techniques: in fact, a comparison of the two technologies (see the extended version of this paper), clearly highlights their complementarity. This includes not only identifying conditions that will suggest which one to use for which implicant, but also trying to let each method benefit from results obtained with the other. Finally, we will consider to exploit the temporal hierarchy to identify sufficient conditions for temporal satisfiability.

References

1. Accellera. Property specification language reference manual, version 1.1.
2. R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec Temporal Logic: A New Temporal Property-Specification Language. In *TACAS*, 2002.
3. I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic Sugar. In *CAV*, 2001.
4. S. Ben-David, R. Bloem, D. Fisman, A. Griesmayer, I. Pill, and S. Ruah. Automata Construction Algorithms Optimized for PSL, 2005. PROSYD deliverable D 3.2/4.
5. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, 1999.
6. A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2, 2006.
7. R. Bloem, A. Cimatti, I. Pill, M. Roveri, and S. Semprini. Symbolic Implementation of Alternating Automata. In *CIAA*, 2006.
8. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. MathSAT: Tight integration of SAT and mathematical decision procedures. *Journal of Automated Reasoning*, 35(1–3):265–293, 2005.
9. A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In *CAV*, 1999.
10. A. Cimatti, M. Roveri, S. Semprini, and S. Tonetta. From PSL to NBA: a modular symbolic encoding. In *FMCAD*, 2006.

11. A. Cimatti, M. Roveri, and S. Tonetta. Syntactic optimizations for PSL verification. In *TACAS*, 2007.
12. E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design*, 10(1):47–71, 1997.
13. S. Ben David and A. Orni. Property-by-Example guide: a handbook of PSL/Sugar examples, 2005. PROSYD deliverable D 1.1/3.
14. N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT*, 2003.
15. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of TCS, Volume B: Formal Models and Sematics*, pages 995–1072. 1990.
16. M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Trans. Comput. Logic*, 2(1):12–56, 2001.
17. A. Fuxman, L. Liu, , M. Pistore, M. Roveri, and J. Mylopoulos. Specifying and analyzing early requirements in Tropos: Some experimental results. In *RE*, 2003.
18. J.Y. Halpern, Z. Manna, and B.C. Moszkowski. A hardware semantics based on temporal intervals. In *ICALP*, 1983.
19. K. Heljanko, T. Junttila, M. Keinänen, M. Lange, and T. Latvala. Bounded model checking for weak alternating büchi automata. In *CAV*, 2006.
20. K. Heljanko, T. Junttila, and T. Latvala. Incremental and complete bounded model checking for full PLTL. In *CAV*, 2005.
21. O. Kupferman and M. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.
22. M. Lange and C. Stirling. Focus Games for Satisfiability and Completeness of Temporal Logic. In *LICS*, 2001.
23. O. Lichtenstein and A. Pnueli. Propositional Temporal Logics: Decidability and Completeness. *Logic Journal of the IGPL*, 8(1), 2000.
24. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
25. S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.
26. B.C. Moszkowski. A Hierarchical Completeness Proof for Propositional Interval Temporal Logic with Finite Time. *Journal of Applied Non-Classical Logics*, 14(1-2):55–104, 2004.
27. I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *DAC*, 2006.
28. A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.
29. A. Pnueli and A. Zaks. PSL Model Checking and Run-Time Verification Via Testers. In *FM*, pages 573–586, 2006.
30. R. Sebastiani and S. Tonetta. "More Deterministic" vs. "Smaller" Büchi Automata for Efficient LTL Model Checking. In *CHARME*, 2003.
31. M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *FMCAD*, 2000.
32. A. Sistla and E. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.
33. M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.
34. P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56(1/2):72–99, 1983.