Fondazione Bruno Kessler
Via Sommarive 18
38123 Trento Povo (TN), Italy
http://www.fbk.eu/

Towards a Notion of Unsatisfiable Cores for LTL

V. Schuppan

April 2009

Technical Report # FBK-200901000

Id: fsen09.tex,v 1.44 2009-04-22 14:20:40 schuppan Exp

Towards a Notion of Unsatisfiable Cores for LTL[⋆]

Viktor Schuppan

FBK-irst, Via Sommarive 18, 38123 Trento Povo (TN), Italy
`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

The importance of requirements to delivering high quality hardware and software products on time is being increasingly recognized in industry. Temporal logics such as LTL have become a standard formalism to specify requirements for reactive systems [Pnu77]. Consequentially, in recent years methodologies for property-based design based on temporal logics have been developed (e.g., [pro]).

Increasing use of temporal logic requirements in the design process necessitates the availability of efficient validation and debugging methodologies. Vacuity checking [BBDER01, KV03] and coverage [CKV06] are complementary approaches developed in the context of model checking [CE81,QS82,CGP99,BK08] for validating requirements given as temporal logic properties. They focus on the relation between the model and its requirements beyond the simple correctness relation as established by model checking. However, with the exception of [CS07,FKSFV08], 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 [PR89, ALW89] checking are approaches that can handle requirements without a model being avaiable. Tool support for both is available (e.g., [BCP$^+$07]).

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.

[⋆] This is the full version of [Sch09].

Consider, e.g., the EURAILCHECK project [eur] that developed a methodology and a tool for the validation of requirements in the context of railway signaling and control [CRST08b]. Part of the methodology consists of translating the set of (implicitly conjoined) requirements given by a textual specification into a variant [CRST08c] of LTL whose atoms are constraints in a first order theory and subsequent checking for satisfiability; if the requirements turn out to be unsatisfiable, an unsatisfiable subset of them is returned to the user. The textual specification that was considered as a feasibility study in the project is a few hundred pages long.

Another application for determining reasons for unsatisfiability are algorithms that try to find a solution to a problem in an iterative fashion. These algorithms 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 all guesses that are doomed to fail for the same reason. Two examples are found in verification algorithms using counterexample guided abstraction refinement (CEGAR) (e.g., [CTVW03]) and in SMT (e.g., [WW99]). Here, too, automated support for determining a reason for unsatisfiability is clearly essential.

Current implementations for satisfiability checking (e.g., [CRST07]) 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 [CRST07] 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 [PG86]; 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) [BCCZ99] runs. We finally show how to extract an UC from a tableau proof [GPVW95] 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 specification formalisms, for application of UCs, and for technically related approaches such as vacuity checking see Sect. 9.

The paper is structured as follows. 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. Some aspects that are not treated in-depth in this paper are mentioned in Sect. 8. Related work is discussed in Sect. 9 before we conclude in Sect. 10.

2 Preliminaries

In the following we give standard definitions for LTL [Pnu77], see, e.g., [Eme90, BK08]. 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. The Boolean constants* 0 *(false),* 1 *(true)* $\in \mathbb{B}$ *and any atomic proposition* $p \in AP$ *are LTL formulas. If* ψ, ψ' *are LTL formulas, so are* $\neg\psi$ *(negation),* $\psi \vee \psi'$ *(or),* $\psi \wedge \psi'$ *(and),* $\mathbf{X}\psi$ *(next time),* $\psi\mathbf{U}\psi'$ *(until),* $\psi\mathbf{R}\psi'$ *(releases),* $\mathbf{F}\psi$ *(finally), and* $\mathbf{G}\psi$ *(globally). We use* $\psi \rightarrow \psi'$ *(implication) as an abbreviation for* $\neg\psi \vee \psi'$, $\psi \leftarrow \psi'$ *(reverse implication) for* $\psi \vee \neg\psi'$, *and* $\psi \leftrightarrow \psi'$ *(biimplication) 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).

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

An infinite word π satisfies *a formula* ϕ *iff the formula holds at the beginning of that word:* $\pi \models \phi \Leftrightarrow (\pi,0) \models \phi$. *In that case we also 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.*

Conversion of an LTL formula into NNF can be achieved by pushing negations inward and dualizing operators (replacing them with their duals), see, e.g., [BK08].

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 independently of the precise notion of UC used. It is not a goal of this paper to formalize the notions below towards a general framework of UCs; in particular, we do not specify formal requirements on the set of operations to take an input to a core. Instead, in the remainder of this paper we focus on the case of LTL where suitable instantiations are readily available.

The terminology used in the literature for these notions is diverse. We decided to settle for the at least somewhat common term "unsatisfiable core" that has been used for such notions, e.g., in the context of Boolean satisfiability (e.g., [GN03, ZM03a, ZM03b]), SMT (e.g., [CGS07]), and declarative specifications (e.g., [TCJ08]).

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[1] $: \Phi \mapsto \mathbb{B}$ 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

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

the fact that *foo*(ϕ) 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*(ϕ') $= 0$. 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. An example is the number of conjuncts when inputs are conjunctions of formulas and *foo* is satisfiability. This example hints that simplistic measures might be suboptimal as the complexity of the conjuncts is not taken into account. In the remainder of this paper we do not consider specific cost functions; in most of the notions used some straightforward size measure is decreased by each of the above mentioned operations.

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 notion if it provides at least as much information on the relevance of certain aspects of an input as the other notion.

As an example consider a notion of UC that takes a set of formulas as input and defines a core to be a subset of this set of formulas without proceeding to modify the member formulas versus a notion that also modifies the members of the input set of 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.

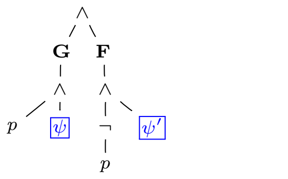
4 Unsatisfiable Cores via Parse Trees

4.1 Intuition and Example

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

[KV03], 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).


(a) A formula given as a parse tree

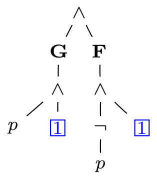

(b) and its UC.

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

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).

4.2 Formalization

Definition 7 (Parse Tree). *Let ϕ be an LTL formula. The* parse tree *of ϕ, $pt_\phi = (V_{pt_\phi}, E_{pt_\phi})$, is a tree with a non-empty set of nodes V_{pt_ϕ}, a set of edges E_{pt_ϕ}, root $root(pt_\phi) \in V_{pt_\phi}$, a labeling $op_{pt_\phi} : V_{pt_\phi} \mapsto \{\neg, \vee, \wedge, \mathbf{X}, \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{G}\} \cup AP \cup \mathbb{B}$ that maps inner nodes $V^i_{pt_\phi}$ to operators and leaf nodes $V^l_{pt_\phi}$ to Boolean constants and atomic propositions such that a node v labeled with a unary operator has one child $left_{pt_\phi}(v)$ and a node labeled with a binary operator has two children $left_{pt_\phi}(v), right_{pt_\phi}(v)$. The father of a non-root node v is given by $father_{pt_\phi}(v)$. Each node v represents a formula $f_{pt_\phi}(v)$ in the natural fashion. pt_ϕ represents the formula given by its root node: $f(pt_\phi) = f_{pt_\phi}(root(pt_\phi))$. The polarity of a node $polarity_{pt_\phi}(v)$ is the polarity of its subformula $f_{pt_\phi}(v)$ in ϕ.*

Definition 8 (Core of a Parse Tree). *Let pt, pt' be parse trees. pt' is a* core *of pt if 1. nodes and edges of pt' are a subset of those of pt: $V_{pt'} \subseteq V_{pt}$,*

$E_{pt'} \subseteq E_{pt}$*, 2.* pt *and* pt' *have the same root node:* $root(pt') = root(pt)$*, 3. the labeling of inner nodes of* pt' *agrees with the labeling of the corresponding nodes in* pt*:* $\forall v \in V^i_{pt'} \, . \, op_{pt'}(v) = op_{pt}(v)$*, and 4. the labeling of leaf nodes of* pt' *either agrees with the labeling of the corresponding nodes in* pt *or is* 1 *(resp.* 0*) if* v *has positive (resp. negative) polarity:* $\forall v \in V^l_{pt'} \, . \, (op_{pt'}(v) = op_{pt}(v)) \vee (polarity_{pt'}(v) = + \wedge op_{pt'}(v) = 1) \vee (polarity_{pt'}(v) = - \wedge op_{pt'}(v) = 0)$*.*

pt' *is a* proper core *of* pt *if* $pt' \neq pt$*.*

Definition 9 (UC of a Parse Tree). *Let* pt*,* pt' *be parse trees.* pt' *is an* unsatisfiable core *of* pt *if 1.* $f(pt)$ *is unsatisfiable, 2.* pt' *is a core of* pt*, and 3.* $f(pt')$ *is unsatisfiable.* pt' *is an* irreducible unsatisfiable core *(IUC) of* pt *if there does not exist a proper UC of* pt'*.*

Formulas in and not in NNF Let ϕ be an unsatisfiable LTL formula with parse tree pt_ϕ in which every two subsequent occurrences of Boolean negation have been eliminated. Assume for the remainder of this section that negations are not represented as separate nodes in the parse tree of pt_ϕ but rather as an additional Boolean marking on each node. In that setting conversion of ϕ to NNF results in a formula whose parse tree is isomorphic to pt_ϕ up to labeling of the nodes with operators and negations.

Let $D_{nnf(\phi)}$ denote the set of nodes in the parse tree of ϕ that are dualized in the conversion from ϕ to $nnf(\phi)$. It is not hard to see that there exists a reverse operation nnf^{-1} that takes $nnf(\phi)$ and the set of dualized nodes $D_{nnf(\phi)}$ and returns the original formula ϕ.

Now it turns out that the following lead to the same result: 1. Compute an UC ${pt_\phi}^{uc}$ of pt_ϕ by replacing some subformulas (nodes) V'_{pt_ϕ} of pt_ϕ with 1 or 0 depending on each node's polarity. 2. (a) Convert pt_ϕ into NNF yielding $pt_{nnf(\phi)}$ with set of dualized nodes $D_{nnf(\phi)}$. (b) Replace the subformulas (nodes) isomorphic to V'_{pt_ϕ} in $pt_{nnf(\phi)}$ with fresh nodes with operator 1, yielding ${pt_{nnf(\phi)}}^{uc}$. (c) Apply nnf^{-1} to ${pt_{nnf(\phi)}}^{uc}$ with set of dualized nodes $D_{nnf(\phi)}$. (d) Replace each fresh node labeled 1 that had its negation flag set in previous step with a fresh node 0. For sample implementations of nnf and nnf^{-1} see App. A.

In other words, the set of UCs that can be obtained directly from pt_ϕ is the same as the one that can be obtained by converting ϕ to NNF, computing the set of UCs for ϕ in NNF, and undoing the conversion to NNF for each of the resulting cores.

5 Unsatisfiable Cores via Definitional Conjunctive Normal Form

Structure preserving translations (e.g., [PG86, Boy92, ER00]) 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 makes no attempt to simplify conjuncts in order to use some restricted set of operators as is done, e.g., in [Fis91]. The subsequent result on equisatisfiability is standard.

Definition 10 (Definitional Conjunctive Normal Form). *Let ϕ be an LTL formula over atomic propositions AP, let $x, x', \ldots \in X$ be fresh atomic propositions 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* $\in dCNF_{aux}(\phi)$
b *with* $b \in \mathbb{B}$	$x_\psi \leftrightarrow b$
p *with* $p \in AP$	$x_\psi \leftrightarrow p$
$\circ_1 \psi'$ *with* $\circ_1 \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\}$	$x_\psi \leftrightarrow \circ_1 x_{\psi'}$
$\psi' \circ_2 \psi''$ *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.

Fact 1 (Equisatisfiability of ϕ and $dCNF(\phi)$) *Let ϕ be an LTL formula. Then ϕ and $dCNF(\phi)$ are equisatisfiable such that 1. satisfying assignments agree on AP and 2. $x_\psi \in X$ is 1 at some time point i of a satisfying assignment π to $dCNF(\phi)$ iff the subformula ψ holds in $\pi[i, \infty]$.*

Note that as we only consider formulas given as parse trees, i.e., without sharing of subformulas, the dCNF of ϕ according to Def. 10 contains exactly one definition and one use for each occurrence of a non-root subformula.

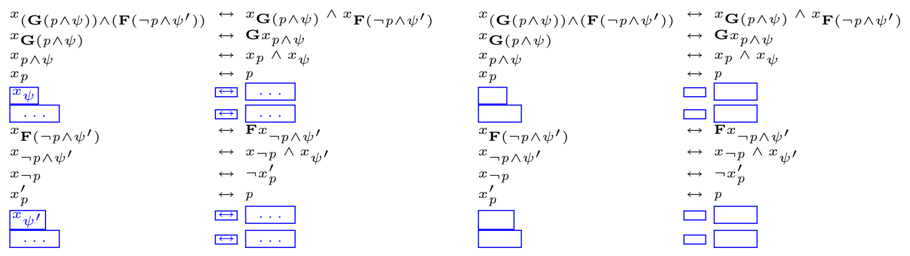
By letting the operations to derive a core from an input be the removal of elements of $dCNF_{aux}(\phi)$ we obtain the notions of core, unsatisfiable core, proper unsatisfiable core, and irreducible unsatisfiable core *via dCNF*. We additionally require that all conjuncts are discarded that contain definitions for which no (more) conjunct with a corresponding use exists. Clearly that does not impact equisatisfiability with the original formula and makes comparison with cores via parse trees (where entire subformulas are removed) easier.

The formal definitions now can be stated as follows:

Definition 11 (Core of a dCNF). *Let ϕ be an LTL formula with dCNF $dCNF(\phi)$. Let $dCNF' \equiv x' \wedge \mathbf{G} \bigwedge_{c' \in dCNF'_{aux}} c'$ be such that 1. $x' = x_\phi$, 2. $dCNF'_{aux} \subseteq dCNF_{aux}(\phi)$, and 3. for each $x \neq x_\phi$ if a definition of x is contained in $dCNF'_{aux}$, then a use of x is contained in $dCNF'_{aux}$. $dCNF'$ is a* proper core *if $dCNF'_{aux} \subset dCNF_{aux}(\phi)$.*

Definition 12 (UC of a dCNF). *Let $dCNF'$ be a core of $dCNF$. $dCNF'$ is an* unsatisfiable core *of $dCNF$ if both $dCNF$ and $dCNF'$ are unsatisfiable. $dCNF'$ is an* irreducible unsatisfiable core *of $dCNF$ if there does not exist a proper UC of $dCNF'$.*

Example 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.


$x_{(\mathbf{G}(p\wedge\psi))\wedge(\mathbf{F}(\neg p\wedge\psi'))} \leftrightarrow x_{\mathbf{G}(p\wedge\psi)} \wedge x_{\mathbf{F}(\neg p\wedge\psi')}$
$x_{\mathbf{G}(p\wedge\psi)} \leftrightarrow \mathbf{G}x_{p\wedge\psi}$
$x_{p\wedge\psi} \leftrightarrow x_p \wedge x_\psi$
$x_p \leftrightarrow p$
x_ψ
$\leftrightarrow$
...
$x_{\mathbf{F}(\neg p\wedge\psi')} \leftrightarrow \mathbf{F}x_{\neg p\wedge\psi'}$
$x_{\neg p\wedge\psi'} \leftrightarrow x_{\neg p} \wedge x_{\psi'}$
$x_{\neg p} \leftrightarrow \neg x'_p$
$x'_p \leftrightarrow p$
$x_{\psi'}$

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

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

Translating Back to LTL In Tab. 1 we indicate how to translate an IUC obtained by Def. 12 back to an LTL formula.[2] The first column states the subformula ψ, the second column indicates the polarity of the occurrence of ψ in ϕ, the third column lists the conjuncts found in the IUC (where $x_{\psi'} \leftrightarrow$ without a right-hand side stands for the definition of ψ'), and the last column shows the formula to replace ψ in the IUC as an LTL formula. The cases where none of the conjuncts is part of the IUC are omitted. All other cases cannot occur in an IUC.

To see correctness of replacing the set of conjuncts in the third column with the formulas in the fourth column it is sufficient to replace propositions used

[2] Here the translation essentially performs simplification — a translation without simplification could easily be obtained by replacing atomic propositions used but not defined with 0 or 1 depending on polarity. However, this will not be possible without loss of information for the variants of dCNF we will investigate later.

but not defined in the IUC with 1 for positive polarity occurrences and with 0 otherwise.

The argument that a certain case cannot occur in an IUC is via contradiction. Consider the example of a negative polarity occurrence of $\psi = \psi' \mathbf{R} \psi''$. Assume $x_\psi \leftrightarrow x_{\psi'} \mathbf{R} x_{\psi''}$, $x_{\psi'} \leftrightarrow$ are present in an IUC while $x_{\psi''} \leftrightarrow$ is not. Hence, removing $x_\psi \leftrightarrow x_{\psi'} \mathbf{R} x_{\psi''}$ (and, consequentially, $x_{\psi'} \leftrightarrow$) leads to a satisfiable dCNF. A satisfying assignment for that dCNF can be modified to obtain a satisfying assignment for the dCNF including $x_\psi \leftrightarrow x_{\psi'} \mathbf{R} x_{\psi''}$, $x_{\psi'} \leftrightarrow$ by setting $x_{\psi''}$ (which is unconstrained) and x_ψ to 0 at all time points. This contradicts the assumption of the latter dCNF being unsatisfiable.

Correspondence Between Cores via Parse Trees and via dCNF Let ϕ be an LTL formula. From Def. 10 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 depending on polarity, 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; specifically, the only detail that is added when converting cores between parse trees and dCNFs is turning Boolean constants that originate from replacing subformulas in a parse tree into fresh propositions from X in a dCNF and vice versa. Hence, the notions of UC obtained by Def. 9 and by Def. 12 are equivalent.

5.2 Variants

We now examine some variants of Def. 10 w.r.t. the information contained in the UCs that they can yield. Each variant is built on top of the previous one. Definitions 11 and 12 apply correspondingly.

Replacing Biimplications with Implications

Intuition and Example Definition 10 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. In other words, using an implication and a reverse implication rather than a biimplication has no benefit in terms of granularity of the obtained cores.

[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.

ψ	P	Conjuncts in IUC of ϕ via dCNF	Replacement for ψ in ϕ
$b \in \mathbb{B}$	$+/-$	$x_\psi \leftrightarrow b$	b
$p \in AP$	$+/-$	$x_\psi \leftrightarrow p$	p
$\circ_1 \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\}$	$+/-$	$x_\psi \leftrightarrow \circ_1 x_{\psi'}$ $x_{\psi'} \leftrightarrow$	$\circ_1 \psi'$
$\psi' \vee \psi''$	$+$	$x_\psi \leftrightarrow x_{\psi'} \vee x_{\psi''}$ $x_{\psi'} \leftrightarrow$ $x_{\psi''} \leftrightarrow$	$\psi' \vee \psi''$
$\psi' \vee \psi''$	$-$	$x_\psi \leftrightarrow x_{\psi'} \vee x_{\psi''}$ $x_{\psi'} \leftrightarrow$	ψ'
		$x_\psi \leftrightarrow x_{\psi'} \vee x_{\psi''}$ $x_{\psi''} \leftrightarrow$	ψ''
		$x_\psi \leftrightarrow x_{\psi'} \vee x_{\psi''}$ $x_{\psi'} \leftrightarrow$ $x_{\psi''} \leftrightarrow$	$\psi' \vee \psi''$
$\psi' \wedge \psi''$	$+$	$x_\psi \leftrightarrow x_{\psi'} \wedge x_{\psi''}$ $x_{\psi'} \leftrightarrow$	ψ'
		$x_\psi \leftrightarrow x_{\psi'} \wedge x_{\psi''}$ $x_{\psi''} \leftrightarrow$	ψ''
		$x_\psi \leftrightarrow x_{\psi'} \wedge x_{\psi''}$ $x_{\psi'} \leftrightarrow$ $x_{\psi''} \leftrightarrow$	$\psi' \wedge \psi''$
$\psi' \wedge \psi''$	$-$	$x_\psi \leftrightarrow x_{\psi'} \wedge x_{\psi''}$ $x_{\psi'} \leftrightarrow$ $x_{\psi''} \leftrightarrow$	$\psi' \wedge \psi''$
$\psi' \mathbf{U} \psi''$	$+$	$x_\psi \leftrightarrow x_{\psi'} \mathbf{U} x_{\psi''}$ $x_{\psi''} \leftrightarrow$	$\mathbf{F}\psi''$
		$x_\psi \leftrightarrow x_{\psi'} \mathbf{U} x_{\psi''}$ $x_{\psi'} \leftrightarrow$ $x_{\psi''} \leftrightarrow$	$\psi' \mathbf{U} \psi''$
$\psi' \mathbf{U} \psi''$	$-$	$x_\psi \leftrightarrow x_{\psi'} \mathbf{U} x_{\psi''}$ $x_{\psi''} \leftrightarrow$	ψ''
		$x_\psi \leftrightarrow x_{\psi'} \mathbf{U} x_{\psi''}$ $x_{\psi'} \leftrightarrow$ $x_{\psi''} \leftrightarrow$	$\psi' \mathbf{U} \psi''$
$\psi' \mathbf{R} \psi''$	$+$	$x_\psi \leftrightarrow x_{\psi'} \mathbf{R} x_{\psi''}$ $x_{\psi''} \leftrightarrow$	ψ''
		$x_\psi \leftrightarrow x_{\psi'} \mathbf{R} x_{\psi''}$ $x_{\psi'} \leftrightarrow$ $x_{\psi''} \leftrightarrow$	$\psi' \mathbf{R} \psi''$
$\psi' \mathbf{R} \psi''$	$-$	$x_\psi \leftrightarrow x_{\psi'} \mathbf{R} x_{\psi''}$ $x_{\psi''} \leftrightarrow$	$\mathbf{G}\psi''$
		$x_\psi \leftrightarrow x_{\psi'} \mathbf{R} x_{\psi''}$ $x_{\psi'} \leftrightarrow$ $x_{\psi''} \leftrightarrow$	$\psi' \mathbf{R} \psi''$

Table 1. Translating an IUC based on Def. 12 back to an LTL formula.

Formalization The formal definition is given below.

Definition 13 (Definitional Conjunctive Normal Form with Implications). *$dCNFimpl(\phi)$ is defined as $dCNF(\phi)$ except that the biimplication $\leftrightarrow$ is replaced with $\rightleftharpoons$, which is defined as $\rightarrow$ if the occurrence of ψ is positive in ϕ and with $\leftarrow$ otherwise:*

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

The translation back into an LTL formula can be achieved via Tab. 1 by replacing biimplications with (reverse) implications.

Splitting Implications for Binary Operators

Intuition and Example 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 (reverse) implications. 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. 10 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. 10, the subformula occurrence ψ'' can be seen to be irrelevant to that core. The case for $\vee$ is similar.

Formalization

Definition 14 (Definitional Conjunctive Normal Form with Split Implications). *$dCNFsplitimpl(\phi)$ is defined as $dCNFimpl(\phi)$ except in the following cases:*

ψ	*Polarity of* ψ *in* ϕ	*Conjuncts* $\in dCNF_{aux}(\phi)$
$\psi' \wedge \psi''$	$+$	$x_\psi \rightarrow x_{\psi'}, x_\psi \rightarrow x_{\psi''}$
$\psi' \vee \psi''$	$-$	$x_\psi \leftarrow x_{\psi'}, x_\psi \leftarrow x_{\psi''}$

The translation back into an LTL formula is given in Tab. 2. Only cases different from Tab. 1 (modulo (reverse) implications vs. biimplications) are listed.

Temporal Unfolding

Intuition and Example 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 $x_{\psi' \mathbf{U} \psi''} \rightarrow \mathbf{F} x_{\psi''}$.

ψ	P	Conjuncts in IUC of ϕ via dCNF	Replacement for ψ in ϕ
$\psi' \vee \psi''$	$-$	$x_\psi \leftarrow x_{\psi'}$ $x_{\psi'} \leftarrow$	ψ'
		$x_\psi \leftarrow x_{\psi''}$ $x_{\psi''} \leftarrow$	ψ''
		$x_\psi \leftarrow x_{\psi'}$ $x_\psi \leftarrow x_{\psi''}$ $x_{\psi'} \leftarrow$ $x_{\psi''} \leftarrow$	$\psi' \vee \psi''$
$\psi' \wedge \psi''$	$+$	$x_\psi \rightarrow x_{\psi'}$ $x_{\psi'} \rightarrow$	ψ'
		$x_\psi \rightarrow x_{\psi''}$ $x_{\psi''} \rightarrow$	ψ''
		$x_\psi \rightarrow x_{\psi'}$ $x_\psi \rightarrow x_{\psi''}$ $x_{\psi'} \rightarrow$ $x_{\psi''} \rightarrow$	$\psi' \wedge \psi''$

Table 2. Translating an IUC based on Def. 14 back to an LTL formula.

This can be seen to provide improved information for positive polarity occurrences of $\mathbf{U}$ subformulas in an IUC compared to Def. 14 as follows. A dCNF for a positive occurrence of an $\mathbf{U}$ subformula $\psi'\mathbf{U}\psi''$ obtained by Def. 14 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, $c''' \in C'''$, $c'''' \in C''''$. On the other hand, a dCNF with temporal unfolding as suggested 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 (the IUC contains c', $c''' \in C'''$, $c'''' \in C''''$) and a situation where either some finite unfolding of that formula or meeting its eventuality are possible but not both (the IUC contains c', c'', $c''' \in C'''$, $c'''' \in C''''$). See also Tab. 1 and Tab. 3.

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 based on Def. 14 will contain c, $c''' \in C'''$, and $c'''' \in C''''$ in both cases while one based on Def. 15 below will contain c', $c''' \in C'''$, and $c'''' \in C''''$ in the first case and additionally c'' in the second case.

Temporal unfolding leading to more fine-granular IUCs can also be applied to negative polarity occurrences of $\mathbf{R}$ formulas in a similar fashion. Here a corresponding example is 1. $(\neg(\psi'\mathbf{R}\psi'')) \wedge (\psi' \wedge \psi'')$ versus 2. $(\neg(\psi'\mathbf{R}\psi'')) \wedge ((\psi' \wedge \psi'') \vee \mathbf{G}\psi'')$. In the formal definition below we also include the opposite polarity occurrences for $\mathbf{U}$ and $\mathbf{R}$ as well as negative polarity occurrences of $\mathbf{F}$

and positive polarity occurrences of $\mathbf{G}$ subformulas.[4] However, these cases do not lead to more fine-granular IUCs.

Formalization

Definition 15 (Definitional Conjunctive Normal Form with Temporal Unfolding). *dCNFtempunf(ϕ) is defined as dCNFsplitimpl(ϕ) except in the following cases:*

ψ	*Polarity of ψ in ϕ*	*Conjuncts $\in$ dCNF$_{aux}(\phi)$*
$\psi'\mathbf{U}\psi''$	$+$	$x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee (x_{\psi'} \wedge \mathbf{X}x_{\psi'\mathbf{U}\psi''}), x_{\psi'\mathbf{U}\psi''} \rightarrow \mathbf{F}x_{\psi''}$
$\psi'\mathbf{U}\psi''$	$-$	$x_{\psi'\mathbf{U}\psi''} \leftarrow x_{\psi''} \vee (x_{\psi'} \wedge \mathbf{X}x_{\psi'\mathbf{U}\psi''})$
$\psi'\mathbf{R}\psi''$	$+$	$x_{\psi'\mathbf{R}\psi''} \rightarrow x_{\psi''} \wedge (x_{\psi'} \vee \mathbf{X}x_{\psi'\mathbf{R}\psi''})$
$\psi'\mathbf{R}\psi''$	$-$	$x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge (x_{\psi'} \vee \mathbf{X}x_{\psi'\mathbf{R}\psi''}), x_{\psi'\mathbf{R}\psi''} \leftarrow \mathbf{G}x_{\psi''}$
$\mathbf{F}\psi'$	$-$	$x_{\mathbf{F}\psi'} \leftarrow x_{\psi'} \vee \mathbf{X}x_{\mathbf{F}\psi'}$
$\mathbf{G}\psi'$	$+$	$x_{\mathbf{G}\psi'} \rightarrow x_{\psi'} \wedge \mathbf{X}x_{\mathbf{G}\psi'}$

The translation back into an LTL formula is given in Tab. 3. Only cases different from Tab. 2 are listed. In order to handle some of the additional cases provided by temporal unfolding one can either introduce a weak $\mathbf{U}$ and a strong $\mathbf{R}$ operator, which do not ($\mathbf{U}$) or do ($\mathbf{R}$) enforce the eventuality, or rewrite the additional case.

Splitting Conjunctions from Temporal Unfolding

Intuition and Example Our final variant splits the conjunctions that arise from temporal unfolding in Def. 15. 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 $\mathbf{U}$, $\mathbf{R}$, $\mathbf{F}$, or $\mathbf{G}$ formula is supposed (not) to hold in versus in the first and/or some later step.[5] Examples where this distinction comes into play are:

$\mathbf{U}$ (+ pol.):	$(\psi'\mathbf{U}\psi'') \wedge (\neg\psi' \wedge \neg\psi'')$	and	$(\psi'\mathbf{U}\psi'') \wedge (\neg\psi'' \wedge \mathbf{X}(\neg\psi' \wedge \neg\psi''))$
$\mathbf{R}$ (− pol.):	$(\neg(\psi'\mathbf{R}\psi'')) \wedge (\psi' \wedge \psi'')$	and	$(\neg(\psi'\mathbf{R}\psi'')) \wedge (\psi'' \wedge \mathbf{X}(\psi' \wedge \psi''))$
$\mathbf{F}$ (− pol.):	$(\neg\mathbf{F}\psi') \wedge \psi'$	and	$(\neg\mathbf{F}\psi') \wedge \mathbf{X}\psi'$
$\mathbf{G}$ (+ pol.):	$(\mathbf{G}\psi') \wedge \neg\psi'$	and	$(\mathbf{G}\psi') \wedge \mathbf{X}\neg\psi'$

Formalization The formal definition is as follows:

Definition 16 (Definitional Conjunctive Normal Form with Split Temporal Unfolding). *dCNFsplittempunf(ϕ) is defined as dCNFtempunf(ϕ) except in the following cases:*

[4] Unfolding the opposite polarities for $\mathbf{F}$ and $\mathbf{G}$ subformulas would require the original conjunct as without unfolding to ensure the correct fixed point and, therefore, does not make sense.

[5] Note that for the remaining 2 cases of negative polarity occurrences of $\mathbf{U}$ formulas and positive polarity occurrences of $\mathbf{R}$ formulas that level of granularity is already provided by Def. 10: either a definition of ψ' is present in an IUC or not. See also Tab. 1.

ψ	P	Conjuncts in IUC of ϕ via dCNF	Replacement for ψ in ϕ
$\psi'\mathbf{U}\psi''$	$+$	$x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee (x_{\psi'} \wedge \mathbf{X}x_{\psi'\mathbf{U}\psi''})$ $x_{\psi'} \rightarrow$ $x_{\psi''} \rightarrow$	$(\psi'\mathbf{U}\psi'') \vee \mathbf{G}\psi'$ (weak until)
		$x_{\psi'\mathbf{U}\psi''} \rightarrow \mathbf{F}x_{\psi''}$ $x_{\psi''} \rightarrow$	$\mathbf{F}\psi''$
		$x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee (x_{\psi'} \wedge \mathbf{X}x_{\psi'\mathbf{U}\psi''})$ $x_{\psi'\mathbf{U}\psi''} \rightarrow \mathbf{F}x_{\psi''}$ $x_{\psi'} \rightarrow$ $x_{\psi''} \rightarrow$	$\psi'\mathbf{U}\psi''$
$\psi'\mathbf{U}\psi''$	$-$	$x_{\psi'\mathbf{U}\psi''} \leftarrow x_{\psi''} \vee (x_{\psi'} \wedge \mathbf{X}x_{\psi'\mathbf{U}\psi''})$ $x_{\psi''} \leftarrow$	ψ''
		$x_{\psi'\mathbf{U}\psi''} \leftarrow x_{\psi''} \vee (x_{\psi'} \wedge \mathbf{X}x_{\psi'\mathbf{U}\psi''})$ $x_{\psi'} \leftarrow$ $x_{\psi''} \leftarrow$	$\psi'\mathbf{U}\psi''$
$\psi'\mathbf{R}\psi''$	$+$	$x_{\psi'\mathbf{R}\psi''} \rightarrow x_{\psi''} \wedge (x_{\psi'} \vee \mathbf{X}x_{\psi'\mathbf{R}\psi''})$ $x_{\psi''} \rightarrow$	ψ''
		$x_{\psi'\mathbf{R}\psi''} \rightarrow x_{\psi''} \wedge (x_{\psi'} \vee \mathbf{X}x_{\psi'\mathbf{R}\psi''})$ $x_{\psi'} \rightarrow$ $x_{\psi''} \rightarrow$	$\psi'\mathbf{R}\psi''$
$\psi'\mathbf{R}\psi''$	$-$	$x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge (x_{\psi'} \vee \mathbf{X}x_{\psi'\mathbf{R}\psi''})$ $x_{\psi'} \leftarrow$ $x_{\psi''} \leftarrow$	$(\psi'\mathbf{R}\psi'') \wedge \mathbf{F}\psi'$ (strong releases)
		$x_{\psi'\mathbf{R}\psi''} \leftarrow \mathbf{G}x_{\psi''}$ $x_{\psi''} \leftarrow$	$\mathbf{G}\psi''$
		$x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge (x_{\psi'} \vee \mathbf{X}x_{\psi'\mathbf{R}\psi''})$ $x_{\psi'\mathbf{R}\psi''} \leftarrow \mathbf{G}x_{\psi''}$ $x_{\psi'} \leftarrow$ $x_{\psi''} \leftarrow$	$\psi'\mathbf{R}\psi''$
$\mathbf{F}\psi'$	$-$	$x_{\mathbf{F}\psi'} \leftarrow x_{\psi'} \vee \mathbf{X}x_{\mathbf{F}\psi'}$ $x_{\psi'} \leftarrow$	$\mathbf{F}\psi'$
$\mathbf{G}\psi'$	$+$	$x_{\mathbf{G}\psi'} \rightarrow x_{\psi'} \wedge \mathbf{X}x_{\mathbf{G}\psi'}$ $x_{\psi'} \rightarrow$	$\mathbf{G}\psi'$

Table 3. Translating an IUC based on Def. 15 back to an LTL formula.

ψ	*Polarity of* ψ *in* ϕ	*Conjuncts* $\in dCNF_{aux}(\phi)$
$\psi'\mathbf{U}\psi''$	$+$	$x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee x_{\psi'}, x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee \mathbf{X}x_{\psi'\mathbf{U}\psi''}, x_{\psi'\mathbf{U}\psi''} \rightarrow \mathbf{F}x_{\psi''}$
$\psi'\mathbf{U}\psi''$	$-$	$x_{\psi'\mathbf{U}\psi''} \leftarrow x_{\psi''}, x_{\psi'\mathbf{U}\psi''} \leftarrow x_{\psi'} \wedge \mathbf{X}x_{\psi'\mathbf{U}\psi''}$
$\psi'\mathbf{R}\psi''$	$+$	$x_{\psi'\mathbf{R}\psi''} \rightarrow x_{\psi''}, x_{\psi'\mathbf{R}\psi''} \rightarrow x_{\psi'} \vee \mathbf{X}x_{\psi'\mathbf{R}\psi''}$
$\psi'\mathbf{R}\psi''$	$-$	$x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge x_{\psi'}, x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge \mathbf{X}x_{\psi'\mathbf{R}\psi''}, x_{\psi'\mathbf{R}\psi''} \leftarrow \mathbf{G}x_{\psi''}$
$\mathbf{F}\psi'$	$-$	$x_{\mathbf{F}\psi'} \leftarrow x_{\psi'}, x_{\mathbf{F}\psi'} \leftarrow \mathbf{X}x_{\mathbf{F}\psi'}$
$\mathbf{G}\psi'$	$+$	$x_{\mathbf{G}\psi'} \rightarrow x_{\psi'}, x_{\mathbf{G}\psi'} \rightarrow \mathbf{X}x_{\mathbf{G}\psi'}$

As before we indicate in Tab. 4 how to translate an IUC based on Def. 16 back to an LTL formula. Only cases different from Tab. 3 are listed.

5.3 Comparison with Separated Normal Form

Separated Normal Form (SNF) [Fis91, FN92, FDP01] is a conjunctive normal form for LTL originally proposed by Fisher to develop a resolution method for LTL. The method was implemented by Hustadt and Konev [HK02, HK03]; later applications of SNF include encodings for BMC [BCCZ99] without [FSW02] and with [CRS04] past time operators.[6]

The original SNF [Fis91, FN92] 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 [FDP01, CRS04] that allow propositions (present time formulas) on the left-hand side of the implications.

While the main contribution of [FDP01] is a full completeness result for the temporal resolution method, it also contains a simpler future time variant of SNF. It handles formulas not in NNF and uses a weak **U** operator instead of **R**. [FDP01] further refines Def. 16 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.

[CRS04] uses a less constrained version of [FDP01]: 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 of Def. 16 unnecessary. It uses **R** rather than weak **U** operators. The resulting normal form differs from Def. 15 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

[6] For the notion of past time operators see Sect. 8.

ψ	P	Conjuncts in IUC of ϕ via dCNF	Replacement for ψ in ϕ
$\psi'\mathbf{U}\psi''$	$+$	$x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee x_{\psi'}$ $x_{\psi'} \rightarrow$ $x_{\psi''} \rightarrow$	$\psi' \vee \psi''$
		$x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee x_{\psi'}$ $x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee \mathbf{X}x_{\psi'\mathbf{U}\psi''}$ $x_{\psi'} \rightarrow$ $x_{\psi''} \rightarrow$	$(\psi'\mathbf{U}\psi'') \vee \mathbf{G}\psi'$ (weak until)
		$x_{\psi'\mathbf{U}\psi''} \rightarrow \mathbf{F}x_{\psi''}$ $x_{\psi''} \rightarrow$	$\mathbf{F}\psi''$
		$x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee x_{\psi'}$ $x_{\psi'\mathbf{U}\psi''} \rightarrow \mathbf{F}x_{\psi''}$ $x_{\psi'} \rightarrow$ $x_{\psi''} \rightarrow$	$(\psi' \vee \psi'') \wedge (\mathbf{F}\psi'')$
		$x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee x_{\psi'}$ $x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee \mathbf{X}x_{\psi'\mathbf{U}\psi''}$ $x_{\psi'\mathbf{U}\psi''} \rightarrow \mathbf{F}x_{\psi''}$ $x_{\psi'} \rightarrow$ $x_{\psi''} \rightarrow$	$\psi'\mathbf{U}\psi''$
$\psi'\mathbf{U}\psi''$	$-$	$x_{\psi'\mathbf{U}\psi''} \leftarrow x_{\psi''}$ $x_{\psi''} \leftarrow$	ψ''
		$x_{\psi'\mathbf{U}\psi''} \leftarrow x_{\psi''}$ $x_{\psi'\mathbf{U}\psi''} \leftarrow x_{\psi'} \wedge \mathbf{X}x_{\psi'\mathbf{U}\psi''}$ $x_{\psi'} \leftarrow$ $x_{\psi''} \leftarrow$	$\psi'\mathbf{U}\psi''$
$\psi'\mathbf{R}\psi''$	$+$	$x_{\psi'\mathbf{R}\psi''} \rightarrow x_{\psi''}$ $x_{\psi''} \rightarrow$	ψ''
		$x_{\psi'\mathbf{R}\psi''} \rightarrow x_{\psi''}$ $x_{\psi'\mathbf{R}\psi''} \rightarrow x_{\psi'} \vee \mathbf{X}x_{\psi'\mathbf{R}\psi''}$ $x_{\psi'} \rightarrow$ $x_{\psi''} \rightarrow$	$\psi'\mathbf{R}\psi''$
$\psi'\mathbf{R}\psi''$	$-$	$x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge x_{\psi'}$ $x_{\psi'} \leftarrow$ $x_{\psi''} \leftarrow$	$\psi' \wedge \psi''$
		$x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge x_{\psi'}$ $x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge \mathbf{X}x_{\psi'\mathbf{R}\psi''}$ $x_{\psi'} \leftarrow$ $x_{\psi''} \leftarrow$	$(\psi'\mathbf{R}\psi'') \wedge \mathbf{F}\psi'$ (strong releases)
		$x_{\psi'\mathbf{R}\psi''} \leftarrow \mathbf{G}x_{\psi''}$ $x_{\psi''} \leftarrow$	$\mathbf{G}\psi''$
		$x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge x_{\psi'}$ $x_{\psi'\mathbf{R}\psi''} \leftarrow \mathbf{G}x_{\psi''}$ $x_{\psi'} \leftarrow$ $x_{\psi''} \leftarrow$	$(\psi' \wedge \psi'') \vee (\mathbf{G}\psi'')$
		$x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge x_{\psi'}$ $x_{\psi'\mathbf{R}\psi''} \leftarrow x_{\psi''} \wedge \mathbf{X}x_{\psi'\mathbf{R}\psi''}$ $x_{\psi'\mathbf{R}\psi''} \leftarrow \mathbf{G}x_{\psi''}$ $x_{\psi'} \leftarrow$ $x_{\psi''} \leftarrow$	$\psi'\mathbf{R}\psi''$
$\mathbf{F}\psi'$	$-$	$x_{\mathbf{F}\psi'} \leftarrow x_{\psi'}$ $x_{\psi'} \leftarrow$	ψ'
		$x_{\mathbf{F}\psi'} \leftarrow x_{\psi'}$ $x_{\mathbf{F}\psi'} \leftarrow \mathbf{X}x_{\mathbf{F}\psi'}$ $x_{\psi''} \leftarrow$	$\mathbf{F}\psi'$
$\mathbf{G}\psi'$	$+$	$x_{\mathbf{G}\psi'} \rightarrow x_{\psi'}$ $x_{\psi'} \rightarrow$	ψ'
		$x_{\mathbf{G}\psi'} \rightarrow x_{\psi'}$ $x_{\mathbf{G}\psi'} \rightarrow \mathbf{X}x_{\mathbf{G}\psi'}$ $x_{\psi'} \rightarrow$	$\mathbf{G}\psi'$

Table 4. Translating an IUC based on Def. 16 back to an LTL formula.

of occurrence of the respective **U**, **R**, or **G** formula. 4. As in [FDP01] atomic propositions remain unchanged at their place of occurrence. The combination of 2 and 4 leads to this variant of SNF yielding less information than Def. 15 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 than Def. 15; $(\mathbf{G}p) \wedge \neg p$ is an example for that.

6 Unsatisfiable Cores via Bounded Model Checking

6.1 Intuition and Example

By encoding existence of counterexamples of bounded length into a set of CNF clauses SAT-based Bounded Model Checking (BMC) (e.g., [BCCZ99, BCC+99, BCRZ99]) reduces model checking of LTL to SAT. Utilizing performance increases in SAT solving technology (for an overview see, e.g., [KS08]) SAT-based methods have become an established standard that complement BDD-based methods in verification; a survey on SAT-based verification methods is available in [PBG05]. Details and references on BMC can be found, e.g., in [BHJ+06].

To prove correctness of properties (rather than existence of a counterexample) BMC needs to determine when to stop searching for longer and longer counterexamples. The original works (e.g., [BCCZ99]) imposed an upper bound derived from the graph structure of the model (see also [CKOS05]). A more refined method (e.g., [SSS00]) 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. Often, bounds are tightened using some form of induction [SSS00]. For a discussion of other methods to prove properties in BMC see, e.g., [BHJ+06].

By assuming a universal model BMC provides a way to determine LTL satisfiability (used, e.g., in [CRST07]) 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 Def. 16 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 **F** or **G** operator once for each $0 \leq i \leq k$. Add the time 0 instance of the root of the dCNF, x_{ϕ}^{0}, to the set of clauses. 4. Replace each occurrence of $\mathbf{X}x_\psi^i$ with x_ψ^{i+1}. 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 [HJL05] 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 a (preferably irreducible) UC of that CNF as a subset of clauses. It is easy to see that we can extract an UC of the granularity of Def. 16 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 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 we currently have no translation back to LTL for this finer 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 Def. 16 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.

6.2 Formalization

In the following definition we spell out the translation of ϕ into a CNF for a given bound k.

Definition 17. *Let ϕ be an LTL formula over atomic propositions AP, let $k \in \mathbb{N}$. For all $0 \leq i \leq k+1$ let $y^i, y'^i, \ldots \in Y$ be fresh atomic propositions not in AP and let $p^i, q^i, \ldots$ be fresh atomic propositions — neither in AP nor in Y — denoting the values of $p, q, \ldots \in AP$ for each time step. CNFsplittempunf(ϕ, k) is a set of clauses, i.e., a CNF, containing (y_ϕ^0) and one or more clauses for each occurrence of a subformula ψ in ϕ according to Tab. 5.*

It is easy to see that *CNFsplittempunf*(ϕ, k) essentially contains a subset of the conjuncts of a dCNF according to Def. 16 and enforces each of them only for the time steps from 0 to k. Hence, the equisatisfiability of *dCNFsplittempunf*(ϕ) and ϕ implies:

Fact 2 *Let ϕ be an LTL formula over atomic propositions AP. If for some $k \in \mathbb{N}$ CNFsplittempunf(ϕ, k) is unsatisfiable, then so is ϕ. The converse does not hold.*

$\boxed{x^0_\phi}$				
$\boxed{(x^0_\phi \rightarrow x^0_{\psi\wedge\psi'})}$	$(x^1_\phi \rightarrow x^1_{\psi\wedge\psi'})$	$(x^2_\phi \rightarrow x^2_{\psi\wedge\psi'})$	$(x^3_\phi \rightarrow x^3_{\psi\wedge\psi'})$	$(x^4_\phi \rightarrow x^4_{\psi\wedge\psi'})$
$\boxed{(x^0_{\psi\wedge\psi'} \rightarrow x^0_\psi)}$	$(x^1_{\psi\wedge\psi'} \rightarrow x^1_\psi)$	$(x^2_{\psi\wedge\psi'} \rightarrow x^2_\psi)$	$(x^3_{\psi\wedge\psi'} \rightarrow x^3_\psi)$	$(x^4_{\psi\wedge\psi'} \rightarrow x^4_\psi)$
$\boxed{(x^0_\psi \rightarrow x^0_p \vee x^0_{\mathbf{XX}p})}$	$(x^1_\psi \rightarrow x^1_p \vee x^1_{\mathbf{XX}p})$	$(x^2_\psi \rightarrow x^2_p \vee x^2_{\mathbf{XX}p})$	$(x^3_\psi \rightarrow x^3_p \vee x^3_{\mathbf{XX}p})$	$(x^4_\psi \rightarrow x^4_p \vee x^4_{\mathbf{XX}p})$
$\boxed{(x^0_p \rightarrow p)}$	$(x^1_p \rightarrow p)$	$(x^2_p \rightarrow p)$	$(x^3_p \rightarrow p)$	$(x^4_p \rightarrow p)$
$\boxed{(x^0_{\mathbf{XX}p} \rightarrow x^1_{\mathbf{X}p})}$	$(x^1_{\mathbf{XX}p} \rightarrow x^2_{\mathbf{X}p})$	$(x^2_{\mathbf{XX}p} \rightarrow x^3_{\mathbf{X}p})$	$(x^3_{\mathbf{XX}p} \rightarrow x^4_{\mathbf{X}p})$	$(x^4_{\mathbf{XX}p} \rightarrow x^5_{\mathbf{X}p})$
$(x^0_{\mathbf{X}p} \rightarrow x^1_{p,2})$	$\boxed{(x^1_{\mathbf{X}p} \rightarrow x^2_{p,2})}$	$(x^2_{\mathbf{X}p} \rightarrow x^3_{p,2})$	$(x^3_{\mathbf{X}p} \rightarrow x^4_{p,2})$	$(x^4_{\mathbf{X}p} \rightarrow x^5_{p,2})$
$(x^0_{p,2} \rightarrow p)$	$(x^1_{p,2} \rightarrow p)$	$\boxed{(x^2_{p,2} \rightarrow p)}$	$(x^3_{p,2} \rightarrow p)$	$(x^4_{p,2} \rightarrow p)$
$\boxed{(x^0_{\psi\wedge\psi'} \rightarrow x^0_{\psi'})}$	$(x^1_{\psi\wedge\psi'} \rightarrow x^1_{\psi'})$	$(x^2_{\psi\wedge\psi'} \rightarrow x^2_{\psi'})$	$(x^3_{\psi\wedge\psi'} \rightarrow x^3_{\psi'})$	$(x^4_{\psi\wedge\psi'} \rightarrow x^4_{\psi'})$
$(x^0_{\psi'} \rightarrow x^0_{\neg q})$	$(x^1_{\psi'} \rightarrow x^1_{\neg q})$	$\boxed{(x^2_{\psi'} \rightarrow x^2_{\neg q})}$	$(x^3_{\psi'} \rightarrow x^3_{\neg q})$	$\boxed{(x^4_{\psi'} \rightarrow x^4_{\neg q})}$
$(x^0_{\neg q} \rightarrow \neg x^0_q)$	$(x^1_{\neg q} \rightarrow \neg x^1_q)$	$\boxed{(x^2_{\neg q} \rightarrow \neg x^2_q)}$	$(x^3_{\neg q} \rightarrow \neg x^3_q)$	$\boxed{(x^4_{\neg q} \rightarrow \neg x^4_q)}$
$(x^0_q \leftarrow q)$	$(x^1_q \leftarrow q)$	$\boxed{(x^2_q \leftarrow q)}$	$(x^3_q \leftarrow q)$	$\boxed{(x^4_q \leftarrow q)}$
$\boxed{(x^0_{\psi'} \rightarrow x^1_{\psi'})}$	$\boxed{(x^1_{\psi'} \rightarrow x^2_{\psi'})}$	$\boxed{(x^2_{\psi'} \rightarrow x^3_{\psi'})}$	$\boxed{(x^3_{\psi'} \rightarrow x^4_{\psi'})}$	$(x^4_{\psi'} \rightarrow x^5_{\psi'})$
$\boxed{(x^0_\phi \rightarrow x^0_{\psi''})}$	$(x^1_\phi \rightarrow x^1_{\psi''})$	$(x^2_\phi \rightarrow x^2_{\psi''})$	$(x^3_\phi \rightarrow x^3_{\psi''})$	$(x^4_\phi \rightarrow x^4_{\psi''})$
$\boxed{(x^0_{\psi''} \rightarrow x^0_\chi)}$	$(x^1_{\psi''} \rightarrow x^1_\chi)$	$\boxed{(x^2_{\psi''} \rightarrow x^2_\chi)}$	$(x^3_{\psi''} \rightarrow x^3_\chi)$	$(x^4_{\psi''} \rightarrow x^4_\chi)$
$\boxed{(x^0_\chi \rightarrow x^0_{\neg p} \vee x^0_{\mathbf{XX}q})}$	$(x^1_\chi \rightarrow x^1_{\neg p} \vee x^1_{\mathbf{XX}q})$	$\boxed{(x^2_\chi \rightarrow x^2_{\neg p} \vee x^2_{\mathbf{XX}q})}$	$(x^3_\chi \rightarrow x^3_{\neg p} \vee x^3_{\mathbf{XX}q})$	$(x^4_\chi \rightarrow x^4_{\neg p} \vee x^4_{\mathbf{XX}q})$
$\boxed{(x^0_{\neg p} \rightarrow \neg x^0_{p,3})}$	$(x^1_{\neg p} \rightarrow \neg x^1_{p,3})$	$\boxed{(x^2_{\neg p} \rightarrow \neg x^2_{p,3})}$	$(x^3_{\neg p} \rightarrow \neg x^3_{p,3})$	$(x^4_{\neg p} \rightarrow \neg x^4_{p,3})$
$\boxed{(x^0_{p,3} \leftarrow p)}$	$(x^1_{p,3} \leftarrow p)$	$\boxed{(x^2_{p,3} \leftarrow p)}$	$(x^3_{p,3} \leftarrow p)$	$(x^4_{p,3} \leftarrow p)$
$\boxed{(x^0_{\mathbf{XX}q} \rightarrow x^1_{\mathbf{X}q})}$	$(x^1_{\mathbf{XX}q} \rightarrow x^2_{\mathbf{X}q})$	$\boxed{(x^2_{\mathbf{XX}q} \rightarrow x^3_{\mathbf{X}q})}$	$(x^3_{\mathbf{XX}q} \rightarrow x^4_{\mathbf{X}q})$	$(x^4_{\mathbf{XX}q} \rightarrow x^5_{\mathbf{X}q})$
$(x^0_{\mathbf{X}q} \rightarrow x^1_{q,2})$	$\boxed{(x^1_{\mathbf{X}q} \rightarrow x^2_{q,2})}$	$(x^2_{\mathbf{X}q} \rightarrow x^3_{q,2})$	$\boxed{(x^3_{\mathbf{X}q} \rightarrow x^4_{q,2})}$	$(x^4_{\mathbf{X}q} \rightarrow x^5_{q,2})$
$(x^0_{q,2} \rightarrow q)$	$(x^1_{q,2} \rightarrow q)$	$\boxed{(x^2_{q,2} \rightarrow q)}$	$(x^3_{q,2} \rightarrow q)$	$\boxed{(x^4_{q,2} \rightarrow q)}$
$\boxed{(x^0_{\psi''} \rightarrow x^1_{\psi''})}$	$\boxed{(x^1_{\psi''} \rightarrow x^2_{\psi''})}$	$(x^2_{\psi''} \rightarrow x^3_{\psi''})$	$(x^3_{\psi''} \rightarrow x^4_{\psi''})$	$(x^4_{\psi''} \rightarrow x^5_{\psi''})$
time step 0	time step 1	time step 2	time step 3	time step 4

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 $\chi = p \rightarrow \mathbf{XX}q$.

ψ	Polarity of ψ in ϕ	Clauses for each $0 \leq i \leq k \in \mathit{CNFsplittempunf}(\phi, k)$
$b \in \mathbb{B}$	$+$	$(\neg y^i_\psi \vee b)$
$b \in \mathbb{B}$	$-$	$(y^i_\psi \vee \neg b)$
$p \in AP$	$+$	$(\neg y^i_\psi \vee p^i)$
$p \in AP$	$-$	$(y^i_\psi \vee \neg p^i)$
$\neg\psi'$	$+$	$(\neg y^i_\psi \vee \neg y^i_{\psi'})$
$\neg\psi'$	$-$	$(y^i_\psi \vee y^i_{\psi'})$
$\psi' \wedge \psi''$	$+$	$(\neg y^i_\psi \vee y^i_{\psi'}), (\neg y^i_\psi \vee y^i_{\psi''})$
$\psi' \wedge \psi''$	$-$	$(y^i_\psi \vee \neg y^i_{\psi'} \vee \neg y^i_{\psi''})$
$\psi' \vee \psi''$	$+$	$(\neg y^i_\psi \vee y^i_{\psi'} \vee y^i_{\psi''})$
$\psi' \vee \psi''$	$-$	$(y^i_\psi \vee \neg y^i_{\psi'}), (y^i_\psi \vee \neg y^i_{\psi''})$
$\mathbf{X}\psi'$	$+$	$(\neg y^i_\psi \vee y^{i+1}_{\psi'})$
$\mathbf{X}\psi'$	$-$	$(y^i_\psi \vee \neg y^{i+1}_{\psi'})$
$\psi'\mathbf{U}\psi''$	$+$	$(\neg y^i_{\psi'\mathbf{U}\psi''} \vee y^i_{\psi''} \vee y^i_{\psi'}), (\neg y^i_{\psi'\mathbf{U}\psi''} \vee y^i_{\psi''} \vee y^{i+1}_{\psi'\mathbf{U}\psi''})$
$\psi'\mathbf{U}\psi''$	$-$	$(y^i_{\psi'\mathbf{U}\psi''} \vee \neg y^i_{\psi''}), (y^i_{\psi'\mathbf{U}\psi''} \vee \neg y^i_{\psi'} \vee \neg y^{i+1}_{\psi'\mathbf{U}\psi''})$
$\psi'\mathbf{R}\psi''$	$+$	$(\neg y^i_{\psi'\mathbf{R}\psi''} \vee y^i_{\psi''}), (\neg y^i_{\psi'\mathbf{R}\psi''} \vee y^i_{\psi'} \vee y^{i+1}_{\psi'\mathbf{R}\psi''})$
$\psi'\mathbf{R}\psi''$	$-$	$(y^i_{\psi'\mathbf{R}\psi''} \vee \neg y^i_{\psi''} \vee \neg y^i_{\psi'}), (y^i_{\psi'\mathbf{R}\psi''} \vee \neg y^i_{\psi''} \vee \neg y^{i+1}_{\psi'\mathbf{R}\psi''})$
$\mathbf{F}\psi'$	$+$	$\emptyset$
$\mathbf{F}\psi'$	$-$	$(y^i_{\mathbf{F}\psi'} \vee \neg y^i_{\psi'}), (y^i_{\mathbf{F}\psi'} \vee \neg y^{i+1}_{\mathbf{F}\psi'})$
$\mathbf{G}\psi'$	$+$	$(\neg y^i_{\mathbf{G}\psi'} \vee y^i_{\psi'}), (\neg y^i_{\mathbf{G}\psi'} \vee y^{i+1}_{\mathbf{G}\psi'})$
$\mathbf{G}\psi'$	$-$	$\emptyset$

Table 5. Clauses in *CNFsplittempunf* for formula ϕ and bound k. The $\emptyset$ indicates that no clause is generated.

Let CNF' be an IUC of $\mathit{CNFsplittempunf}(\phi, k)$. To translate that back to an LTL formula proceed as follows. Let c^i denote the instantiation of some conjunct $c \in \mathit{dCNFsplittempunf}(\phi)$ for time step i. 1. Construct a dCNF UC based on Def. 16 as follows by setting dCNF'_{aux} such that it contains c iff c^i is part of the CNF IUC for some $0 \leq i \leq k$: $\forall c \in \mathit{dCNFsplittempunf}(\phi)$. $((\exists 0 \leq i \leq k \mathrel{.} c^i \in \mathit{CNF}') \Leftrightarrow c \in \mathit{dCNF}'_{aux})$ 2. Translate the resulting dCNF UC to LTL as described in Sect. 5. If for some subformula the corresponding set of conjuncts cannot be found in Tab. 4, then extend the set of conjuncts in the UC as needed.

Note that a CNF IUC does not guarantee a dCNF IUC. As an example consider $(\mathbf{G}(p \rightarrow (q \wedge r))) \wedge ((\neg q \wedge \neg r) \wedge (\mathbf{X}(\neg q \wedge \neg r))) \wedge (p \vee \mathbf{X}p)$. At the CNF level an UC can use, e.g., q in time step 0 and r in time step 1 and still be irreducible. Clearly such CNF IUC does not yield a dCNF IUC.

7 Unsatisfiable Cores via Tableaux

7.1 Intuition and Example

Tableaux are widely used for temporal logics. Most common methods in BDD-based symbolic model checking (e.g., [BCM+92, CGH97]) and in explicit state model checking (e.g., [GPVW95]) 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 [GPVW95]. We differ from, e.g., [GPVW95] in that we retain and continue to expand closed (i.e., contradictory) 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 the Boolean constant 0, 3. *terminal* if it contains no obligations left for the next time step, and 4. *accepting* (for some **U** or **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 **U** and **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. Satisfied paths yield satisfying assignments for the LTL formula for which the tableau is constructed.

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 **U** or **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 but fails to do so or runs into another contradiction. 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. A step-by-step description is given in the next subsection.

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. Choosing p and $\neg p$ also in n_1 leads to $\mathbf{X}(((\mathbf{G}(p \wedge 1 \wedge 1)) \wedge (\mathbf{F}(p \wedge \neg 1))) \vee (p \wedge 1 \wedge \neg p \wedge 1))$ and selecting n_5 instead of n_3 leads to two more possibilities with $\mathbf{X}p$ and $\mathbf{X}\neg p$ rather than p and $\neg p$ being preserved in the second disjunct.

The latter two possibilities show that it is not sufficient to stop the tableau construction once a closed node has been reached when it is desired that all IUCs of a formula can be extracted from an unsatisfied tableau.

Below we show that the set of UCs that can be extracted in that way is equivalent to the set of UCs obtained by Def. 9. 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.

7.2 Formalization

Below we first give our formal definition of a tableau for LTL and then explain differences w.r.t. the standard construction. The exposition is closer to constructions that are not geared towards on-the-fly expansion, e.g., [LP85].

Definition 18 (Tableau). *Let ϕ be an LTL formula in NNF with parse tree pt_ϕ. A* tableau *for ϕ is a directed graph $t_\phi = (W_{t_\phi}, F_{t_\phi})$ whose nodes W_{t_ϕ} represent sets of formulas expected to hold at a certain time point and whose edges F_{t_ϕ} represent transitions from one time point to the next.*

The closure *CL_ϕ of ϕ is the smallest set that contains all nodes in the parse tree of ϕ, V_{pt_ϕ}, and, for any node $v \in V_{pt_\phi}$ whose operator is $\mathbf{U}$, $\mathbf{R}$, $\mathbf{F}$, or $\mathbf{G}$, also contains a fresh node $v' \notin V_{pt_\phi}$ s.t. $op(v') = \mathbf{X}$ and $left(v') = v$. Given a node v representing a $\mathbf{U}$, $\mathbf{R}$, $\mathbf{F}$, or $\mathbf{G}$ formula, we denote the corresponding fresh node with $\mathbf{X}v$.*

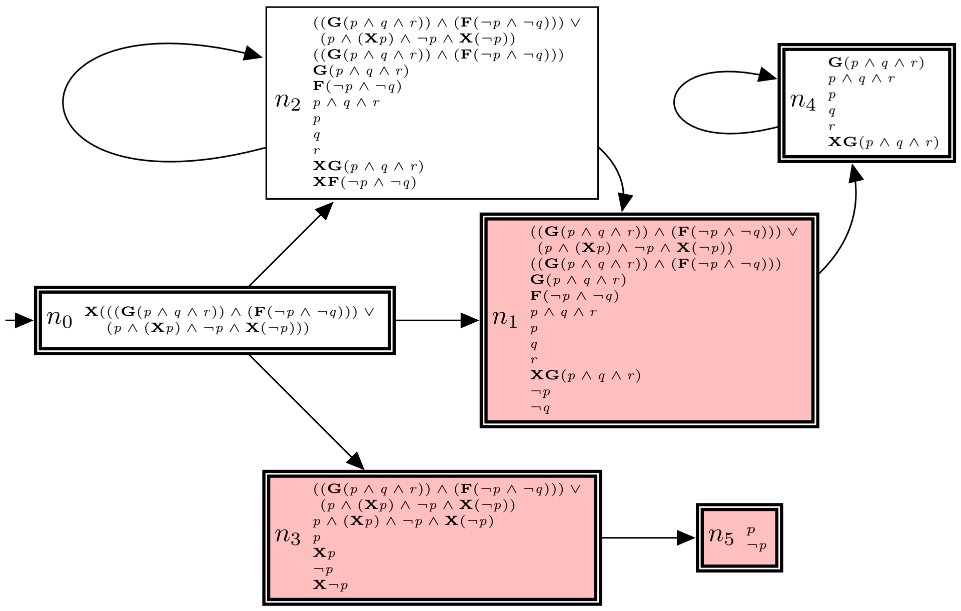

n_0
$\mathbf{X}(((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q))) \vee (p \wedge (\mathbf{X}p) \wedge \neg p \wedge \mathbf{X}(\neg p)))$
n_2
$((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q))) \vee (p \wedge (\mathbf{X}p) \wedge \neg p \wedge \mathbf{X}(\neg p))$
$((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q)))$
$\mathbf{G}(p \wedge q \wedge r)$
$\mathbf{F}(\neg p \wedge \neg q)$
$p \wedge q \wedge r$
p
q
r
$\mathbf{XG}(p \wedge q \wedge r)$
$\mathbf{XF}(\neg p \wedge \neg q)$
n_1
$((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q))) \vee (p \wedge (\mathbf{X}p) \wedge \neg p \wedge \mathbf{X}(\neg p))$
$((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q)))$
$\mathbf{G}(p \wedge q \wedge r)$
$\mathbf{F}(\neg p \wedge \neg q)$
$p \wedge q \wedge r$
p
q
r
$\mathbf{XG}(p \wedge q \wedge r)$
$\neg p$
$\neg q$
n_4
$\mathbf{G}(p \wedge q \wedge r)$
$p \wedge q \wedge r$
p
q
r
$\mathbf{XG}(p \wedge q \wedge r)$
n_3
$((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q))) \vee (p \wedge (\mathbf{X}p) \wedge \neg p \wedge \mathbf{X}(\neg p))$
$p \wedge (\mathbf{X}p) \wedge \neg p \wedge \mathbf{X}(\neg p)$
p
$\mathbf{X}p$
$\neg p$
$\mathbf{X}\neg p$
n_5
p
$\neg p$

Fig. 4. Example of an unsatisfied tableau along the lines of [GPVW95] but with closed nodes still expanded further. The formula is $\phi = \mathbf{X}(((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q))) \vee (p \wedge (\mathbf{X}p) \wedge \neg p \wedge \mathbf{X}(\neg p)))$. 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) are drawn with thick double lines, and the terminal node n_5 has no outgoing arrow.

Nodes in W_{t_ϕ} are subsets of CL_ϕ; W_{t_ϕ} contains all nodes w s.t. $\forall v \in CL_\phi$ 1. if v represents a disjunction, then w contains v iff it contains one of its children: $op(v) = \vee \Rightarrow (v \in w \Leftrightarrow left(v) \in w \vee right(v) \in w)$, 2. if v represents a conjunction, then w contains v iff it contains both children: $op(v) = \wedge \Rightarrow (v \in w \Leftrightarrow left(v), right(v) \in w)$, 3. if v represents an $\mathbf{U}$ formula, then w contains v iff it contains either the right-hand (eventuality) child or the left-hand child and the node representing the obligation for $f(v)$ to hold in the next step: $op(v) = \mathbf{U} \Rightarrow (v \in w \Leftrightarrow right(v) \in w \vee (left(v) \in w \wedge \mathbf{X}v \in w))$, 4. if v represents a $\mathbf{R}$ formula, then w contains v iff it contains both left-hand and right-hand children or the right-hand child and the obligation for $f(v)$ to hold in the next step: $op(v) = \mathbf{R} \Rightarrow (v \in w \Leftrightarrow right(v) \in w \wedge (left(v) \in w \vee \mathbf{X}v \in w))$, 5. if v represents a $\mathbf{F}$ formula, then w contains v iff it contains its left-hand (eventuality) child or the obligation for $f(v)$ to hold in the next step: $op(v) = \mathbf{F} \Rightarrow (v \in w \Leftrightarrow left(v) \in w \vee \mathbf{X}v \in w)$, and 6. if v represents a $\mathbf{G}$ formula, then w contains v iff it contains the left-hand (body) child of v and the obligation for $f(v)$ to hold in the next step: $op(v) = \mathbf{G} \Rightarrow (v \in w \Leftrightarrow left(v), \mathbf{X}v \in w)$.

A node w is 1. initial *iff it contains the root node $root(pt_\phi)$: $root(pt_\phi) \in w$, 2.* closed *iff it contains node(s) representing* 0 *or a pair of contradicting literals: $(\exists v \in w \,.\, f(v) = 0)$ or $(\exists v, v' \in w \,.\, \exists p \in AP \,.\, f(v) = p \wedge f(v') = \neg p)$, and 3.* terminal *iff it contains no obligations for the next time step: $\forall v \in w \,.\, (op(v) \neq \mathbf{X}) \wedge (v \neq \mathbf{X}v')$*

There is an edge (w, w') in t_ϕ iff, for each node $v \in w$ with either $op(v) = \mathbf{X}$ or $v = \mathbf{X}v'$ in the source node w, v is contained in the target node w': $(w, w') \in F_{t_\phi} \Leftrightarrow \forall v \in w \,.\, ((op(v) = \mathbf{X} \vee v = \mathbf{X}v') \Rightarrow v \in w')$.

A path π is initialized *iff it starts in an initial node: $root(pt_\phi) \in \pi[0]$. An infinite path π in t_ϕ is* fair *iff each eventuality that appears on some node on π is eventually fulfilled: $\forall i \in \mathbb{N} \,.\, \forall v \in \pi[i] \,.\, ((op(v) = \mathbf{U} \wedge right(v) = v') \vee (op(v) = \mathbf{F} \wedge left(v) = v')) \Rightarrow (\exists i' \geq i \,.\, v' \in \pi[i'])$.*

A path in t_ϕ is satisfied *iff 1. it is initialized, 2. it does not contain a closed node, and 3. (a) it is finite and ends in a terminal node or (b) it is infinite and fair.*

A tableau is satisfied *iff it contains a satisfied path,* unsatisfied *otherwise.*

The definition above deviates from standard definitions for LTL tableaux in that nodes are sets of parse tree nodes (occurrences of subformulas) rather than subformulas. Moreover, it does not require nodes to not contain contradictions but rather delays this check to the detection of satisfied paths. This affects neither arguments of correctness (non-existence versus non-consideration of nodes) nor of termination (finiteness of the number of nodes). Hence:

Fact 3 (ϕ is Satisfiable iff t_ϕ is Satisfied) *Let ϕ be an LTL formula in NNF with tableau t_ϕ. ϕ is satisfiable iff t_ϕ is satisfied.*

A step-by-step description to extract an UC is given below.

Definition 19 (UC Extracted From Unsatisfied Tableau). *Let ϕ be an unsatisfiable LTL formula in NNF with parse tree pt_ϕ and tableau t_ϕ. Let C_{t_ϕ} be* *the set of closed nodes in t_ϕ. Proceed as follows: 1. Choose a subset $W \subseteq W_{t_\phi}$ of nodes in t_ϕ s.t. W contains a set of closed nodes that are sufficient to prevent satisfaction of t_ϕ; in other words, even when allowing a satisfied path to contain nodes in $W_{t_\phi} \backslash (W \cap C_{t_\phi})$ rather than in $W_{t_\phi} \backslash C_{t_\phi}$, then t_ϕ would still be unsatisfied. 2. Choose a subset $V \subseteq V_{pt_\phi}$ of parse tree nodes of pt_ϕ s.t. the intersection of each closed node in W with V contains parse tree node(s) representing the Boolean constant 0 or a pair of contradicting literals. 3. Mark the nodes in pt_ϕ that are contained in V. 4. Recursively mark the fathers of marked nodes in pt_ϕ. 5. Finally remove unmarked nodes from pt_ϕ and redirect dangling edges to fresh 1 nodes.*

The following theorem states equivalence of the sets of UCs that can be obtained by extraction from an unsatisfied tableau and via parse tree.

Theorem 4 (Equivalence of UCs Extracted From Unsatisfied Tableaux and via Parse Tree). *Let ϕ be an unsatisfiable LTL formula in NNF with parse tree pt_ϕ and tableau t_ϕ. A parse tree pt' can be obtained from t_ϕ as a result of Def. 19 iff pt' is an UC of pt_ϕ via parse tree (Def. 9).*

Proof. Lemmas 1 and 2.

Lemma 1 (Correctness of UC Extraction From Unsatisfied Tableau). *Let ϕ be an unsatisfiable LTL formula in NNF with parse tree pt_ϕ and tableau t_ϕ. Let pt' be a parse tree obtained from t_ϕ as a result of Def. 19. Then pt' is an UC of pt_ϕ via parse tree (Def. 9).*

Proof. (Sketch.) We have to show that Def. 9 holds, i.e., pt' is a core (Def. 8) and it represents an unsatisfiable formula.

It is easy to see that the procedure outlined in Def. 19 selects a non-empty set of nodes in pt_ϕ and marks all of these nodes as well as all nodes on the way between any one of them and the root. It then removes subtrees rooted at unmarked nodes (including all children, also being unmarked by construction) and replaces them with 1. With ϕ being in NNF this establishes Def. 8.

In order to show that the formula represented by pt' is unsatisfiable we consider a variant of pt' that is obtained as follows. Rather than replacing unmarked subtrees with 1 we only replace unmarked leafs with 1. Let the resulting parse tree be pt'' with associated formula ϕ''. pt_ϕ and pt'' are isomorphic up to the labeling of leaf nodes. By Def. 18, their tableaux are isomorphic s.t. two isomorphic tableau nodes are sets of isomorphic parse tree nodes.

We can now state the following. 1. A node in $t_{\phi''}$ is initial (resp. terminal) iff the isomorphic node in t_ϕ is initial (resp. terminal). 2. For any parse tree node representing a formula of the form $\psi' \mathbf{U} \psi$ or $\mathbf{F}\psi$, a tableau node w in $t_{\phi''}$ contains the parse tree node v representing ψ iff the tableau node isomorphic to w in t_ϕ contains the parse tree node isomorphic to v. 3. If a node in $t_{\phi''}$ is isomorphic to a node in $W \cap C_{t_\phi}$, then it is closed.

Now it's easy to see that $t_{\phi''}$ is unsatisfied and, hence, ϕ'' is unsatisfiable. As ϕ'' simplifies to ϕ', so is the latter.

Lemma 2 (Completeness of UC Extraction From Unsatisfied Tableau). *Let ϕ be an unsatisfiable LTL formula in NNF with parse tree pt_ϕ and tableau t_ϕ. Let pt' be an UC of pt_ϕ via parse tree (Def. 9). Then pt' can be obtained from t_ϕ as a result of Def. 19.*

Proof. (Sketch.) First assume pt' is an IUC of pt_ϕ by Def. 9. Extend pt' s.t. it is isomorphic to pt_ϕ as in the proof of Lemma. 1 and name the result pt''. By correctness of the tableau method (Fact 3) the tableau for $f(pt'')$ is unsatisfied. When applying the core extraction method in Def. 19 to the tableau for $f(pt'')$ it is both possible and sufficient to mark all leaf nodes of pt'' in the tableau for $f(pt'')$ in steps 1 and 2 of Def. 19 and, hence, obtain pt'' as an UC. By a similar argument as in the proof of Thm. 1 the same core can be extracted from t_ϕ.

Now let pt' be not irreducible, and let pt'' be an IUC of pt'. By the previous argument pt'' can be extracted as an UC from pt_ϕ. Note that the tableau construction for ϕ in Def. 18 is such that every parse tree node of pt_ϕ will appear as parse tree node in some tableau node of t_ϕ. Furthermore, the core extraction according to Def. 19 allows to mark any node v appearing in some tableau node and, hence, add any parse tree node v and the corresponding path from the root of the parse tree pt_ϕ to v to the core. Hence, pt'' can be extracted as an UC from the tableau t_ϕ.

Marking More Than 0 *and Contradicting Literals* The proof of Thm. 4 makes it clear that, when IUCs are desired, in Def. 19 it is never necessary to start marking from subformulas other than 0 and contradicting literals in closed nodes.

8 Discussion

Past Time Operators Intuitively, past time operators are symmetric to the future time operators $\{\mathbf{X}, \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{G}\}$, but (assuming that the current time point is i) speak about past time points $i' \leq i$ rather than future time points $i' \geq i$. Their exact definition is unimportant for us. It is well-known that past time operators (e.g., [Eme90]) do not add expressive power to LTL [Kam68, GPSS80] but sometimes allow for more natural [LPZ85] or succinct [LMS02] specifications. While we have no formal result we conjecture that past time operators don't pose major difficulties for any of the notions of UC we suggest. We have therefore omitted past time operators from the presentation. Note, though, that we have not investigated the role of past time operators and, in particular, the influence of separation on the notion of an UC obtained via SNF.

Sharing of Subformulas Taking sharing of subformulas into account can considerably improve the presentation of an UC to a user. An example is $\phi \equiv \psi \wedge \neg\psi$: ϕ is unsatisfiable regardless of the actual definition of ψ and, most likely, the user would prefer a presentation of an UC at that abstract level rather than with all details of ψ present. As is known from vacuity checking [AFF$^+$03] neither considering occurrences of a subformula as shared nor considering occurrences of a subformula separately will always lead to the (likely) best feedback to the user: let ψ

be satisfiable, let ψ' be implied by ψ, and consider $(\psi \wedge 0) \vee (\psi \wedge \neg\psi) \vee (\psi \wedge \neg\psi')$. The first occurrence of ψ is not relevant at all, the second occurrence is relevant at this abstract level only, and the third occurrence is relevant at some more detailed level. This might suggest to strive for the most fine-grained partition of some subset of the set of occurrences of a shared subformula that still allows to establish unsatisfiability, but we have to leave further investigation of this issue to future work.

Relevant Algorithms In this paper we were mostly concerned with notions of UCs rather than with efficient ways to obtain them. Here we provide a few selected pointers to relevant algorithms. We assume that the goal is to obtain a small, possibly irreducible, UC.

Straightforward but naive algorithms work directly on the representations as parse trees or sets of conjunctions, trying to remove subtrees (resp. subsets of conjuncts), keeping them removed if the result is still unsatisfiable, and restoring the previous state otherwise. On a parse tree a possibility is to work top-down in a breadth-first fashion. On a set of conjuncts one can start with a full set of conjuncts and remove conjuncts one by one (e.g., [CD91]). Another possibility is to proceed by partitioning the set of conjuncts into two and halving the size of the partitions in each subsequent iteration (e.g., [Jun01]). If the expected size of a core is small compared to the number of clauses, then it might be more efficient to start from an empty set of conjuncts and add conjuncts until the result becomes unsatisfiable (e.g., [BS01]).

Less naive algorithms extract an UC from a proof of unsatisfiability (e.g., [GN03,ZM03a,ZM03b]). When using a technique that yields an unsatisfiable but maybe not irreducible core (such as extracting a core from a proof), then one can iterate this (or another such) technique until a fixed point or a resource limit is reached (e.g., [ZM03b]). Clearly, reuse of intermediate results in subsequent iterations may aid efficient performance (e.g., [DHN06]). When the iterative method is terminated one can continue with one of the naive techniques (e.g., [ZM03b,DHN06]). Prefixing conjuncts with activation or clause selector variables can be used to obtain several cores with a single satisfiability check (e.g., [LS05, CRST07]) or to search for IUCs with a minimum number of conjuncts (e.g., [LS04]).

9 Related Work

Notions of Core [CRST07] 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., [KS08]), i.e., by 1. treating the input formula as a Boolean combination of temporal formulas, 2. abstracting the temporal formulas with fresh Boolean propositions, 3. obtaining satisfying assignments in the Boolean space, 4. concretizing the Boolean satisfying assignments, and 5. checking satisfiability of the concretized assignments in the temporal space.

As a consequence, an UC in [CRST07] 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., [BS01]). Similarly, in constraint programming, an UC is a subset of the set of input constraints (e.g., [BDTW93]); recently, a more fine-grained notion based on unsatisfiable tuples has been suggested [GMP07]. Finally, also in satisfiability modulo theories (SMT) UCs are subsets of formulas (e.g., [CGS07]).

For realizability [PR89, ALW89] of a set of LTL formulas, partitioned into a set of assumptions and a set of guarantees, [CRST08a] suggests to first reduce the number of guarantees and then, additionally, to reduce the set of assumptions.

Extracting Cores from Proofs In [PPZ01] a successful run of a model checker, which essentially corresponds to an unsatisfied tableau, is used to extract a temporal proof from the tableau [GPVW95] as a certificate that the model fulfills the specification. [Nam01] generates certificates for successful model checking runs of μ-calculus specifications. [SC03] 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., [GN03, ZM03a, ZM03b]). In SMT UCs from SAT can be used to extract UCs for SMT [CGS07]. Extraction from proofs is also used in vacuity checking [Nam04, SDGC07].

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., [CD91, BDTW93, BS01, ZM03b].

[SSJ$^+$03] presents a method for debugging declarative specifications by translating an abstract syntax tree (AST) of an inconsistent specification to CNF, extracting an UC from the CNF, and mapping the result back to AST highlighting only the relevant parts. That work has some similarities with our discussion; however, there are also a number of differences. 1. The exposition in [SSJ$^+$03] is for first order relational logic and generalizes to languages that are reducible to SAT, while our logic is LTL. 2. The motivation and focus of [SSJ$^+$03] is on the method of core extraction, and it is accompanied by some experimental results. The notion of a core as parts of the AST is taken as a given. On the other hand, our focus is on investigating different notions of cores and on comparing the resulting information that can be gained. 3. Finally, [SSJ$^+$03] does not consider tableaux.

[TCJ08] suggests improved algorithms for core extraction compared to [SSJ$^+$03]; the improved algorithms produce IUCs at a reasonable cost by using mechanisms similar to [ZM03b, DHN06]. 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 [SC03, WHR$^+$05]. For temporal logic, the methodology proposed in

[PSC$^+$06] suggests to return a subset of the specification in case of a problem. For [CRST08a] see above.

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

Vacuity Checking Vacuity checking [BBDER01, KV03, AFF$^+$03, BFG$^+$05, GC04b, GC04a, PS02] is a technique in model checking to determine whether a model satisfies the specification in an undesired way, e.g., by never sending a request when the specification is a request response property [BB94]. 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 [BBDER01, KV03] replaces occurrences of subformulas (i.e., as we do, it does not consider sharing) 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. [CS07] 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. [FKSFV08] 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. [GC04b, CS07] discuss mutual vacuity, i.e., vacuity w.r.t. (possibly maximal) sets of subformulas. [CGS08] 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., [BBDER01,KV03,FKSFV08] w.r.t. both passing and failing formulas.

[FKSFV08] 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 [BBDER01], 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' by Def. 9 of some unsatisfiable formula ϕ, 1. ϕ is inherently vacuous, and 2. ϕ' is an *IUC* iff it is not inherently vacuous. [FKSFV08] focuses on satisfiable/realizable instances and doesn't make a connection to the notion of unsatisfiable or unrealizable cores.

[SDGC07] 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 [SDGC07] 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., [AFF$^+$03] 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.

10 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, their practical implementations could yield quite different UCs due to the way 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-irst and Verimag for helpful discussions and comments, in particular, 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

AFF$^+$03. R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. Enhanced vacuity detection in linear temporal logic. In W. Hunt Jr. and F. Somenzi, editors, *CAV*, volume 2725 of *LNCS*, pages 368–380. Springer, 2003. Links: ee, Google Scholar.

ALW89. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *ICALP*, volume 372 of *LNCS*, pages 1–17. Springer, 1989. Links: Google Scholar.

BB94. D. Beatty and R. Bryant. Formally verifying a microprocessor using a simulation methodology. In *DAC*, pages 596–602, 1994. Links: ee, Google Scholar.

BBDER01. 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):141–163, 2001. Links: Google Scholar.

BCC$^+$99. A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *DAC*, pages 317–320, 1999. Links: ee, Google Scholar.

BCCZ99. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In R. Cleaveland, editor, *TACAS*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999. Links: Google Scholar.

BCM$^+$92. J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992. Links: Google Scholar.

BCP$^+$07. R. Bloem, R. Cavada, I. Pill, M. Roveri, and A. Tchaltsev. RAT: A tool for the formal analysis of requirements. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *LNCS*, pages 263–267. Springer, 2007. Links: ee, Google Scholar.

BCRZ99. A. Biere, E. Clarke, R. Raimi, and Y. Zhu. Verifiying safety properties of a Power PC microprocessor using symbolic model checking without BDDs. In N. Halbwachs and D. Peled, editors, *CAV*, volume 1633 of *LNCS*, pages 60–71. Springer, 1999. Links: ee, Google Scholar.

BDTW93. R Bakker, F. Dikker, F. Tempelman, and P Wognum. Diagnosing and solving over-determined constraint satisfaction problems. In *IJCAI*, pages 276–281, 1993. Links: Google Scholar.

BFG$^+$05. D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M. Vardi. Regular vacuity. In D. Borrione and W. Paul, editors, *CHARME*, volume 3725 of *LNCS*, pages 191–206. Springer, 2005. Links: ee, Google Scholar.

BHJ$^+$06. 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. Links: ee, Google Scholar.

BK08. C. Baier and J. Katoen. *Principles of Model Checking*. MIT Press, 2008. Links: Google Scholar.

Boy92. T. Boy de la Tour. An optimality result for clause form translation. *J. Symb. Comput.*, 14(4):283–302, 1992. Links: Google Scholar.

BS01. R. Bruni and A. Sassano. Restoring satisfiability or maintaining unsatisfiability by finding small unsatisfiable subformulae. In H. Kautz and B. Selman, editors, *SAT*, volume 9 of *Electronic Notes in Discrete Mathematics*, pages 162–173. Elsevier, 2001. Links: ee, Google Scholar.

CD91. J. Chinneck and E. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2):157–168, 1991. Links: Google Scholar.

CE81. E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981. Links: Google Scholar.

CGH97. E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design*, 10(1):47–71, 1997. Links: Google Scholar.

CGP99. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999. Links: Google Scholar.

CGS07. A. Cimatti, A. Griggio, and R. Sebastiani. A simple and flexible way of computing small unsatisfiable cores in SAT modulo theories. In J. Marques-Silva and K. Sakallah, editors, *SAT*, volume 4501 of *LNCS*, pages 334–339. Springer, 2007. Links: ee, Google Scholar.

CGS08. H. Chockler, A. Gurfinkel, and O. Strichman. Beyond vacuity: Towards the strongest passing formula. In A. Cimatti and R. Jones, editors, *FMCAD*, pages 188–195. IEEE, 2008. Links: ee, Google Scholar.

CKOS05. E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Computational challenges in bounded model checking. *STTT*, 7(2):174–183, 2005. Links: ee, Google Scholar.

CKV06. H. Chockler, O. Kupferman, and M. Vardi. Coverage metrics for temporal logic model checking. *Formal Methods in System Design*, 28(3):189–212, 2006. Links: ee, Google Scholar.

CRS04. A. Cimatti, M. Roveri, and D. Sheridan. Bounded verification of past LTL. In A. Hu and A. Martin, editors, *FMCAD*, volume 3312 of *LNCS*, pages 245–259. Springer, 2004. Links: ee, Google Scholar.

CRST07. A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. Boolean abstraction for temporal logic satisfiability. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *LNCS*, pages 532–546. Springer, 2007. Links: ee, Google Scholar.

CRST08a. A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic information for realizability. In F. Logozzo, D. Peled, and L. Zuck, editors, *VMCAI*, volume 4905 of *LNCS*, pages 52–67. Springer, 2008. Links: ee, Google Scholar.

CRST08b. A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. From informal requirements to property-driven formal validation. In *FMICS*, 2008. To appear. Links: Google Scholar.

CRST08c. A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Object models with temporal constraints. In *SEFM*, pages 249–158. IEEE Computer Society, 2008. Links: ee, Google Scholar.

CS07. H. Chockler and O. Strichman. Easier and more informative vacuity checks. In *MEMOCODE*, pages 189–198. IEEE, 2007. Links: ee, Google Scholar.

CTVW03. E. Clarke, M. Talupur, H. Veith, and D. Wang. SAT based predicate abstraction for hardware verification. In Giunchiglia E and A. Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 78–92. Springer, 2003. Links: ee, Google Scholar.

DHN06. N. Dershowitz, Z. Hanna, and A. Nadel. A scalable algorithm for minimal unsatisfiable core extraction. In A. Biere and C. Gomes, editors, *SAT*, volume 4121 of *LNCS*, pages 36–41. Springer, 2006. Links: ee, Google Scholar.

Eme90. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 995–1072. Elsevier and MIT Press, 1990. Links: Google Scholar.

ER00. U. Egly and T. Rath. Practically useful variants of definitional translations to normal form. *Inf. Comput.*, 162(1-2):255–264, 2000. Links: Google Scholar.

eur. Formal verification of ETCS specifications. `http://es.fbk.eu/events/formal-etcs/`.

FDP01. M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Trans. Comput. Log.*, 2(1):12–56, 2001. Links: ee, Google Scholar.

Fis91. M. Fisher. A resolution method for temporal logic. In *IJCAI*, pages 99–104, 1991. Links: Google Scholar.

FKSFV08. D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M. Vardi. A framework for inherent vacuity. In *HVC*, 2008. To appear. Links: Google Scholar.

FN92. M. Fisher and P. Noël. Transformation and synthesis in METATEM. Part I: Propositional METATEM. Technical Report UMCS-92-2-1, University of Manchester, Department of Computer Science, 1992. Available from `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.4998`.

FSW02. A. Frisch, D. Sheridan, and T. Walsh. A fixpoint based encoding for bounded model checking. In M. Aagaard and J. O'Leary, editors, *FMCAD*, volume 2517 of *LNCS*, pages 238–255. Springer, 2002. Links: ee, Google Scholar.

GC04a. A. Gurfinkel and M. Chechik. Extending extended vacuity. In A. Hu and A. Martin, editors, *FMCAD*, volume 3312 of *LNCS*, pages 306–321. Springer, 2004. Links: ee, Google Scholar.

GC04b. A. Gurfinkel and M. Chechik. How vacuous is vacuous? In K. Jensen and A. Podelski, editors, *TACAS*, volume 2988 of *LNCS*, pages 451–466. Springer, 2004. Links: ee, Google Scholar.

GMP07. É. Grégoire, B. Mazure, and C. Piette. MUST: Provide a finer-grained explanation of unsatisfiability. In C. Bessiere, editor, *CP*, volume 4741 of *LNCS*, pages 317–331. Springer, 2007. Links: ee, Google Scholar.

GN03. E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *DATE*, pages 10886–10891. IEEE Computer Society, 2003. Links: ee, Google Scholar.

GPSS80. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal basis of fairness. In *POPL*, pages 163–173. ACM Press, 1980. Links: Google Scholar.

GPVW95. R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In P. Dembinski and M. Sredniawa, editors, *PSTV*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1995. Links: Google Scholar.

HJL05. K. Heljanko, T. Junttila, and T. Latvala. Incremental and complete bounded model checking for full PLTL. In K. Etessami and S. Rajamani, editors, *CAV*, volume 3576 of *LNCS*, pages 98–111. Springer, 2005. Links: ee, Google Scholar.

HK02. U. Hustadt and B. Konev. TRP++: A temporal resolution prover. In R. Nieuwenhuis, editor, *WIL*, 2002. Available from `http://www.lsi.upc.es/~roberto/wil/1.ps.gz`.

HK03. U. Hustadt and B. Konev. Trp++2.0: A temporal resolution prover. In F. Baader, editor, *CADE*, volume 2741 of *LNCS*, pages 274–278. Springer, 2003. Links: ee, Google Scholar.

Jun01. U. Junker. QuickXplain: Conflict detection for arbitrary constraint propagation algorithms. In *CONS*, 2001. Available from `http://www.lirmm.fr/~bessiere/ws_ijcai01/junker.ps.gz`.

Kam68. J. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California at Los Angeles, 1968. Links: Google Scholar.

KS08. D. Kroening and O. Strichman. *Decision Procedures.* Springer, 2008. Links: Google Scholar.

KV03. O. Kupferman and M. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2):224–233, 2003. Links: ee, Google Scholar.

LMS02. F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *LICS*, pages 383–392. IEEE Computer Society, 2002. Links: Google Scholar.

LP85. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL*, pages 97–107, 1985. Links: Google Scholar.

LPZ85. O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In R. Parikh, editor, *LoP*, volume 193 of *LNCS*, pages 196–218. Springer, 1985. Links: Google Scholar.

LS04. I. Lynce and J. Marques Silva. On computing minimum unsatisfiable cores. In H. Hoos and D. Mitchell, editors, *SAT*, volume 3542 of *LNCS*, pages 305–310. Springer, 2004. Links: ee, Google Scholar.

LS05. M. Liffiton and K. Sakallah. On finding all minimally unsatisfiable subformulas. In F. Bacchus and T. Walsh, editors, *SAT*, volume 3569 of *LNCS*, pages 173–186. Springer, 2005. Links: ee, Google Scholar.

Nam01. K. Namjoshi. Certifying model checkers. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *LNCS*, pages 2–13. Springer, 2001. Links: ee, Google Scholar.

Nam04. K. Namjoshi. An efficiently checkable, proof-based formulation of vacuity in model checking. In R. Alur and D. Peled, editors, *CAV*, volume 3114 of *LNCS*, pages 57–69. Springer, 2004. Links: ee, Google Scholar.

PBG05. M. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *STTT*, 7(2):156–173, 2005. Links: ee, Google Scholar.

PG86. D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304, 1986. Links: Google Scholar.

Pnu77. A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977. Links: Google Scholar.

PPZ01. D. Peled, A. Pnueli, and L. Zuck. From falsification to verification. In R. Hariharan, M. Mukund, and V. Vinay, editors, *FSTTCS*, volume 2245 of *LNCS*, pages 292–304. Springer, 2001. Links: ee, Google Scholar.

PR89. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. Links: Google Scholar.

pro. Prosyd. `http://www.prosyd.org/`.

PS02. M. Purandare and F. Somenzi. Vacuum cleaning CTL formulae. In E. Brinksma and K. Larsen, editors, *CAV*, volume 2404 of *LNCS*, pages 485–499. Springer, 2002. Links: ee, Google Scholar.

PSC$^+$06. I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In E. Sentovich, editor, *DAC*, pages 821–826. ACM, 2006. Links: ee, Google Scholar.

QS82. J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer, 1982. Links: Google Scholar.

SC03. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In G. Gottlob and T. Walsh, editors, *IJCAI*, pages 355–362. Morgan Kaufmann, 2003. Links: Google Scholar.

Sch09. V. Schuppan. Towards a notion of unsatisfiable cores for LTL. In F. Arbab and M. Sirjani, editors, *FSEN*, pages 57–72. School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Iran, 2009. Links: ee, Google Scholar.

SDGC07. J. Simmonds, J. Davies, A. Gurfinkel, and M. Chechik. Exploiting resolution proofs to speed up LTL vacuity detection for BMC. In *FMCAD*, pages 3–12. IEEE Computer Society, 2007. Links: ee, Google Scholar.

SSJ$^+$03. I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. Debugging overconstrained declarative models using unsatisfiable cores. In *ASE*, pages 94–105. IEEE Computer Society, 2003. Links: ee, Google Scholar.

SSS00. M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *FMCAD*, volume 1954 of *LNCS*, pages 108–125. Springer, 2000. Links: ee, Google Scholar.

TCJ08. E. Torlak, F. Chang, and D. Jackson. Finding minimal unsatisfiable cores of declarative specifications. In J. Cuéllar, T. Maibaum, and K. Sere, editors, *FM*, volume 5014 of *LNCS*, pages 326–341. Springer, 2008. Links: ee, Google Scholar.

WHR$^+$05. H. Wang, M. Horridge, A. Rector, N. Drummond, and J. Seidenberg. Debugging OWL-DL ontologies: A heuristic approach. In Y. Gil, E. Motta, V. Benjamins, and M. Musen, editors, *ISWC*, volume 3729 of *LNCS*, pages 745–757. Springer, 2005. Links: ee, Google Scholar.

WW99. S. Wolfman and D. Weld. The LPSAT engine & its application to resource planning. In T. Dean, editor, *IJCAI*, pages 310–317. Morgan Kaufmann, 1999. Links: Google Scholar.

ZM03a. L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *DATE*, pages 10880–10885. IEEE Computer Society, 2003. Links: ee, Google Scholar.

ZM03b. L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable Boolean formula. Presented at *SAT, 2003*. Available from `http://research.microsoft.com/users/lintaoz/papers/SAT_2003_core.pdf`.

A Conversion to and from NNF

In this section we show algorithms to convert an LTL formula to NNF and back by remembering the set of dualized nodes during the conversion.

Algorithm 1 Conversion of an LTL formula given as parse tree pt to NNF. D stores the set of dualized nodes.

```
procedure NNFREC(var $v$: parse tree node, var $D$: set parse tree node, $negate$: boolean)
    if $op_{pt}(v) == p$ then
        $isnegated_{pt}(v) \leftarrow isnegated_{pt}(v) \oplus negate$;
        if $negate$ then
            $D \leftarrow D \cup v$
        end if
    else
        if $isnegated_{pt}(v) \wedge \neg negate \vee \neg isnegated_{pt}(v) \wedge negate$ then
            $isnegated_{pt}(v) \leftarrow 0$
            $op_{pt}(v) \leftarrow dual(op_{pt}(v))$
            $D \leftarrow D \cup v$
            $negatedown \leftarrow 1$
        else
            if $isnegated_{pt}(v) \wedge negate$ then
                $isnegated_{pt}(v) \leftarrow 0$
            end if
            $negatedown \leftarrow 0$
        end if
        if $op_{pt}(v) == \circ_1 \vee op_{pt}(v) == \circ_2$ then
            NNFrec($left_{pt}(v)$, $v$, $D$, $negatedown$)
            if $op_{pt}(v) == \circ_2$ then
                NNFrec($right_{pt}(v)$, $v$, $D$, $negatedown$)
            end if
        end if
    end if
end procedure

procedure NNF(var $v$: parse tree node, var $D$: set parse tree node)
    $D \leftarrow \emptyset$
    NNFrec($v$, $D$)
end procedure
```

Claim. Let ϕ be an LTL formula with parse tree pt_ϕ. Let pt', D be the result of executing NNF($root(pt_\phi)$, D). Then $f(pt')$ is the NNF of ϕ.

Claim. Let ϕ be an LTL formula with parse tree pt_ϕ. Let pt', D be the result of executing NNF($root(pt_\phi)$, D), let pt'' be the result of executing deNNFrec($root(pt'')$, D). Then pt_ϕ and pt'' are isomorphic.

Algorithm 2 Undoing the conversion of an LTL formula given as parse tree pt to NNF. D stores the set of dualized nodes.

```
1: procedure deNNFrec(var $v$: parse tree node, var $D$: set parse tree node)
2:     if $op_{pt}(v) == p$ then
3:         if $v \in D$ then
4:             $isnegated_{pt}(v) \leftarrow \neg isnegated_{pt}(v)$
5:         end if
6:     else
7:         if $v \in D$ then
8:             $op_{pt}(v) \leftarrow dual(op_{pt}(v))$
9:         end if
10:        if $v == root(pt)$ then
11:            $isnegated_{pt}(v) \leftarrow v \in D$
12:        else
13:            $isnegated_{pt}(v) \leftarrow v \in D \oplus father_{pt}(v) \in D$
14:        end if
15:        if $op_{pt}(v) == \circ_1 \vee op_{pt}(v) == \circ_2$ then
16:            deNNFrec($left_{pt}(v)$, $v$, $D$)
17:            if $op_{pt}(v) == \circ_2$ then
18:                deNNFrec($right_{pt}(v)$, $v$, $D$)
19:            end if
20:        end if
21:    end if
22: end procedure

23: procedure deNNF(var $v$: parse tree node, var $D$: set parse tree node, var $N$: set parse tree
    node)
24:    deNNFrec($v$, $D$)
25:    for all $v' \in N$ do
26:        if $isnegated_{pt}(v')$ then
27:            $isnegated_{pt}(v') \leftarrow 0$
28:            $op_{pt}(v') \leftarrow 0$
29:        end if
30:    end for
31: end procedure
```