Towards a Notion of Unsatisfiable Cores for LTL

Viktor Schuppan

FBK-irst, Via Sommarive 18, 38100 Trento, Italy. Email: schuppan@fbk.eu

Abstract. Unsatisfiable cores, i.e., parts of an unsatisfiable formula that are themselves unsatisfiable, have important uses in debugging specifications, speeding up search in model checking or SMT, and generating certificates of unsatisfiability. While unsatisfiable cores have been well investigated for Boolean SAT and constraint programming, the notion of unsatisfiable cores for temporal logics such as LTL has not received much attention. In this paper we investigate notions of unsatisfiable cores for LTL that arise from the syntax tree of an LTL formula, from converting it into a conjunctive normal form, and from proofs of its unsatisfiability. The resulting notions are more fine-granular than existing ones.

1 Introduction

Temporal logics such as LTL have become a standard formalism to specify requirements for reactive systems [37]. Hence, in recent years methodologies for property-based design based on temporal logics have been developed (e.g., [1]).

Increasing use of temporal logic requirements in the design process necessitates the availability of efficient validation and debugging methodologies. Vacuity checking [5, 31] and coverage [12] are complementary approaches developed in the context of model checking (e.g., [3]) for validating requirements given as temporal logic properties. However, with the exception of [13, 24], both vacuity and coverage assume presence of both a model and its requirements. Particularly in early stages of the design process the former might not be available. Satisfiability and realizability [38] checking are approaches that can handle requirements without a model being avaiable. Tool support for both is available (e.g., [8]).

Typically, unsatisfiability of a set of requirements signals presence of a problem; finding a reason for unsatisfiability can help with the ensuing debugging. In practice, determining a reason for unsatisfiability of a formula without automated support is often doomed to fail due to the sheer size of the formula. Consider, e.g., the EURAILCHECK project that developed a methodology and a tool for the validation of requirements [18]. Part of the methodology consists of translating the set of requirements given by a textual specification into a variant of LTL and subsequent checking for satisfiability; if the requirements are unsatisfiable, an unsatisfiable subset of them is returned to the user. The textual specification considered as a feasibility study is a few 100 pages long.

Another application for determining reasons for unsatisfiability are algorithms that find a solution to a problem in an iterative fashion. They start with a guess of a solution and check whether that guess is indeed a solution. If not,

rather than ruling out only that guess, they determine a reason for that guess not being a solution and rule out guesses doomed to fail for the same reason. Examples are verification with CEGAR (e.g., [20]) and SMT (e.g., [46]). Automated support for determining a reason for unsatisfiability is clearly essential.

Current implementations for satisfiability checking (e.g., [16]) point out reasons for unsatisfiability by returning a part of an unsatisfiable formula that is by itself unsatisfiable. This is called an unsatisfiable core (UC). However, these UCs are coarse-grained in the following sense. The input formula is a Boolean combination of temporal logic formulas. When extracting an UC current implementations do not look inside temporal subformulas: when, e.g., $\phi = (\mathbf{G}\psi) \wedge (\mathbf{F}\psi')$ is found to be unsatisfiable, then [16] will return ϕ as an UC irrespective of the complexity of ψ and ψ'. Whether the resulting core is inspected for debugging by a human or used as a filter in a search process by a machine: in either case a more fine-granular UC will likely make the corresponding task easier.

In this paper we take first steps to overcome the restrictions of UCs for LTL by investigating more fine-grained notions of UCs for LTL. We start with a notion based on the syntactic structure of the input formula where entire subformulas are replaced with 1 (true) or 0 (false) depending on the polarity of the corresponding subformula. We then consider conjunctive normal forms obtained by structure-preserving clause form translations [36]; the resulting notion of core is one of a subset of conjuncts. That notion is reused when looking at UCs extracted from resolution proofs from bounded model checking (BMC) [6] runs. We finally show how to extract an UC from a tableau proof [25] of unsatisfiability. All 4 notions can express UCs that are as fine-grained as the one based on the syntactic formula structure. The notion based on conjunctive normal forms provides more fine-grained resolution in the temporal dimension, and those based on BMC and on unsatisfied tableau proofs raise the hope to do even better. At this point we would like to emphasize the distinction between notions of UCs and methods to obtain them. While there is some emphasis in this paper on methods for UC extraction, here we see such methods only as a vehicle to suggest notions of UCs. We are not aware of similar systematic investigation of the notion of UC for LTL; for notions of cores for other formalisms, for application of UCs, and for technically related approaches such as vacuity checking see Sect. 8.

In the next Sect. 2 we state the preliminaries and in Sect. 3 we introduce some general notions. In Sect.s 4, 5, 6, and 7 we investigate UCs obtained by syntactic manipulation of parse trees, by taking subsets of conjuncts in conjunctive normal forms, by extracting resolution proofs from BMC runs, and by extraction from closed tableaux nodes. Related work is discussed in Sect. 8 before we conclude in Sect. 9. We do not provide a formalization of some parts and discussion of some aspects in this extended abstract but instead refer to the full version [40].

2 Preliminaries

In the following we give standard definitions for LTL, see, e.g., [3]. Let $\mathbb{B}$ be the set of Booleans, $\mathbb{N}$ the naturals, and AP a finite set of atomic propositions.

Definition 1 (LTL Syntax). *The set of* LTL formulas *is constructed inductively as follows. Boolean constants* $0, 1 \in \mathbb{B}$ *and atomic propositions* $p \in AP$ *are LTL formulas. If* ψ, ψ' *are LTL formulas, so are* $\neg\psi$, $\psi \vee \psi'$, $\psi \wedge \psi'$, $\mathbf{X}\psi$, $\psi\mathbf{U}\psi'$, $\psi\mathbf{R}\psi'$, $\mathbf{F}\psi$, *and* $\mathbf{G}\psi$. *We use* $\psi \rightarrow \psi'$ *as an abbreviation for* $\neg\psi \vee \psi'$, $\psi \leftarrow \psi'$ *for* $\psi \vee \neg\psi'$, *and* $\psi \leftrightarrow \psi'$ *for* $(\psi \rightarrow \psi') \wedge (\psi \leftarrow \psi')$.

The semantics of LTL formulas is defined on infinite words over the alphabet 2^{AP}. If π is an infinite word in $(2^{AP})^\omega$ and i is a position in $\mathbb{N}$, then $\pi[i]$ denotes the letter at the i-th position of π and $\pi[i, \infty]$ denotes the suffix of π starting at position i (inclusive). We now inductively define the semantics of an LTL formula on positions $i \in \mathbb{N}$ of a word $\pi \in (2^{AP})^\omega$:

Definition 2 (LTL Semantics).

$$
\begin{array}{llll}
(\pi, i) \models 1 & (\pi, i) \not\models 0 & (\pi, i) \models \psi\mathbf{U}\psi' \Leftrightarrow \exists i' \geq i \,.\, ((\pi, i') \models \psi' \wedge \forall i \leq i'' < i' \,.\, (\pi, i'') \models \psi) & \\
(\pi, i) \models p & \Leftrightarrow p \in \pi[i] & (\pi, i) \models \psi\mathbf{R}\psi' \Leftrightarrow \forall i' \geq i \,.\, ((\pi, i') \models \psi' \vee \exists i \leq i'' < i' \,.\, (\pi, i'') \models \psi) & \\
(\pi, i) \models \neg\psi & \Leftrightarrow (\pi, i) \not\models \psi & & (\pi, i) \models \mathbf{X}\psi \Leftrightarrow (\pi, i+1) \models \psi \\
(\pi, i) \models \psi \vee \psi' & \Leftrightarrow (\pi, i) \models \psi \text{ or } (\pi, i) \models \psi' & & (\pi, i) \models \mathbf{F}\psi \Leftrightarrow \exists i' \geq i \,.\, (\pi, i') \models \psi \\
(\pi, i) \models \psi \wedge \psi' & \Leftrightarrow (\pi, i) \models \psi \text{ and } (\pi, i) \models \psi' & & (\pi, i) \models \mathbf{G}\psi \Leftrightarrow \forall i' \geq i \,.\, (\pi, i') \models \psi
\end{array}
$$

An infinite word π satisfies *a formula* ϕ *iff the formula holds at the beginning of that word:* $\pi \models \phi \Leftrightarrow (\pi, 0) \models \phi$. *Then we call* π *a satisfying assignment to* ϕ.

Definition 3 (Satisfiability). *An LTL formula* ϕ *is* satisfiable *if there exists a word* π *that satisfies it:* $\exists \pi \in (2^{AP})^\omega \,.\, \pi \models \phi$; *it is* unsatisfiable *otherwise.*

Definition 4 (Negation Normal Form). *An LTL formula* ϕ *is in* negation normal form (NNF) $nnf(\phi)$ *if negations are applied only to atomic propositions.*

Definition 5 (Subformula). *Let* ϕ *be an LTL formula. The set of subformulas* $SF(\phi)$ *of* ϕ *is defined recursively as follows:*

$$
\begin{array}{llll}
\psi = b \text{ or } \psi = p & \text{with} & b \in \mathbb{B}, p \in AP & : \; SF(\psi) = \{\psi\} \\
\psi = \circ_1 \psi' & \text{with} & \circ_1 \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\} & : \; SF(\psi) = \{\psi\} \cup SF(\psi') \\
\psi = \psi' \circ_2 \psi'' & \text{with} & \circ_2 \in \{\vee, \wedge, \mathbf{U}, \mathbf{R}\} & : \; SF(\psi) = \{\psi\} \cup SF(\psi') \cup SF(\psi'')
\end{array}
$$

Definition 6 (Polarity). *Let* ϕ *be an LTL formula, let* $\psi \in SF(\phi)$. ψ *has* positive polarity *(+) in* ϕ *if it appears under an even number of negations,* negative polarity *(−) otherwise.*

We regard LTL formulas as trees, i.e., we don't take sharing of subformulas into account. We don't attempt to simplify formulas before or after UC extraction.

3 Notions and Concepts Related to UCs

In this section we discuss general notions in the context of UCs[1] independently of the notion of UC used. It is not a goal of this paper to formalize the notions below towards a general framework of UCs. Instead, in the remainder of this paper we focus on the case of LTL where instantiations are readily available.

[1] Terminology in the literature for these notions is diverse. We settled for the term "unsatisfiable core", which is used for such notions, e.g., in the context of Boolean satisfiability (e.g., [26,47]), SMT (e.g., [14]), and declarative specifications (e.g., [44]).

UCs, Irreducible UCs, and Least-Cost Irreducible UCs When dealing with *UCs* one typically considers an input ϕ (here: LTL formula) taken from a set of possible inputs Φ (here: all LTL formulas) and a Boolean-valued function *foo*[2] : $\Phi \mapsto \mathbb{B}$ with $foo(\phi) = 0$ (here: LTL satisfiability). The goal is to derive another input ϕ' (the UC) with $foo(\phi') = 0$ from ϕ s.t. 1. the derivation preserves a sufficient set of reasons for *foo* being 0 without adding new reasons, 2. the fact that $foo(\phi')$ is 0 is easier to see for the user than the fact that $foo(\phi)$ is 0, and 3. the derivation is such that preservance/non-addition of reasons for *foo* being 0 on ϕ and ϕ' can be understood by the user. Typically 1 and 3 are met by limiting the derivation to some set of operations on inputs that fulfills these criteria (here: syntactic weakening of LTL formulas). The remaining criterion 2 can be handled by assuming a cost function on inputs where lower cost provides some reason to hope for easier comprehension by the user (here: see below).

Assuming a set of inputs and a set of operations we can define the following notions. An input ϕ' is called a *core* of an input ϕ if it is derived by a sequence of such operations. ϕ' is an *unsatisfiable* core if ϕ' is a core of ϕ and $foo(\phi') = 0$. ϕ' is a *proper* unsatisfiable core if ϕ' is an unsatisfiable core of ϕ and is syntactically different from ϕ. Finally, ϕ' is an *irreducible* unsatisfiable core (IUC) if ϕ' is an unsatisfiable core of ϕ and there is no proper unsatisfiable core of ϕ'. Often IUCs are called minimal UCs and least-cost IUCs minimum UCs.

Cost functions often refer to some size measure of an input as suggested by a specific notion of core, e.g., the number of conjuncts when inputs are conjunctions of formulas and *foo* is satisfiability. We do not consider specific cost functions.

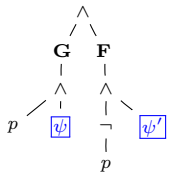
Granularity of a Notion of UC Clearly, the original input contains at least as much information as any of its UCs and, in particular, all reasons for being unsatisfiable. However, our goal when defining notions of UCs is to come up with derived inputs that make some of these reasons easier to see. Therefore we use the term *granularity* of a notion of core as follows. We wish to determine the relevance of certain aspects of an input to the input being unsatisfiable by the mere presence or absence of elements in the UC. In other words, we do not take potential steps of inference by the user into account. Hence, we say that one notion of core provides finer granularity than another if it provides at least as much information on the relevance of certain aspects of an input as the other. Consider, e.g., a notion of UC that takes a set of formulas as input and defines a core to be a subset of this set without proceeding to modify the member formulas versus a notion that also modifies the member formulas. Another example is a notion of UC for LTL that considers relevance of subformulas at certain points in time versus a notion that only either keeps or discards subformulas.

4 Unsatisfiable Cores via Parse Trees

In this section we consider UCs purely based on the syntactic structure of the formula. It is easy to see that, as is done, e.g., in some forms of vacuity checking

[2] Although we write *foo* we still say "unsatisfiable" core rather than "unfooable" core.

(a) A formula given as a parse tree



(b) and its UC.

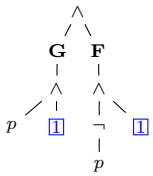


Fig. 1. Example of an UC via parse tree. Modified parts are marked blue boxed.

[31], replacing an occurrence of a subformula with positive polarity with 1 or replacing an occurrence of a subformula with negative polarity with 0 will lead to a weaker formula. This naturally leads to a definition of UC based on parse trees where replacing occurrences of subformulas corresponds to replacing subtrees.

Consider the following formula $\phi = (\mathbf{G}(p \wedge \psi)) \wedge (\mathbf{F}(\neg p \wedge \psi'))$ whose parse tree is depicted in Fig. 1 (a). The formula is unsatisfiable independent of the concrete (and possibly complex) subformulas ψ, ψ'. A corresponding UC with ψ, ψ' replaced with 1 is $\phi' = (\mathbf{G}(p \wedge 1)) \wedge (\mathbf{F}(\neg p \wedge 1))$, shown in Fig. 1 (b).

Hence, by letting the set of operations to derive a core be replacement of occurrences of subformulas of ϕ with 1 (for positive polarity occurrences) or 0 (for negative polarity occurrences), we obtain the notions of core, unsatisfiable core, proper unsatisfiable core, and irreducible unsatisfiable core *via parse tree*.

In the example above ϕ' is both a proper and an IUC of ϕ. Note that $(\mathbf{G}(p \wedge 1)) \wedge (\mathbf{F}(\neg p \wedge \psi'))$ and $(\mathbf{G}(p \wedge \psi)) \wedge (\mathbf{F}(\neg p \wedge 1))$ are UCs of ϕ, too, as is ϕ itself (and possibly many more when ψ and ψ' are taken into account).

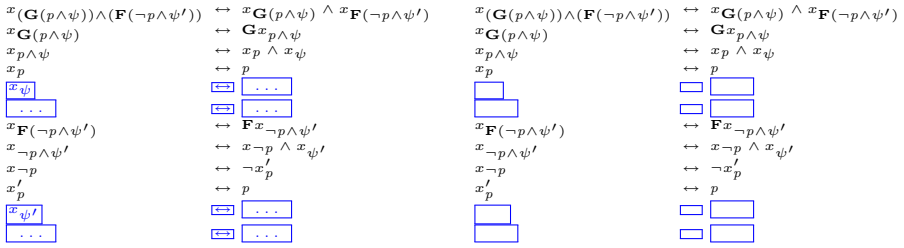
5 Unsatisfiable Cores via Definitional Conjunctive Normal Form

Structure preserving translations (e.g., [36]) of formulas into conjunctive normal form introduce fresh Boolean propositions for (some) subformulas that are constrained by one or more conjuncts to be 1 (if and) only if the corresponding subformulas hold in some satisfying assignment. In this paper we use the term definitional conjunctive normal form (dCNF) to make a clear distinction from the conjunctive normal form used in Boolean satisfiability (SAT), which we denote CNF. dCNF is often a preferred representation of formulas as it's typically easy to convert a formula into dCNF, the expansion in formula size is moderate, and the result is frequently amenable to resolution. Most important in the context of this paper, dCNFs yield a straightforward and most commonly used notion of core in the form of a (possibly constrained) subset of conjuncts.

5.1 Basic Form

Below we define the basic version of dCNF. It is well-known that ϕ and $dCNF(\phi)$ are equisatisfiable.

Definition 7 (Definitional Conjunctive Normal Form). *Let ϕ be an LTL formula over atomic propositions AP, let $x, x', \ldots \in X$ be fresh atomic proposi-*



(a) A formula given as a dCNF (b) and its UC.

Fig. 2. Example of UC via dCNF for $\phi = (\mathbf{G}(p \wedge \psi)) \wedge (\mathbf{F}(\neg p \wedge \psi'))$. The "..." stand for definitions of ψ, ψ', and their subformulas. Modified parts are marked blue boxed.

tions not in AP. $dCNF_{aux}(\phi)$ is a set of conjuncts containing one conjunct for each occurrence of a subformula ψ in ϕ as follows:

ψ	*Conjunct* $\in dCNF_{aux}(\phi)$	ψ	*Conjunct* $\in dCNF_{aux}(\phi)$
$b \in \mathbb{B}$	$x_\psi \leftrightarrow b$	$\circ_1 \psi'$ *with* $\circ_1 \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\}$	$x_\psi \leftrightarrow \circ_1 x_{\psi'}$
$p \in AP$	$x_\psi \leftrightarrow p$	$\psi' \circ_2 \psi''$ *with* $\circ_2 \in \{\vee, \wedge, \mathbf{U}, \mathbf{R}\}$	$x_\psi \leftrightarrow x_{\psi'} \circ_2 x_{\psi''}$

Then the definitional conjunctive normal form *of ϕ is defined as*

$$dCNF(\phi) \equiv x_\phi \wedge \mathbf{G} \bigwedge_{c \in dCNF_{aux}(\phi)} c$$

x_ϕ is called the root *of the dCNF. An occurrence of x on the left-hand side of a biimplication is a* definition *of x, an occurrence on the right-hand side a* use.

By letting the operations to derive a core from an input be the removal of elements of $dCNF_{aux}(\phi)$ we obtain the notions of core, unsatisfiable core, proper unsatisfiable core, and irreducible unsatisfiable core *via dCNF*. We additionally require that all conjuncts are discarded that contain definitions for which no (more) conjunct with a corresponding use exists.

We continue the example from Fig. 1 in Fig. 2. In the figure we identify an UC with its set of conjuncts. In Fig. 2 (b) the definitions for both ψ and ψ' and all dependent definitions are removed. As in Sect. 4 the UC shown in Fig. 2 (b) is an IUC with more UCs existing.

Correspondence Between Cores via Parse Trees and via dCNF Let ϕ be an LTL formula. From Def. 7 it is clear that there is a one-to-one correspondence between the nodes in the parse tree of ϕ and the conjuncts in its dCNF. Therefore, the conversion between the representation of ϕ as a parse tree and as a dCNF is straightforward. Remember that an UC of a parse tree is obtained by replacing an occurrence of a subformula ψ with 1 or 0, while an UC of a dCNF is obtained by removing the definition of ψ and all dependent definitions. Both ways to obtain an UC do not destroy the correspondence between parse trees and dCNFs. Hence, the notions of UC obtained via parse tree and via dCNF are equivalent.

5.2 Variants

We now examine some variants of Def. 7 w.r.t. the information contained in the UCs that they can yield. Each variant is built on top of the previous one.

Replacing Biimplications with Implications Definition 7 uses biimplication rather than implication in order to cover the case of both positive and negative polarity occurrences of subformulas in a uniform way. A seemingly refined variant is to consider both directions of that biimplication separately.[3] However, it is easy to see that in our setting of formulas as parse trees, i.e., without sharing of subformulas, each subformula has a unique polarity and, hence, only one direction of the biimplication will be present in an IUC. I.o.w., using 2 implications rather than a biimplication has no benefit in terms of granularity of the obtained cores.

Splitting Implications for Binary Operators We now consider left-hand and right-hand operands of the $\wedge$ and $\vee$ operators separately by splitting the implications for $\wedge$ and the reverse implications for $\vee$ into two. For example, $x_{\psi' \wedge \psi''} \rightarrow x_{\psi'} \wedge x_{\psi''}$ is split into $x_{\psi' \wedge \psi''} \rightarrow x_{\psi'}$ and $x_{\psi' \wedge \psi''} \rightarrow x_{\psi''}$. That variant can be seen not to yield finer granularity as follows. Assume an IUC *dCNF′* contains a conjunct $x_{\psi' \wedge \psi''} \rightarrow x_{\psi'}$ but not $x_{\psi' \wedge \psi''} \rightarrow x_{\psi''}$. The corresponding IUC *dCNF* based on Def. 7 must contain the conjunct $x_{\psi' \wedge \psi''} \rightarrow x_{\psi'} \wedge x_{\psi''}$ but will not contain a definition of $x_{\psi''}$. Hence, also in the IUC based on Def. 7, the subformula occurrence ψ'' can be seen to be irrelevant to that core. The case for $\vee$ is similar.

Temporal Unfolding Here we rewrite a conjunct for a positive polarity occurrence of an $\mathbf{U}$ subformula as its one-step temporal unfolding and an additional conjunct to enforce the desired fixed point. I.e., we replace a conjunct $x_{\psi' \mathbf{U} \psi''} \rightarrow x_{\psi'} \mathbf{U} x_{\psi''}$ with $x_{\psi' \mathbf{U} \psi''} \rightarrow x_{\psi''} \vee (x_{\psi'} \wedge \mathbf{X} x_{\psi' \mathbf{U} \psi''})$ and $x_{\psi' \mathbf{U} \psi''} \rightarrow \mathbf{F} x_{\psi''}$.

This can be seen to provide improved information for positive polarity occurrences of $\mathbf{U}$ subformulas in an IUC as follows. A dCNF for a positive occurrence of an $\mathbf{U}$ subformula $\psi' \mathbf{U} \psi''$ obtained without temporal unfolding as in the previous variant results (among others) in the following conjuncts: $c = x_{\psi' \mathbf{U} \psi''} \rightarrow x_{\psi'} \mathbf{U} x_{\psi''}$, $C''' = \{x_{\psi'} \rightarrow \ldots\}$, and $C'''' = \{x_{\psi''} \rightarrow \ldots\}$. An IUC based on that dCNF contains either 1. none of c, $c''' \in C'''$, $c'''' \in C''''$, 2. c, $c'''' \in C''''$, or 3. c, $c''' \in C'''$, $c'''' \in C''''$. O.t.o.h., a dCNF with temporal unfolding results in the conjuncts: $c' = x_{\psi' \mathbf{U} \psi''} \rightarrow x_{\psi''} \vee (x_{\psi'} \wedge \mathbf{X} x_{\psi' \mathbf{U} \psi''})$, $c'' = x_{\psi' \mathbf{U} \psi''} \rightarrow \mathbf{F} x_{\psi''}$, and C''', C'''' as before. An IUC based on that dCNF contains either 1. none of c', c'', $c''' \in C'''$, $c'''' \in C''''$, 2. c', $c''' \in C'''$, $c'''' \in C''' \in C''''$, 3. c'', $c'''' \in C''''$, or 4. c', c'', $c''' \in C'''$, $c'''' \in C''''$. For some $\mathbf{U}$ subformulas the additional case allows to distinguish between a situation where unsatisfiability arises based on impossibility of some finite unfolding of the $\mathbf{U}$ formula alone (case 2) and a situation where either some finite unfolding of that formula or meeting its eventuality are possible but not both (case 4).

As an illustration consider the following two formulas: 1. $(\psi' \mathbf{U} \psi'') \wedge (\neg\psi' \wedge \neg\psi'')$ and 2. $(\psi' \mathbf{U} \psi'') \wedge ((\neg\psi' \wedge \neg\psi'') \vee (\mathbf{G} \neg\psi''))$ An IUC obtained without temporal unfolding will contain c, $c''' \in C'''$, and $c'''' \in C''''$ in both cases while one obtained with temporal unfolding will contain c', $c''' \in C'''$, and $c'''' \in C''''$ in the first case and additionally c'' in the second case.

[3] While we defined biimplication as an abbreviation in Sect. 2, we treat it in this discussion as if it were available as an atomic operator for conjuncts of this form.

Temporal unfolding leading to more fine-granular IUCs can also be applied to negative polarity occurrences of **R** formulas in a similar fashion. Application to opposite polarity occurrences for **U** and **R** as well as to negative polarity occurrences of **F** and positive polarity occurrences of **G** subformulas is possible but does not lead to more fine-granular IUCs.

Splitting Conjunctions from Temporal Unfolding Our final variant splits the conjunctions that arise from temporal unfolding. In 4 of the 6 cases where temporal unfolding is possible this allows to distinguish the case where unsatisfiability is due to failure of unfolding in only the first time step that a **U**, **R**, **F**, or **G** formula is supposed (not) to hold in versus in the first and/or some later step. An example using an **U** formula is 1. $(\psi \mathbf{U} \psi') \wedge (\neg\psi \wedge \neg\psi')$ versus 2. $(\psi \mathbf{U} \psi') \wedge (\neg\psi' \wedge \mathbf{X}(\neg\psi \wedge \neg\psi'))$.

5.3 Comparison with Separated Normal Form

Separated Normal Form (SNF) [22, 23] is a conjunctive normal form for LTL originally proposed by Fisher to develop a resolution method for LTL.

The original SNF [22] separates past and future time operators by having a strict past time operator at the top level of the left-hand side of the implication in each conjunct and only Boolean disjunction and **F** operators on the right-hand side. We therefore restrict the comparison to two later variants [23,17] that allow propositions (present time formulas) on the left-hand side of the implications.

Compared to [22] the version of SNF in [23] also contains a simpler future time variant of SNF. [23] further refines our final variant in the last subsection in two ways. First, it applies temporal unfolding twice to **U**, weak **U**, and **G** formulas. This allows to distinguish failure of unfolding in the first, second, or some later step relative to the time when a formula is supposed to hold. Second, in some cases it has separate conjuncts for the absolute first and for later time steps. In the example $(p \mathbf{U} (q \wedge r)) \wedge ((\neg q) \wedge \mathbf{XG} \neg r)$ this allows to see that from the eventuality $q \wedge r$ the first operand is only needed in the absolute first time step, while the second operand leads to a contradiction in the second and later time steps. A minor difference is that atomic propositions are not defined using separate fresh propositions but remain unchanged at their place of occurrence.

[17] uses a less constrained version of [23]: right-hand sides of implications and bodies of **X** and **F** operators may now contain positive Boolean combinations of literals. This makes both above mentioned refinements unnecessary. The resulting normal form differs from our variant with temporal unfolding in 4 respects: 1. It works on NNF. 2. Positive Boolean combinations are not split into several conjuncts. 3. Fresh propositions are introduced for **U**, **R**, and **G** formulas representing truth in the next rather than in the current time step. Because of that, temporal unfolding is performed at the place of occurrence of the respective **U**, **R**, or **G** formula. 4. As in [23] atomic propositions remain unchanged at their place of occurrence. The combination of 2 and 4 leads to this variant of SNF yielding less information in the following example: $(\mathbf{F}(p \wedge q)) \wedge \mathbf{G} \neg p$. An IUC resulting from this variant of SNF will contain the conjunct $x \rightarrow \mathbf{F}(p \wedge q)$,

not making it clear that q is irrelevant for unsatisfiability. On the other hand, unsatisfiability due to failure of temporal unfolding at the first time point only can in some cases be distinguished from that at the first and/or or later time points, thus yielding more information; $(\mathbf{G}p) \wedge \neg p$ is an example for that.

6 Unsatisfiable Cores via Bounded Model Checking

By encoding existence of counterexamples of bounded length into a set of CNF clauses SAT-based Bounded Model Checking (BMC) (e.g., [6]) reduces model checking of LTL to SAT. Details on BMC can be found, e.g., in [7].

To prove correctness of properties (rather than existence of a counterexample) BMC needs to determine when to stop searching for longer counterexamples. The original works (e.g., [6]) imposed an upper bound derived from the graph structure of the model. A more refined method (e.g., [41]) takes a two-step approach: For the current bound on the length of counterexamples k, check whether there exists a path that 1. could possibly be extended to form a counterexample to the property and 2. contains no redundant part. If either of the two checks fails and no counterexample of length $\leq k$ has been found, then declare correctness of the property. As there are only finitely many states, step 2 guarantees termination. For other methods to prove properties in BMC see, e.g., [7].

By assuming a universal model BMC provides a way to determine LTL satisfiability (used, e.g., in [16]) and so is a natural choice to investigate notions of UCs. Note that in BMC, as soon as properties are not just simple invariants of the form $\mathbf{G}p$, already the first part of the above check for termination might fail. That observation yields an incomplete method to determine LTL satisfiability. We first sketch the method and then the UCs that can be extracted.

The method essentially employs dCNF with splitting conjunctions from temporal unfolding to generate a SAT problem in CNF as follows: 1. Pick some bound k. 2. To obtain the set of variables instantiate the members of X for each time step $0 \leq i \leq k+1$ and of AP for $0 \leq i \leq k$. We indicate the time step by using superscripts. 3. For the set of CNF clauses instantiate each conjunct in $dCNF_{aux}$ not containing a $\mathbf{F}$ or $\mathbf{G}$ operator once for each $0 \leq i \leq k$. Add the time 0 instance of the root of the dCNF, x^0_ϕ, to the set of clauses. 4. Replace each occurrence of $\mathbf{X}x^i_\psi$ with x^{i+1}_ψ. Note that at this point all temporal operators have been removed and we indeed have a CNF. Now if for any such k the resulting CNF is unsatisfiable, then so is the original LTL formula. The resulting method is very similar to BMC in [29] when checking for termination by using the completeness formula only rather than completeness and simplepath formula together (only presence of the latter can ensure termination).

Assume that for an LTL formula ϕ the above method yields an unsatisfiable CNF for some k and that we are provided with an IUC of that CNF as a subset of clauses. It is easy to see that we can extract an UC of the granularity of a dCNF with splitting conjunctions from temporal unfolding by considering any dCNF conjunct to be part of the UC iff for any time step the corresponding CNF clause is present in the CNF IUC. Note that the CNF IUC provides potentially

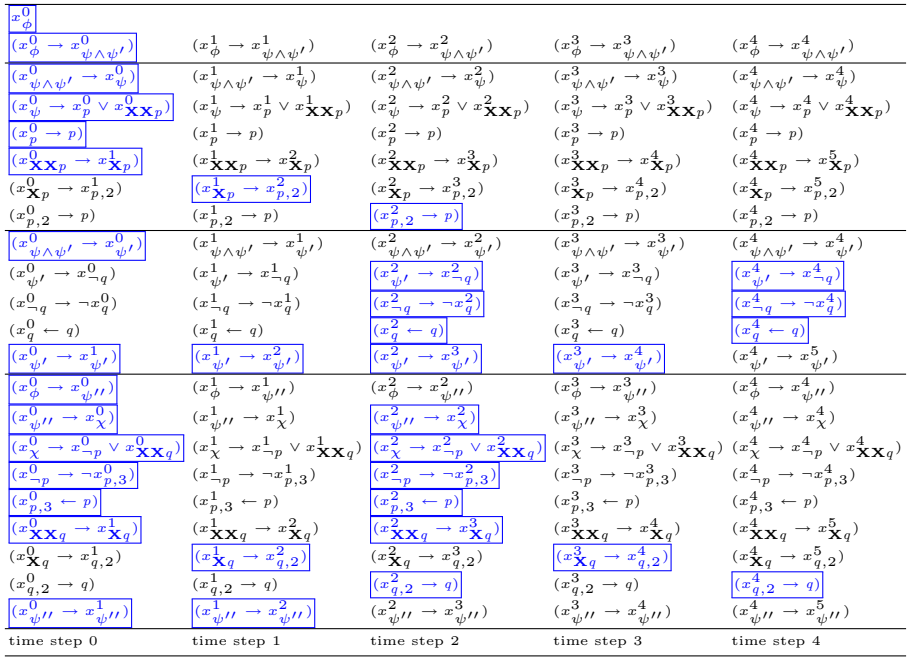


Fig. 3. Example of an UC via BMC. Clauses that form the SAT IUC are marked blue boxed. The input formula is $\phi = ((p \vee \mathbf{XX}p) \wedge (\mathbf{G}\neg q)) \wedge (\mathbf{G}(p \rightarrow \mathbf{XX}q))$. We abbreviate: $\psi = p \vee \mathbf{XX}p$, $\psi' = \mathbf{G}\neg q$, $\psi'' = \mathbf{G}(p \rightarrow \mathbf{XX}q)$, and $\chi = p \rightarrow \mathbf{XX}q$.

finer granularity in the temporal dimension: the CNF IUC contains information about the relevance of parts of the LTL formula to unsatisfiability at each time step. Contrary to the notions of UC in the previous section (see [40]) we currently have no translation back to LTL for this level of detail. Once such translation has been obtained it makes sense to define removal of clauses from the CNF as the operation to derive a core thus giving the notions of core, unsatisfiable core, proper unsatisfiable core, and irreducible unsatisfiable core via BMC.

As an example consider $\phi = ((p \vee \mathbf{XX}p) \wedge (\mathbf{G}\neg q)) \wedge (\mathbf{G}((\neg p) \vee \mathbf{XX}q))$. The translation into a set of CNF clauses and the CNF IUC are depicted in Fig. 3. Extracting an UC at the granularity of a dCNF with splitting conjunctions from temporal unfolding results in ϕ itself. However, the CNF IUC shows that, e.g., the occurrence of p in the last conjunct is relevant only at time steps 0 and 2 and the occurrence of q in the middle conjunct matters at time steps 2 and 4.

7 Unsatisfiable Cores via Tableaux

Tableaux are widely used for temporal logics. Most common methods in BDD-based symbolic model checking (e.g., [19]) and in explicit state model checking (e.g., [25]) of LTL rely on tableaux. Therefore tableaux seem to be a natural candidate for investigating notions of UCs.

In this section we only consider formulas in NNF. We assume that the reader is familiar with standard tableaux constructions for LTL such as [25]. We differ from, e.g., [25] in that we retain and continue to expand closed nodes during tableau construction and only take them into account when searching for satisfied paths in the tableau. We fix some terminology. A node in a tableau is called 1. *initial* if it is a potential start, 2. *closed* if it contains a pair of contradicting literals or 0, 3. *terminal* if it contains no obligations left for the next time step, and 4. *accepting* (for some $\mathbf{U}$ or $\mathbf{F}$ formula), if it either contains both the formula and its eventuality or none of the two. A path in the tableau is *initialized* if it starts in an initial node and *fair* if it contains infinitely many occurrences of accepting nodes for each $\mathbf{U}$ and $\mathbf{F}$ formula. A path is *satisfied* if 1. it is initialized, 2. it contains no closed node, and 3. it is finite and ends in a terminal node or infinite and fair. A tableau is *satisfied* iff it contains a satisfied path.

Intuitively, closed nodes are what prevents satisfied paths. For an initialized path to a terminal node it is obvious that a closed node on that path is a reason for that path not being satisfied. A similar statement holds for initialized infinite fair paths that contain closed nodes. That leaves initialized infinite unfair paths that do not contain a closed node. Still, also in that case closed nodes hold information w.r.t. non-satisfaction: an unfair path contains at least one occurrence of an $\mathbf{U}$ or $\mathbf{F}$ formula whose eventuality is not fulfilled. The tableau construction ensures that for each node containing such an occurrence there will also be a branch that attempts to make the eventuality 1. That implies that the reason for failure of fulfilling eventualities is not to be found on the infinite unfair path, but on its unsuccessful branches. Hence, we focus on closed nodes to extract sufficient information why a formula in unsatisfiable.

The procedure to extract an UC now works as follows. It first chooses a subset of closed nodes that act as a barrier in that at least one of these nodes is in the way of each potentially satisfied path in the tableau. Next it chooses a set of occurrences of contradicting literals and 0 s.t. this set represents a contradiction for each of the selected closed tableau nodes. As these occurrences of subformulas make up the reason for non-satisfaction, they and, transitively, their fathers in the parse tree of the formula are marked and retained while all non-marked occurrences of subformulas in the parse tree are discarded and dangling edges are rerouted to fresh nodes representing 1.

As an example consider the tableau in Fig. 4 for $\phi = \mathbf{X}(((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q))) \vee (p \wedge (\mathbf{X}p) \wedge \neg p \wedge \mathbf{X}(\neg p)))$. Choosing $\{n_1, n_3\}$ as the subset of closed nodes and the occurrences of q, $\neg q$ in n_1 and p, $\neg p$ in n_3 leads to $\mathbf{X}(((\mathbf{G}(1 \wedge q \wedge 1)) \wedge (\mathbf{F}(1 \wedge \neg q))) \vee (p \wedge 1 \wedge \neg p \wedge 1))$ as UC. More UCs result from choosing p, $\neg p$ also in n_1, or n_5 instead of n_3.

In the full version [40] we show that the set of UCs that can be extracted in that way is equivalent to the set of UCs via parse trees. However, we conjecture that the procedure can be extended to extract UCs that indicate relevance of subformulas not only at finitely many time steps as in Sect. 6 but at semilinearly many. Given, e.g., $\phi = p \wedge (\mathbf{G}(p \rightarrow \mathbf{XX}p)) \wedge (\mathbf{F}(\neg p \wedge \mathbf{X}\neg p))$, we would like to see that some subformulas are only relevant at every second time step.

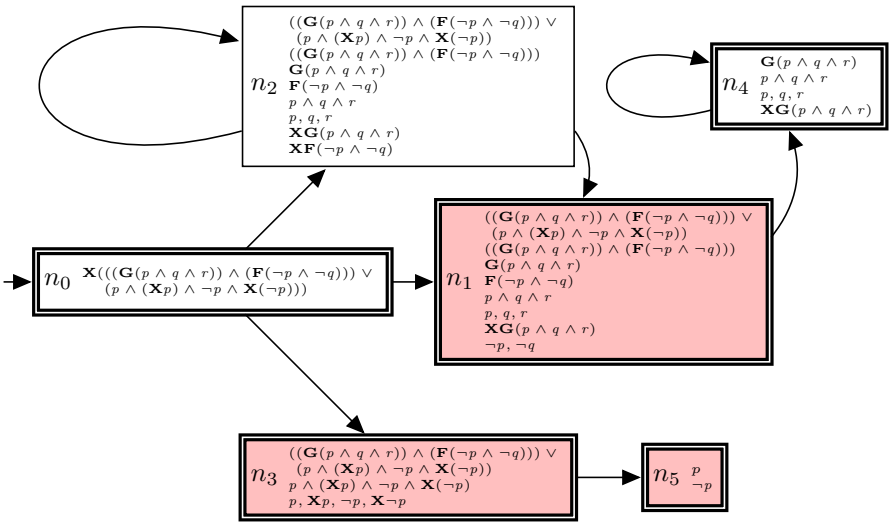


Fig. 4. Example of an unsatisfied tableau for $\phi = \mathbf{X}(((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q))) \vee (p \wedge (\mathbf{X}p) \wedge \neg p \wedge \mathbf{X}(\neg p)))$. The initial node n_0 has an incoming arrow, closed nodes n_1, n_3, n_5 are filled red, accepting nodes (all but n_2) have thick double lines, and the terminal node n_5 has no outgoing arrow.

8 Related Work

Notions of Core [16] proposes a notion of UCs of LTL formulas. The context in that work is a method for satisfiability checking of LTL formulas by using Boolean abstraction (e.g., [30]). As a consequence, an UC in [16] is a subset of the set of top-level temporal formulas, potentially leading to very coarse cores.

SAT uses CNF as a standard format and UCs are typically subsets of clauses (e.g., [9]). Similarly, in constraint programming, an UC is a subset of the set of input constraints (e.g., [4]); [27] suggests a more fine-grained notion based on unsatisfiable tuples. Finally, also in SMT UCs are subsets of formulas (e.g., [14]).

For realizability [38] of a set of LTL formulas, partitioned into a set of assumptions and a set of guarantees, [15] suggests to first reduce the number of guarantees and then, additionally, to reduce the set of assumptions.

Extracting Cores from Proofs In [34] a successful run of a model checker, which essentially corresponds to an unsatisfied tableau, is used to extract a temporal proof from the tableau [25] as a certificate that the model fulfills the specification. [32] generates certificates for successful model checking runs of μ-calculus specifications. [39] extracts UCs from unsatisfied tableaux to aid debugging in the context of description logics. Extracting a core from a resolution proof is an established technique in propositional SAT (e.g., [26,47]). In SMT UCs from SAT can be used to extract UCs for SMT [14]. Extraction from proofs is also used in vacuity checking [33,43].

Applications of Cores Using UCs to help a user debugging by pointing out a subset of the input as part of some problem is stated explicitly as motivation in many works on cores, e.g., [10, 4, 9, 47].

[42] presents a method for debugging declarative specifications by translating an abstract syntax tree (AST) of an inconsistent specification to CNF, extracting an UC from the CNF, and mapping the result back to AST highlighting only the relevant parts. That work has some similarities with our discussion; however, there are also a number of differences. 1. The exposition in [42] is for first order relational logic and generalizes to languages that are reducible to SAT, while our logic is LTL. 2. The motivation and focus of [42] is on the method of core extraction, and it is accompanied by some experimental results. The notion of a core as parts of the AST is taken as a given. On the other hand, our focus is on investigating different notions of cores and on comparing the resulting information that can be gained. 3. [42] does not consider tableaux. [44] suggests improved algorithms for core extraction compared to [42]; the improved algorithms produce IUCs at a reasonable cost by using mechanisms similar to [47, 21]. The scope of the method is extended to specification languages with a (restricted) translation to logics with resolution engine.

Examples of using UCs for debugging in description logics and ontologies are [39, 45]. For temporal logic, the methodology proposed in [35] suggests to return a subset of the specification in case of a problem. For [15] see above.

The application of UCs as filters in an iterative search is mentioned in Sect. 1.

Vacuity Checking Vacuity checking (e.g., [5,31]) is a technique in model checking to determine whether a model satisfies the specification in an undesired way. Vacuity asks whether there exists a strengthening of a specification s.t. the model still passes that strengthened specification. The original notion of vacuity from [5, 31] replaces occurrences of subformulas in the specification with 0 or 1 depending on polarity and is, therefore, related to the notion of UC in Sect. 4.

The comparison of notions of vacuity with UCs is as follows: 1. Vacuity is normally defined with respect to a specific model. [13] proposes vacuity without design as a preliminary check of vacuity: a formula is vacuous without design if it fulfills a variant of itself to which a strengthening operation has been applied. [24] extends that into a framework for inherent vacuity (see below). 2. Vacuity is geared to answer whether there exists at least one strengthening of the specification s.t. the model still satisfies the specification. For that it is sufficient to demonstrate that with a single strengthening step. The question of whether and to which extent the specification should be strengthened is then usually left to the designer. In core extraction one would ideally like to obtain IUCs and do so in a fully automated fashion. [28,13] discuss mutual vacuity, i.e., vacuity w.r.t. sets of subformulas. [11] proceeds to obtain even stronger passing formulas combining several strengthened versions of the original formula. 3. Vacuity typically focuses on strengthening a formula while methods to obtain UCs use weakening. The reason is that in the case of a failing specification a counterexample is considered to be more helpful. Still, vacuity is defined in, e.g., [5, 31, 24] w.r.t. both passing and failing formulas.

[24] proposes a framework to identify inherent vacuity, i.e., specifications that are vacuous in any model. The framework has 4 parameters: 1. vacuity type: occurrences of subformulas, sharing of subformulas, etc., 2. equivalence type: closed or open systems, 3. tightening type: equivalence or preservance of satisfiability/realizability, and 4. polarity type: strengthening or weakening. Our notion of UCs via parse tree is very closely related to the following instance of that framework. Let the vacuity type be that of replacing occurrences of subformulas with 1 or 0 depending on polarity [5], systems be closed, tightening type be equivalence or preservance of unsatisfiability, and polarity type be weakening. Then it is straightforward to show that, given a proper UC ϕ' via parse tree of some unsatisfiable formula ϕ, 1. ϕ is inherently vacuous, and 2. ϕ' is an IUC iff it is not inherently vacuous. [24] focuses on satisfiable/realizable instances and doesn't make a connection to the notion of unsatisfiable or unrealizable cores.

[43] exploits resolution proofs from BMC runs in order to extract information on vacuity including information on relevance of subformulas at specific time steps in a fashion related to our extraction of UCs in Sect. 6. A difference is that the presentation in [43] only explains how to obtain the notion of k-step vacuity from some BMC run with bound k but leaves it unclear how to make the transition from the notion of k-step vacuity to the notion of vacuity and, similarly, how to aggregate results on the relevance of subformulas at specific time steps over results for different ks; our method of UC extraction can return an UC as soon as the generated CNF is unsatisfiable for some k.

Other notions and techniques might be suitable to be carried over from vacuity detection to UCs for LTL and vice versa. E.g., [2] extends vacuity to consider sharing of subformulas. We are not aware of any work in vacuity that takes the perspective of searching an UC of an LTL formula or considers dCNFs as we do.

9 Conclusion

We suggested notions of unsatisfiable cores for LTL formulas that provide strictly more fine-grained information than the (few) previous notions. While basic notions turned out to be equivalent, some variants were shown to provide or potentially provide more information, in particular, in the temporal dimension.

We stated initially that we see methods of UC extraction as a means to suggest notions of UCs. Indeed, it turned out that each method for core extraction suggested a different or a more fine-grained notion of UC that should be taken into account. It seems to be likely, though, that some of the more fine-grained notions can be obtained also with other UC extraction methods.

Directions for future work include defining and obtaining the more fine-grained notions of UC suggested at the end of Sect.s 6 and 7, investigating the notion of UC that results from temporal resolution proofs, taking sharing of subformulas into account, and extending the notions to realizability. Equally important are efficient implementations. Finally, while in theory two algorithms to obtain UCs might be able to come up with the same set of UCs, practical im-

plementations could yield different UCs due to how non-determinism is resolved; hence, an empirical evaluation of the usefulness of the resulting UCs is needed.

Acknowledgements The author thanks the research groups at FBK and Verimag for discussions and comments, esp., A. Cimatti, M. Roveri, and S. Tonetta. Part of this work was carried out while the author was at Verimag/CNRS. He thanks O. Maler for providing the freedom to pursue this work. Finally, the author thanks the Provincia Autonoma di Trento for support (project EMTELOS).

References

1. Prosyd. http://www.prosyd.org/.
2. R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. Enhanced Vacuity Detection in Linear Temporal Logic. In *CAV*, 2003.
3. C. Baier and J. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4. R Bakker, F. Dikker, F. Tempelman, and P Wognum. Diagnosing and Solving Over-Determined Constraint Satisfaction Problems. In *IJCAI*, 1993.
5. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient Detection of Vacuity in Temporal Model Checking. *Formal Methods in System Design*, 18(2), 2001.
6. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *TACAS*, 1999.
7. A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. Linear Encodings of Bounded LTL Model Checking. *Logical Methods in Computer Science*, 2(5), 2006.
8. R. Bloem, R. Cavada, I. Pill, M. Roveri, and A. Tchaltsev. RAT: A Tool for the Formal Analysis of Requirements. In *CAV*, 2007.
9. R. Bruni and A. Sassano. Restoring Satisfiability or Maintaining Unsatisfiability by finding small Unsatisfiable Subformulae. In *SAT*, 2001.
10. J. Chinneck and E. Dravnieks. Locating Minimal Infeasible Constraint Sets in Linear Programs. *ORSA Journal on Computing*, 3(2), 1991.
11. H. Chockler, A. Gurfinkel, and O. Strichman. Beyond Vacuity: Towards the Strongest Passing Formula. In *FMCAD*, 2008.
12. H. Chockler, O. Kupferman, and M. Vardi. Coverage metrics for temporal logic model checking. *Formal Methods in System Design*, 28(3), 2006.
13. H. Chockler and O. Strichman. Easier and More Informative Vacuity Checks. In *MEMOCODE*, 2007.
14. A. Cimatti, A. Griggio, and R. Sebastiani. A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories. In *SAT*, 2007.
15. A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic Information for Realizability. In *VMCAI*, 2008.
16. A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. Boolean Abstraction for Temporal Logic Satisfiability. In *CAV*, 2007.
17. A. Cimatti, M. Roveri, and D. Sheridan. Bounded Verification of Past LTL. In *FMCAD*, 2004.
18. A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. From Informal Requirements to Property-Driven Formal Validation. In *FMICS*, 2008. To appear.
19. E. Clarke, O. Grumberg, and K. Hamaguchi. Another Look at LTL Model Checking. *Formal Methods in System Design*, 10(1), 1997.

20. E. Clarke, M. Talupur, H. Veith, and D. Wang. SAT Based Predicate Abstraction for Hardware Verification. In *SAT*, 2003.
21. N. Dershowitz, Z. Hanna, and A. Nadel. A Scalable Algorithm for Minimal Unsatisfiable Core Extraction. In *SAT*, 2006.
22. M. Fisher. A Resolution Method for Temporal Logic. In *IJCAI*, 1991.
23. M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Trans. Comput. Log.*, 2(1), 2001.
24. D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M. Vardi. A Framework for Inherent Vacuity. In *HVC*, 2008. To appear.
25. R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV*, 1995.
26. E. Goldberg and Y. Novikov. Verification of Proofs of Unsatisfiability for CNF Formulas. In *DATE*, 2003.
27. É. Grégoire, B. Mazure, and C. Piette. MUST: Provide a Finer-Grained Explanation of Unsatisfiability. In *CP*, 2007.
28. A. Gurfinkel and M. Chechik. How Vacuous Is Vacuous? In *TACAS*, 2004.
29. K. Heljanko, T. Junttila, and T. Latvala. Incremental and Complete Bounded Model Checking for Full PLTL. In *CAV*, 2005.
30. D. Kroening and O. Strichman. *Decision Procedures*. Springer, 2008.
31. O. Kupferman and M. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2), 2003.
32. K. Namjoshi. Certifying Model Checkers. In *CAV*, 2001.
33. K. Namjoshi. An Efficiently Checkable, Proof-Based Formulation of Vacuity in Model Checking. In *CAV*, 2004.
34. D. Peled, A. Pnueli, and L. Zuck. From Falsification to Verification. In *FSTTCS'01*.
35. I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *DAC*, 2006.
36. D. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. *J. Symb. Comput.*, 2(3), 1986.
37. A. Pnueli. The Temporal Logic of Programs. In *FOCS*, 1977.
38. A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *POPL*, 1989.
39. S. Schlobach and R. Cornet. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In *IJCAI*. Morgan Kaufmann, 2003.
40. V. Schuppan. Towards a Notion of Unsatisfiable Cores for LTL. Technical Report 200901000, Fondazione Bruno Kessler, 2009.
41. M. Sheeran, S. Singh, and G. Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. In *FMCAD*, 2000.
42. I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. Debugging Overconstrained Declarative Models Using Unsatisfiable Cores. In *ASE*, 2003.
43. J. Simmonds, J. Davies, A. Gurfinkel, and M. Chechik. Exploiting Resolution Proofs to Speed Up LTL Vacuity Detection for BMC. In *FMCAD*, 2007.
44. E. Torlak, F. Chang, and D. Jackson. Finding Minimal Unsatisfiable Cores of Declarative Specifications. In *FM*, 2008.
45. H. Wang, M. Horridge, A. Rector, N. Drummond, and J. Seidenberg. Debugging OWL-DL Ontologies: A Heuristic Approach. In *ISWC*, 2005.
46. S. Wolfman and D. Weld. The LPSAT Engine & Its Application to Resource Planning. In *IJCAI*. Morgan Kaufmann, 1999.
47. L. Zhang and S. Malik. Extracting Small Unsatisfiable Cores from Unsatisfiable Boolean Formula. Presented at *SAT*, *2003*.