Towards a Notion of Unsatisfiable and Unrealizable Cores for LTL[☆]

Viktor Schuppan

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

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-grained than existing ones. We illustrate the benefits of the more fine-grained notions on examples from the literature. We extend some of the notions to realizability and we discuss the relationship of unsatisfiable and unrealizable cores with the notion of vacuity.

Keywords: unsatisfiable cores, unrealizable cores, temporal logic, LTL

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, Eme90]. Consequently, in recent years methodologies for property-based design with temporal logics have been developed (e.g., [PSC+06, 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, BK08a] 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 [CS09, FKSFV08], both vacuity and coverage assume the presence of both a model and its requirements. Particularly in the 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 available. There is tool support for both (e.g., [BCG+10, 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. Consider, e.g., the EURAILCHECK project [CCM+10, eur] which developed a methodology [CRST08b] and a tool [CCM+09] for the validation of requirements in the context of railway signaling and control. Part of the methodology consists of translating the set of (implicitly conjoined) requirements given by a textual specification into a variant [CRST08c, CRT09] of LTL whose atoms are constraints in a first order theory (including continuous real-time aspects), followed by 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.

[☆] A preliminary version of this paper appeared in [Sch09a, Sch09b].

Email address: `schuppan@fbk.eu` (Viktor Schuppan)

URL: `http://www.schuppan.de/viktor/` (Viktor Schuppan)

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 a 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 a 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, a more fine-grained UC will likely make the corresponding task easier. Similar considerations apply to the notions of unrealizable cores that have been proposed so far to help debugging unrealizable specifications [CRST08a, KHB09].

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 a 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 a 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.

We then extend some of the notions to realizability. The notion based on the syntactic structure can be applied almost directly. Transferring the notion based on conjunctive normal forms requires some technical transformations of the specification and is currently restricted to formulas of the GR(1) [PPS06] subset of LTL. Both notions partially improve upon [CRST08a, KHB09].

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 a similar systematic investigation of the notion of a UC and of an unrealizable core for LTL. The relationship with vacuity checking is discussed in depth in Sect. 10.1; for notions of cores for other specification formalisms, for application of UCs, and for other related approaches see Sect. 11.

The paper is structured as follows. In 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 syntax trees, by taking subsets of conjuncts in conjunctive normal forms, by extracting resolution proofs from BMC runs, and by extraction from closed tableaux nodes. The notions are illustrated using examples from the literature in Sect. 8. In Sect. 9 we extend some notions of a UC to unrealizable cores. In Sect. 10 we relate the notion of cores to that of vacuity and state some complexity results. Related work is discussed in Sect. 11 before we conclude in Sect. 12.

2. Preliminaries

In the following we give standard definitions for LTL [Pnu77], see, e.g., [Eme90, BK08a]. 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).

$$\begin{array}{lll}
(\pi, i) \models 1 & & \\
(\pi, i) \not\models 0 & & \\
(\pi, i) \models p & \Leftrightarrow & p \in \pi[i] \\
(\pi, i) \models \neg\psi & \Leftrightarrow & (\pi, i) \not\models \psi \\
(\pi, i) \models \psi \vee \psi' & \Leftrightarrow & (\pi, i) \models \psi \text{ or } (\pi, i) \models \psi' \\
(\pi, i) \models \psi \wedge \psi' & \Leftrightarrow & (\pi, i) \models \psi \text{ and } (\pi, i) \models \psi'
\end{array}$$

$$\begin{array}{lll}
(\pi, i) \models \psi \mathbf{U} \psi' & \Leftrightarrow & \exists i' \geq i \,.\, ((\pi, i') \models \psi' \wedge \forall i \leq i'' < i' \,.\, (\pi, i'') \models \psi) \\
(\pi, i) \models \psi \mathbf{R} \psi' & \Leftrightarrow & \forall i' \geq i \,.\, ((\pi, i') \models \psi' \vee \exists i \leq i'' < i' \,.\, (\pi, i'') \models \psi) \\
(\pi, i) \models \mathbf{X}\psi & \Leftrightarrow & (\pi, i+1) \models \psi \\
(\pi, i) \models \mathbf{F}\psi & \Leftrightarrow & \exists i' \geq i \,.\, (\pi, i') \models \psi \\
(\pi, i) \models \mathbf{G}\psi & \Leftrightarrow & \forall i' \geq i \,.\, (\pi, i') \models \psi
\end{array}$$

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 (Language). The *language* of an LTL formula ϕ, $L(\phi)$, is the set of words satisfying ϕ: $L(\phi) = \{\pi \in (2^{AP})^\omega \mid \pi \models \phi\}$.

Definition 4 (Satisfiability). An LTL formula ϕ is *satisfiable* iff its language is non-empty: $L(\phi) \neq \emptyset$; it is *unsatisfiable* otherwise.

The satisfiability problem for LTL is PSPACE-complete [SC85]; see also [Mar04, BSS+09]. For current work on practical methods for LTL satisfiability solving refer to, e.g., [RV10, CRST07, WDMR08, LH09].

Definition 5 (Negation Normal Form). An LTL formula ϕ is in *negation normal form (NNF) nnf*(ϕ) 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., [BK08a].

Definition 6 (Subformula). Let ϕ be an LTL formula. The set of subformulas $SF(\phi)$ of ϕ is defined recursively as follows:

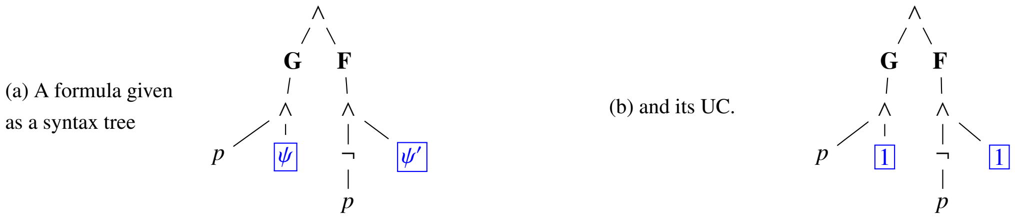
$$\begin{array}{llll}
\psi = b \text{ or } \psi = p & \text{with} & b \in \mathbb{B}, p \in AP & : \; SF(\psi) = \{\psi\} \\
\psi = \circ_1 \psi' & \text{with} & \circ_1 \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\} & : \; SF(\psi) = \{\psi\} \cup SF(\psi') \\
\psi = \psi' \circ_2 \psi'' & \text{with}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

Definition 7 (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 for LTL independently of the precise notion of a UC used. 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]).



(a) A formula given as a syntax tree

(b) and its UC.

Figure 1: Example of a UC via syntax tree. Modified parts are marked blue boxed.

UCs, Irreducible UCs, and Least-Cost Irreducible UCs

A notion of a UC will map LTL formulas to sets of LTL formulas. Here we formulate (though not formalize) some general expectations on that mapping. 1. Given that a UC ϕ' of some LTL formula ϕ should explain unsatisfiability of ϕ the notion of a core should preserve (some) reasons for the unsatisfiability of ϕ and should not add new ones. 2. Unsatisfiability of the UC ϕ' should be easier to understand than unsatisfiability of ϕ. This normally means that a UC ϕ' of ϕ is smaller than ϕ. 3. The UC ϕ' of ϕ is obtained from ϕ in such a way that it is clear that 1 holds. Such a mapping defines a notion of a *core* (note that the mapping applies to satisfiable and unsatisfiable formulas).

Given a notion of a core for LTL formulas, the following additional notions can be defined for a core ϕ' of an LTL formula ϕ. ϕ' is an *unsatisfiable* core if ϕ' is a core of ϕ and ϕ' is unsatisfiable. 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 (assuming some cost function) least-cost IUCs minimum UCs.

Granularity of a Notion of a UC

Clearly, the original LTL formula 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 formulas that make some of these reasons easier to see. Therefore we use the term *granularity* of a notion of a core as follows. We wish to determine the relevance of certain aspects of a formula to the formula 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 a core provides finer granularity than another notion if it provides at least as much information on the relevance of certain aspects of a formula as the other notion.

As an example consider a notion of a 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 a 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 Syntax 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 replacing an occurrence of a subformula with positive polarity with 1 or replacing an occurrence of a subformula with negative polarity with 0 — as is done, e.g., in some forms of vacuity checking [KV03] — will lead to a weaker formula. This naturally leads to a definition of UC based on syntax 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 syntax tree is depicted in Fig. 1 (a). The formula is unsatisfiable independently of the concrete (and possibly complex) subformulas ψ, ψ'. A corresponding UC with ψ, ψ' replaced with 1 is $\phi' = (\mathbf{G}(p \wedge 1)) \wedge (\mathbf{F}(\neg p \wedge 1))$, shown in Fig. 1 (b).

Hence, by deriving a core ϕ' from some LTL formula ϕ by replacing occurrences of subformulas of ϕ with 1 (for positive polarity occurrences) or 0 (for negative polarity occurrences), we obtain the notions of a core, an unsatisfiable core, a proper unsatisfiable core, and an irreducible unsatisfiable core *via syntax 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 8 (Syntax Tree). Let ϕ be an LTL formula. The *syntax 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 $root(pt_\phi) \in V_{pt_\phi}$; and 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 9 (Core of a Syntax Tree). Let pt, pt' be syntax 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' is a core of pt and $pt' \neq pt$.

Definition 10 (UC of a Syntax Tree). Let pt, pt' be syntax 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 syntax 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 syntax 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 syntax 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 syntax 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 a 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.

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 a 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 11 (Definitional Conjunctive Normal Form). Let ϕ be an LTL formula over atomic propositions AP, let $X = \{x, x', \ldots\}$ be a set of fresh atomic propositions not in AP. $dCNF_{aux}(\phi)$ is a set of conjuncts over $AP \cup X$ 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 12 (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 syntax trees, i.e., without sharing of subformulas, the dCNF of ϕ according to Def. 11 contains exactly one definition and one use for each occurrence of a non-root subformula.

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

The formal definitions now can be stated as follows:

Definition 13 (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 $dCNF'_{aux}$. Then $dCNF'$ is a *core* of $dCNF(\phi)$. $dCNF'$ is a *proper core* if $dCNF'_{aux} \subset dCNF_{aux}(\phi)$.

Definition 14 (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 a 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.

$$\begin{array}{lcl}
x_{(\mathbf{G}(p\wedge\psi))\wedge(\mathbf{F}(\neg p\wedge\psi'))} & \leftrightarrow & x_{\mathbf{G}(p\wedge\psi)} \wedge x_{\mathbf{F}(\neg p\wedge\psi')} \\
x_{\mathbf{G}(p\wedge\psi)} & \leftrightarrow & \mathbf{G}x_{p\wedge\psi} \\
x_{p\wedge\psi} & \leftrightarrow & x_p \wedge x_\psi \\
x_p & \leftrightarrow & p \\
\boxed{x_\psi} & \boxed{\leftrightarrow} & \boxed{\ldots} \\
\boxed{\ldots} & \boxed{\leftrightarrow} & \boxed{\ldots} \\
x_{\mathbf{F}(\neg p\wedge\psi')} & \leftrightarrow & \mathbf{F}x_{\neg p\wedge\psi'} \\
x_{\neg p\wedge\psi'} & \leftrightarrow & x_{\neg p} \wedge x_{\psi'} \\
x_{\neg p} & \leftrightarrow & \neg x'_p \\
x'_p & \leftrightarrow & p \\
\boxed{x_{\psi'}} & \boxed{\leftrightarrow} & \boxed{\ldots} \\
\boxed{\ldots} & \boxed{\leftrightarrow} & \boxed{\ldots}
\end{array}$$

(a) A formula given as a dCNF

$$\begin{array}{lcl}
x_{(\mathbf{G}(p\wedge\psi))\wedge(\mathbf{F}(\neg p\wedge\psi'))} & \leftrightarrow & x_{\mathbf{G}(p\wedge\psi)} \wedge x_{\mathbf{F}(\neg p\wedge\psi')} \\
x_{\mathbf{G}(p\wedge\psi)} & \leftrightarrow & \mathbf{G}x_{p\wedge\psi} \\
x_{p\wedge\psi} & \leftrightarrow & x_p \wedge x_\psi \\
x_p & \leftrightarrow & p \\
\boxed{} & \boxed{} & \boxed{} \\
\boxed{} & \boxed{} & \boxed{} \\
x_{\mathbf{F}(\neg p\wedge\psi')} & \leftrightarrow & \mathbf{F}x_{\neg p\wedge\psi'} \\
x_{\neg p\wedge\psi'} & \leftrightarrow & x_{\neg p} \wedge x_{\psi'} \\
x_{\neg p} & \leftrightarrow & \neg x'_p \\
x'_p & \leftrightarrow & p \\
\boxed{\phantom{x_{\psi'}}} & \boxed{} & \boxed{} \\
\boxed{} & \boxed{} & \boxed{}
\end{array}$$

(b) and its UC.

Figure 2: Example of a 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. 14 back to an LTL formula.[1] 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 the 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 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 Syntax Trees and via dCNF

Let ϕ be an LTL formula. From Def. 11 it is clear that there is a one-to-one correspondence between the nodes in the syntax tree of ϕ and the conjuncts in its dCNF. Therefore, the conversion between the representation of ϕ as a syntax tree and as a dCNF is straightforward.

Remember that a UC of a syntax tree is obtained by replacing an occurrence of a subformula ψ with 1 or 0 depending on polarity, while a UC of a dCNF is obtained by removing the definition of ψ and all dependent definitions. Both ways to obtain a UC do not destroy the correspondence between syntax trees and dCNFs; specifically, the only detail that is added when converting cores between syntax trees and dCNFs is turning Boolean constants that originate from replacing subformulas in a syntax tree into fresh propositions from X in a dCNF and vice versa. Hence, the notions of a UC obtained by Def. 10 and by Def. 14 are equivalent.

5.2. *Variants*

We now examine some variants of Def. 11 w.r.t. the information contained in the UCs that they can yield. Each variant is built on top of the previous one. Definitions 13 and 14 apply correspondingly.

[1] 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.

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. 14 back to an LTL formula.

5.2.1. Replacing Biimplications with Implications

Intuition and Example. Definition 11 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.[2] However, it is easy to see that in our setting of formulas as syntax 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.

Formalization. The formal definition is given below.

Definition 15 (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 $\in dCNFimpl_{aux}(\phi)$
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.

5.2.2. 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. 11 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. 11, the subformula occurrence ψ'' can be seen to be irrelevant to that core. The case for $\vee$ is similar.

Formalization.

Definition 16 (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 $\in dCNFsplitimpl_{aux}(\phi)$
$\psi' \wedge \psi''$	$+$	$x_\psi \rightarrow x_{\psi'}, x_\psi \rightarrow x_{\psi''}$
$\psi' \vee \psi''$	$-$	$x_\psi \leftarrow x_{\psi'}, x_\psi \leftarrow x_{\psi''}$

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.

5.2.3. 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 $x_{\psi'\mathbf{U}\psi''} \rightarrow \mathbf{F}x_{\psi''}$.

[2] 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
$\psi' \vee \psi''$	$-$	$x_{\psi} \leftarrow x_{\psi'}$ $x_{\psi'} \leftarrow$	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. 16 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. 16 as follows. A dCNF for a positive occurrence of an $\mathbf{U}$ subformula $\psi'\mathbf{U}\psi''$ obtained by Def. 16 results (among others) in the following conjuncts: $c_0 = x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi'}\mathbf{U}x_{\psi''}$, $C_3 = \{x_{\psi'} \rightarrow \ldots\}$, and $C_4 = \{x_{\psi''} \rightarrow \ldots\}$. An IUC based on that dCNF contains either 1. none of c_0, $c_3 \in C_3$, $c_4 \in C_4$, 2. c_0, $c_4 \in C_4$, or 3. c_0, $c_3 \in C_3$, $c_4 \in C_4$. On the other hand, a dCNF with temporal unfolding as suggested results in the conjuncts: $c_1 = x_{\psi'\mathbf{U}\psi''} \rightarrow x_{\psi''} \vee (x_{\psi'} \wedge \mathbf{X}x_{\psi'\mathbf{U}\psi''})$, $c_2 = x_{\psi'\mathbf{U}\psi''} \rightarrow \mathbf{F}x_{\psi''}$, and C_3, C_4 as before. An IUC based on that dCNF contains either 1. none of c_1, c_2, $c_3 \in C_3$, $c_4 \in C_4$, 2. c_1, $c_3 \in C_3$, $c_4 \in C_4$, 3. c_2, $c_4 \in C_4$, or 4. c_1, c_2, $c_3 \in C_3$, $c_4 \in C_4$. 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_1, $c_3 \in C_3$, $c_4 \in C_4$) and a situation where either some finite unfolding of that formula or meeting its eventuality are possible but not both (the IUC contains $c_1, c_2, c_3 \in C_3, c_4 \in C_4$). 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. 16 will contain c_0, $c_3 \in C_3$, and $c_4 \in C_4$ in both cases, while one based on Def. 17 below will contain c_1, $c_3 \in C_3$, and $c_4 \in C_4$ in the first case and additionally c_2 in the second case.

Temporal unfolding leading to more fine-grained 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.[3] However, these cases do not lead to more fine-grained IUCs.

Formalization.

Definition 17 (Definitional Conjunctive Normal Form with Temporal Unfolding). *dCNFtempunf*(ϕ) is defined as *dCNFsplitimpl*(ϕ) except in the following cases:

[3] 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.

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

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 **U** and a strong **R** operator, which do not (**U**) or do (**R**) enforce the eventuality, or rewrite the additional case.

5.2.4. Splitting Conjunctions from Temporal Unfolding

Intuition and Example. Our final variant splits the conjunctions that arise from temporal unfolding in Def. 17. In 4 of the 6 cases where temporal unfolding is possible, this allows to distinguish the case where unsatisfiability is due to failure of unfolding in only the first time step that a **U**, **R**, **F**, or **G** formula is supposed (not) to hold in versus in the first and/or some later step.[4] Examples where this distinction comes into play are:

$$\begin{array}{lll}
\mathbf{U}\ (+\ \text{pol.}): & (\psi'\mathbf{U}\psi'') \wedge (\neg\psi' \wedge \neg\psi'') & \text{and} \quad (\psi'\mathbf{U}\psi'') \wedge (\neg\psi'' \wedge \mathbf{X}(\neg\psi' \wedge \neg\psi'')) \\
\mathbf{R}\ (-\ \text{pol.}): & (\neg(\psi'\mathbf{R}\psi'')) \wedge (\psi' \wedge \psi'') & \text{and} \quad (\neg(\psi'\mathbf{R}\psi'')) \wedge (\psi'' \wedge \mathbf{X}(\psi' \wedge \psi'')) \\
\mathbf{F}\ (-\ \text{pol.}): & (\neg\mathbf{F}\psi') \wedge \psi' & \text{and} \quad (\neg\mathbf{F}\psi') \wedge \mathbf{X}\psi' \\
\mathbf{G}\ (+\ \text{pol.}): & (\mathbf{G}\psi') \wedge \neg\psi' & \text{and} \quad (\mathbf{G}\psi') \wedge \mathbf{X}\neg\psi'
\end{array}$$

Formalization. The formal definition is as follows:

Definition 18 (Definitional Conjunctive Normal Form with Split Temporal Unfolding). *dCNFsplittempunf*(ϕ) is defined as *dCNFtempunf*(ϕ) except in the following cases:

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

As before we indicate in Tab. 4 how to translate an IUC based on Def. 18 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.

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

[4] Note that for the remaining 2 cases of negative polarity occurrences of **U** formulas and positive polarity occurrences of **R** formulas that level of granularity is already provided by Def. 11: 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. 17 back to an LTL formula.

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. 18 back to an LTL formula.

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. 18 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. 18 unnecessary. It uses **R** rather than weak **U** operators. The resulting normal form differs from Def. 17 in 4 respects: 1. It works on NNF. 2. Positive Boolean combinations are not split into several conjuncts. 3. Fresh propositions are introduced for **U**, **R**, and **G** formulas representing truth in the next rather than in the current time step. Because of that, temporal unfolding is performed at the place of occurrence of the respective **U**, **R**, or **G** formula. 4. As in [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. 17 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. 17; $(\mathbf{G}p) \wedge \neg p$ is an example for that.

6. Unsatisfiable Cores via Bounded Model Checking

6.1. Intuition and Example

By encoding the 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. 18 to generate a SAT problem in CNF as follows: 1. Pick some bound k.[5] 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 $0 \leq i \leq k$.

[5] In practice, one typically starts with $k = 0$ and increases k by 1 until either (un)satisfiability is determined or a resource limit is reached. For references to a discussion on upper bounds see above. In addition, some discussion on the effects of guessing the right/a slightly too large bound can be found in [ES03].

	x_ϕ^0		
✔	$\boxed{(x_\phi^0 \rightarrow x_{p\vee\mathbf{XX}p}^0)}$	$(x_\phi^1 \rightarrow x_{p\vee\mathbf{XX}p}^1)$	$(x_\phi^2 \rightarrow x_{p\vee\mathbf{XX}p}^2)$
✔	$\boxed{(x_{p\vee\mathbf{XX}p}^0 \rightarrow x_{p,0}^0 \vee x_{\mathbf{XX}p}^0)}$	$(x_{p\vee\mathbf{XX}p}^1 \rightarrow x_{p,0}^1 \vee x_{\mathbf{XX}p}^1)$	$(x_{p\vee\mathbf{XX}p}^2 \rightarrow x_{p,0}^2 \vee x_{\mathbf{XX}p}^2)$
✔	$\boxed{(x_{p,0}^0 \rightarrow p)}$	$(x_{p,0}^1 \rightarrow p)$	$(x_{p,0}^2 \rightarrow p)$
✔	$\boxed{(x_{\mathbf{XX}p}^0 \rightarrow x_{\mathbf{X}p}^1)}$	$(x_{\mathbf{XX}p}^1 \rightarrow x_{\mathbf{X}p}^2)$	$(x_{\mathbf{XX}p}^2 \rightarrow x_{\mathbf{X}p}^3)$
✔	$(x_{\mathbf{X}p}^0 \rightarrow x_{p,1}^1)$	$\boxed{(x_{\mathbf{X}p}^1 \rightarrow x_{p,1}^2)}$	$(x_{\mathbf{X}p}^2 \rightarrow x_{p,1}^3)$
✔	$(x_{p,1}^0 \rightarrow p)$	$(x_{p,1}^1 \rightarrow p)$	$\boxed{(x_{p,1}^2 \rightarrow p)}$
✔	$\boxed{(x_\phi^0 \rightarrow x_{\mathbf{G}(\neg p\wedge q)}^0)}$	$(x_\phi^1 \rightarrow x_{\mathbf{G}(\neg p\wedge q)}^1)$	$(x_\phi^2 \rightarrow x_{\mathbf{G}(\neg p\wedge q)}^2)$
✔	$\boxed{(x_{\mathbf{G}(\neg p\wedge q)}^0 \rightarrow x_{\mathbf{G}(\neg p\wedge q)}^1)}$	$\boxed{(x_{\mathbf{G}(\neg p\wedge q)}^1 \rightarrow x_{\mathbf{G}(\neg p\wedge q)}^2)}$	$(x_{\mathbf{G}(\neg p\wedge q)}^2 \rightarrow x_{\mathbf{G}(\neg p\wedge q)}^3)$
✔	$\boxed{(x_{\mathbf{G}(\neg p\wedge q)}^0 \rightarrow x_{\neg p\wedge q}^0)}$	$(x_{\mathbf{G}(\neg p\wedge q)}^1 \rightarrow x_{\neg p\wedge q}^1)$	$\boxed{(x_{\mathbf{G}(\neg p\wedge q)}^2 \rightarrow x_{\neg p\wedge q}^2)}$
✔	$\boxed{(x_{\neg p\wedge q}^0 \rightarrow x_{\neg p}^0)}$	$(x_{\neg p\wedge q}^1 \rightarrow x_{\neg p}^1)$	$\boxed{(x_{\neg p\wedge q}^2 \rightarrow x_{\neg p}^2)}$
✔	$\boxed{(x_{\neg p}^0 \rightarrow \neg x_{p,2}^0)}$	$(x_{\neg p}^1 \rightarrow \neg x_{p,2}^1)$	$\boxed{(x_{\neg p}^2 \rightarrow \neg x_{p,2}^2)}$
✔	$\boxed{(\neg x_{p,2}^0 \rightarrow \neg p)}$	$(\neg x_{p,2}^1 \rightarrow \neg p)$	$\boxed{(\neg x_{p,2}^2 \rightarrow \neg p)}$
	$(x_{\neg p\wedge q}^0 \rightarrow x_q^0)$	$(x_{\neg p\wedge q}^1 \rightarrow x_q^1)$	$(x_{\neg p\wedge q}^2 \rightarrow x_q^2)$
	$(x_q^0 \rightarrow q)$	$(x_q^1 \rightarrow q)$	$(x_q^2 \rightarrow q)$
dCNF core	time step 0	time step 1	time step 2

Figure 3: Example of a UC via BMC. The input formula is $\phi = (p \vee \mathbf{XX}p) \wedge \mathbf{G}(\neg p \wedge q)$. Clauses that form the SAT IUC are marked blue boxed. A tick in the leftmost column indicates that the corresponding dCNF clause is part of a UC via dCNF.

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 a UC of the granularity of Def. 18 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 the notion of a core via removal of clauses from the CNF thus giving the notions of a core, an unsatisfiable core, a proper unsatisfiable core, and an irreducible unsatisfiable core via BMC.

As an example consider $\phi = (p \vee \mathbf{XX}p) \wedge \mathbf{G}(\neg p \wedge q)$. The translation into a set of CNF clauses and the CNF IUC are depicted in Fig. 3. Extracting a UC at the granularity of Def. 18 results in a dCNF equivalent to $(p \vee \mathbf{XX}p) \wedge \mathbf{G}(\neg p \wedge 1)$. The CNF IUC shows that the occurrence of $\neg p$ is relevant only at time steps 0 and 2.

6.2. Formalization

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

Definition 19. 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 —

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

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

denoting the values of $p, q, \ldots \in AP$ for each time step. $\mathit{CNFsplittempunf}(\phi, k)$ 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 $\mathit{CNFsplittempunf}(\phi, k)$ essentially contains a subset of the conjuncts of a dCNF according to Def. 18 and enforces each of them only for the time steps from 0 to k. Hence, the equisatisfiability of $\mathit{dCNFsplittempunf}(\phi)$ and ϕ implies:

Fact 20. *Let ϕ be an LTL formula over atomic propositions AP. If for some $k \in \mathbb{N}$ $\mathit{CNFsplittempunf}(\phi, k)$ is unsatisfiable, then so is ϕ. The converse does not hold.*

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. 18 as follows by setting $dCNF'_{aux}$ such that it contains c iff c^{i} is part of the CNF IUC for some $0 \leq i \leq k$: $\forall c \in \mathit{dCNFsplittempunf}(\phi) \; . \; ((\exists 0 \leq i \leq k \; . \; c^{i} \in CNF') \Leftrightarrow c \in dCNF'_{aux})$ 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 a 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. Typically, a tableau construction for LTL establishes (un)satisfiability of some LTL formula ϕ by constructing and analyzing a graph (a tableau) for ϕ. The tableau is constructed such that ϕ is satisfiable iff the tableau contains a path fulfilling certain properties (such a path is called satisfied below). Any such path corresponds to a model of ϕ. In other words, constructing and analyzing a tableau means searching for a model. Nodes in the tableau are characterized by sets of subformulas of ϕ and formulas derived from subformulas of ϕ. The set of formulas characterizing a node should be free of contradictions at some level of abstraction (e.g., considering all formulas that are literals); contradicting nodes are called closed below and are often dropped from consideration. The set of formulas characterizing a node represents the formulas that hold on any path in the tableau starting at that node (and possibly fulfilling certain additional properties). As a consequence, there is a directed edge from one tableau node to another, if the formulas characterizing the target node imply the obligations that the formulas characterizing the source node leave for the next time step. A node of the tableau is initial if its characterizing set of formulas contains ϕ. For a path in the tableau to be a model of ϕ, the path will normally have to start in an initial node and not infinitely often visit a node that contains a **U** formula without infinitely often visiting a node that contains the right hand side of the **U** formula.

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 at 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 it is 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 is unsatisfiable.

The procedure to extract a 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 syntax tree of the formula are marked and retained, while all non-marked occurrences of subformulas in the syntax 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.

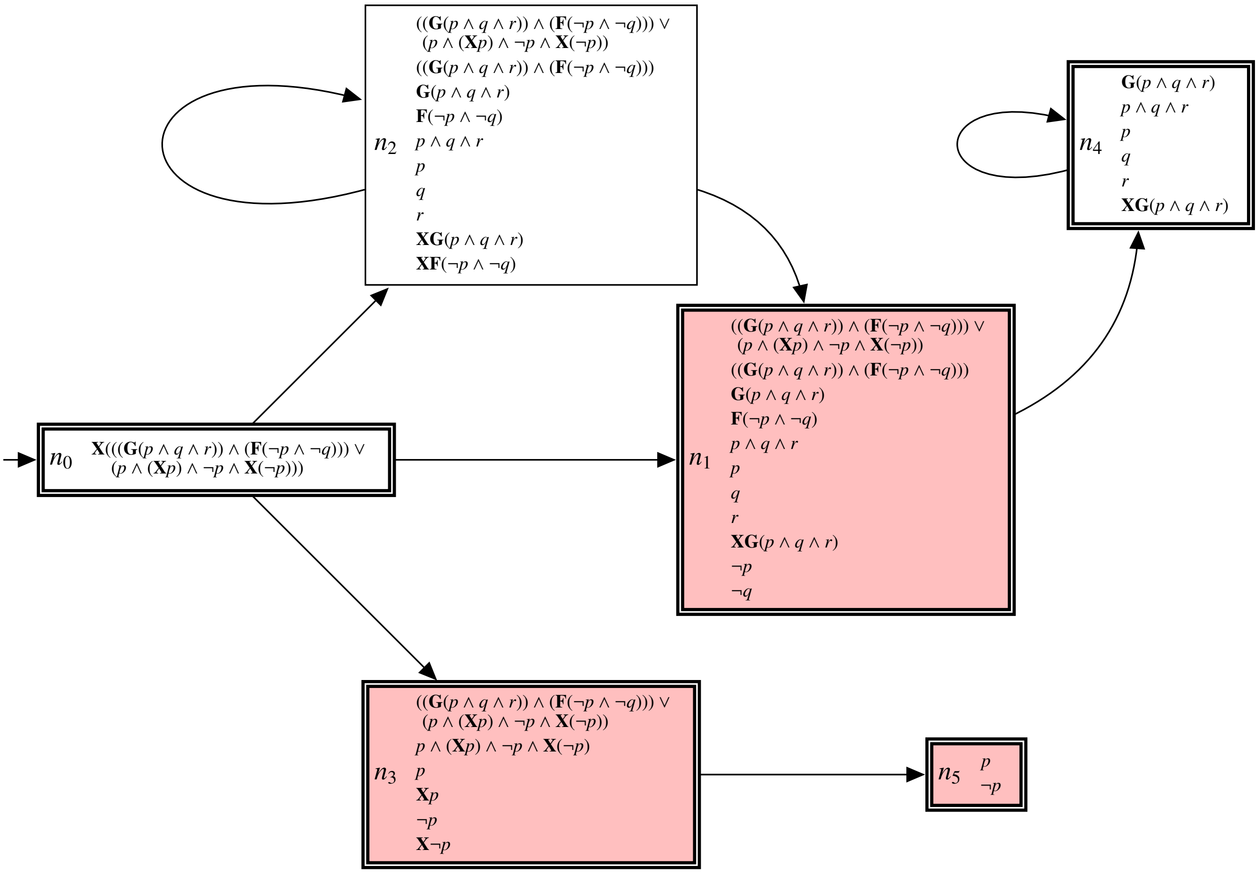

n_2
$((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q))) \vee (p \wedge (\mathbf{X}p) \wedge \neg p \wedge \mathbf{X}(\neg p))$
$((\mathbf{G}(p \wedge q \wedge r)) \wedge (\mathbf{F}(\neg p \wedge \neg q)))$
$\mathbf{G}(p \wedge q \wedge r)$
$\mathbf{F}(\neg p \wedge \neg q)$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

Figure 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.

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. 10. 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 21 (Tableau). Let ϕ be an LTL formula in NNF with syntax 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 syntax tree of ϕ, V_{pt_ϕ}, and, for any node $v \in V_{pt_\phi}$ whose operator is **U**, **R**, **F**, or **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 **U**, **R**, **F**, or **G** formula, we denote the corresponding fresh node with $\mathbf{X}v$.

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 **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 **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 **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 **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 (resp. v') is contained in the target node w': $(w, w') \in F_{t_\phi} \Leftrightarrow \forall v \in w \;.\; ((op(v) = \mathbf{X} \Rightarrow v \in w') \wedge (v = \mathbf{X}v' \Rightarrow v' \in w'))$.

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 syntax 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 22 (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 a UC is given below.

Definition 23 (UC Extracted From Unsatisfied Tableau). Let ϕ be an unsatisfiable LTL formula in NNF with syntax 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 $W_{t_\phi} \setminus (W \cap C_{t_\phi})$ rather than in $W_{t_\phi} \setminus C_{t_\phi}$, then t_ϕ would still be unsatisfied. 2. Choose a subset $V \subseteq V_{pt_\phi}$ of syntax tree nodes of pt_ϕ s.t. the intersection of each closed node in W with V contains syntax 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 syntax tree.

Theorem 24 (Equivalence of UCs Extracted From Unsatisfied Tableaux and via Syntax Tree). *Let ϕ be an unsatisfiable LTL formula in NNF with syntax tree pt_ϕ and tableau t_ϕ. A syntax tree pt' can be obtained from t_ϕ as a result of Def. 23 iff pt' is a UC of pt_ϕ via syntax tree (Def. 10).*

Proof. Lemmas 25 and 26.

Lemma 25 (Correctness of UC Extraction From Unsatisfied Tableau). *Let ϕ be an unsatisfiable LTL formula in NNF with syntax tree pt_ϕ and tableau t_ϕ. Let pt' be a syntax tree obtained from t_ϕ as a result of Def. 23. Then pt' is a UC of pt_ϕ via syntax tree (Def. 10).*

Proof. (Sketch.) We have to show that Def. 10 holds, i.e., pt' is a core (Def. 9) and it represents an unsatisfiable formula.

It is easy to see that the procedure outlined in Def. 23 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. 9.

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 syntax tree be pt'' with associated formula ϕ''. pt_ϕ and pt'' are isomorphic up to the labeling of leaf nodes. By Def. 21, their tableaux are isomorphic s.t. two isomorphic tableau nodes are sets of isomorphic syntax 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 syntax 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 syntax tree node v representing ψ iff the tableau node isomorphic to w in t_ϕ contains the syntax 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 26 (Completeness of UC Extraction From Unsatisfied Tableau). *Let ϕ be an unsatisfiable LTL formula in NNF with syntax tree pt_ϕ and tableau t_ϕ. Let pt' be a UC of pt_ϕ via syntax tree (Def. 10). Then pt' can be obtained from t_ϕ as a result of Def. 23.*

Proof. (Sketch.) First assume pt' is an IUC of pt_ϕ by Def. 10. Extend pt' s.t. it is isomorphic to pt_ϕ as in the proof of Lemma 25 and name the result pt''. By correctness of the tableau method (Fact 22) the tableau for $f(pt'')$ is unsatisfied. When applying the core extraction method in Def. 23 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. 23 and, hence, obtain pt'' as a UC. By a similar argument as in the proof of Lemma 25 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 a UC from pt_ϕ. Note that the tableau construction for ϕ in Def. 21 is such that every syntax tree node of pt_ϕ will appear as syntax tree node in some tableau node of t_ϕ. Furthermore, the core extraction according to Def. 23 allows to mark any node v appearing in some tableau node and, hence, add any syntax tree node v and the corresponding path from the root of the syntax tree pt_ϕ to v to the core. Hence, pt'' can be extracted as a UC from the tableau t_ϕ.

Marking More Than 0 *and Contradicting Literals.* The proof of Thm. 24 makes it clear that, when IUCs are desired, in Def. 23 it is never necessary to start marking from subformulas other than 0 and contradicting literals in closed nodes.

8. Examples

In this section we apply the notions suggested in the previous sections to three examples found in [WDMR08, Har05, ala] and [RV10, roz].[6] The first two examples fall in the application category. Using the traditional notion of a UC as a subset of a set of conjuncts, we show that the specification of a lift is buggy. We then use the notions introduced in the previous sections to understand impossibility of a different scenario for the corrected lift. The last example is a random formula from a set of benchmark formulas. That formula doesn't admit any simplification by the traditional notion of a UC but can be reduced in size significantly by our notions.

8.1. Tracking Down a Problem in a Lift Specification

Figure 5 (a)–(c) shows the example used in this subsection. In Fig. 5 (a) we list the example as obtained from [ala] with the following modifications. 1. We rewrote the example as a set of conjuncts. We turned biimplications into implications. 2. We removed button 0 from conjuncts 40–42. This corresponds to the original version of the specification in [Har05]. 3. We added the scenario we are interested in as conjunct 49.

The specification characterizes a lift that serves 3 floors. Presence of the lift at floor i is indicated by variable fi being 1. The lift is requested to serve floor i by pressing button bi. The lift and its users take turns in their actions: variable u is 1 if it's the users' turn, 0 if it is the turn of the lift. sb is essentially a macro that is 1 iff button 1 or button 2 are pressed. up observes whether the lift moves up (the value must be 1), down (the value must be 0), or remains at a given floor (the value doesn't change). For explanations of the conjuncts see the comments in Fig. 5 (a). For a more detailed explanation we refer the reader to Chapter 4.3.2 of [Har05].

Clearly, a reasonable lift specification will permit the lift to leave its initial position to serve other floors. This is the first scenario we would like to explore. Surprisingly, it turns out that the specification in Fig. 5 (a) is unsatisfiable, i.e., it does not permit the lift to leave floor 0; hence, it should be considered buggy.

Using the traditional notion of a UC as a subset of a set, we obtain 6 out of 49 conjuncts as an irreducible unsatisfiable subset of conjuncts (Fig. 5 (b)). The more fine-grained notions suggested in Sect.s 4–7 do not lead to further simplification. Inspection of the IUC in Fig. 5 (b) suggests that the variable up is not used consistently. In conjunct 31 up records whether the lift has moved between the previous and the current time step. In conjunct 36 it records whether the lift will move between the current and the next time step. Correspondingly modifying conjuncts 36–39 by adding a $\mathbf{X}$ operator as suggested in Fig. 5 (c) makes the specification satisfiable.

8.2. Exploring a Possibility in a Lift Specification

The second example is based on the corrected lift specification, see Fig. 6 (a)–(e). This time we explore a different scenario: we would like to know whether the lift ever needs to return to floor 0 after the first time step. We correspondingly replace conjunct 49 with 49' (expressing the fact that it doesn't). The resulting specification is unsatisfiable.

Again, the traditional notion of a UC as a subset of a set of conjuncts reduces the conjuncts to consider from 49 to 6 (Fig. 6 (b)). This time, however, further reduction is possible. Using the notion of a core via syntax trees, the most complex of the remaining conjuncts can be simplified by removing two literal occurrences and a conjunction operator, and turning an $\mathbf{U}$ into a $\mathbf{F}$ (Fig. 6 (c)). The result of rewriting the simplified specification in Fig. 6 (c) into dCNF is shown in Fig. 6 (d). Computing an IUC via dCNF allows to drop two conjuncts from the dCNF. By inspection it becomes clear that each removal allows the removal of a $\mathbf{G}$ operator in Fig. 6 (c). For the result see Fig. 6 (e).

[6] We encourage the reader to validate the examples in this section using any LTL satisfiability solver; for a selection see [RV10]. The examples are available for the NuSMV model checker at `http://www.schuppan.de/viktor/fsen09_scp_examples/`.

// the lift is only at one floor at a time

1. $\mathbf{G}(f0 \rightarrow \neg f1)$
2. $\mathbf{G}(f0 \rightarrow \neg f2)$
3. $\mathbf{G}(f1 \rightarrow \neg f2)$

// initial values

4. $\neg u$
5. $f0$
6. $\neg b0$
7. $\neg b1$
8. $\neg b2$
9. $\neg \mathit{up}$

// the lift and the users take turns

10. $\mathbf{G}(u \rightarrow \neg \mathbf{X} u)$
11. $\mathbf{G}((\neg \mathbf{X} u) \rightarrow u)$

// when it is the users' turn, then the floor remains unchanged

12. $\mathbf{G}(u \rightarrow (f0 \rightarrow \mathbf{X} f0))$
13. $\mathbf{G}(u \rightarrow ((\mathbf{X} f0) \rightarrow f0))$
14. $\mathbf{G}(u \rightarrow (f1 \rightarrow \mathbf{X} f1))$
15. $\mathbf{G}(u \rightarrow ((\mathbf{X} f1) \rightarrow f1))$
16. $\mathbf{G}(u \rightarrow (f2 \rightarrow \mathbf{X} f2))$
17. $\mathbf{G}(u \rightarrow ((\mathbf{X} f2) \rightarrow f2))$

// the lift can move at most to one neighboring floor in one
// step

18. $\mathbf{G}(f0 \rightarrow \mathbf{X}(f0 \vee f1))$
19. $\mathbf{G}(f1 \rightarrow \mathbf{X}(f0 \vee f1 \vee f2))$
20. $\mathbf{G}(f2 \rightarrow \mathbf{X}(f1 \vee f2))$

// when it is the lift's turn, then the buttons remain unchanged

21. $\mathbf{G}((\neg u) \rightarrow (b0 \rightarrow \mathbf{X} b0))$
22. $\mathbf{G}((\neg u) \rightarrow ((\mathbf{X} b0) \rightarrow b0))$
23. $\mathbf{G}((\neg u) \rightarrow (b1 \rightarrow \mathbf{X} b1))$
24. $\mathbf{G}((\neg u) \rightarrow ((\mathbf{X} b1) \rightarrow b1))$
25. $\mathbf{G}((\neg u) \rightarrow (b2 \rightarrow \mathbf{X} b2))$
26. $\mathbf{G}((\neg u) \rightarrow ((\mathbf{X} b2) \rightarrow b2))$

// the buttons remain pressed while the corresponding floor has not
// been reached

27. $\mathbf{G}((b0 \wedge \neg f0) \rightarrow \mathbf{X} b0)$
28. $\mathbf{G}((b1 \wedge \neg f1) \rightarrow \mathbf{X} b1)$
29. $\mathbf{G}((b2 \wedge \neg f2) \rightarrow \mathbf{X} b2)$

// *up* is true if the lift moves up, false if it moves down, and
// unchanged if it doesn't move

30. $\mathbf{G}((f0 \wedge (\mathbf{X} f0)) \rightarrow (\mathit{up} \rightarrow \mathbf{X} \mathit{up}))$
31. $\mathbf{G}((f0 \wedge (\mathbf{X} f0)) \rightarrow ((\mathbf{X} \mathit{up}) \rightarrow \mathit{up}))$
32. $\mathbf{G}((f1 \wedge (\mathbf{X} f1)) \rightarrow (\mathit{up} \rightarrow \mathbf{X} \mathit{up}))$
33. $\mathbf{G}((f1 \wedge (\mathbf{X} f1)) \rightarrow ((\mathbf{X} \mathit{up}) \rightarrow \mathit{up}))$
34. $\mathbf{G}((f2 \wedge (\mathbf{X} f2)) \rightarrow (\mathit{up} \rightarrow \mathbf{X} \mathit{up}))$
35. $\mathbf{G}((f2 \wedge (\mathbf{X} f2)) \rightarrow ((\mathbf{X} \mathit{up}) \rightarrow \mathit{up}))$
36. $\mathbf{G}((f0 \wedge (\mathbf{X} f1)) \rightarrow \mathit{up})$
37. $\mathbf{G}((f1 \wedge (\mathbf{X} f2)) \rightarrow \mathit{up})$
38. $\mathbf{G}((f1 \wedge (\mathbf{X} f0)) \rightarrow \neg \mathit{up})$
39. $\mathbf{G}((f2 \wedge (\mathbf{X} f1)) \rightarrow \neg \mathit{up})$

// *sb* is true iff *b*1 or *b*2 are pressed

40. $\mathbf{G}(\mathit{sb} \rightarrow (b1 \vee b2))$
41. $\mathbf{G}(b1 \rightarrow \mathit{sb})$
42. $\mathbf{G}(b2 \rightarrow \mathit{sb})$

// when it is not used, then the lift parks at floor 0

43. $\mathbf{G}((f0 \wedge \neg \mathit{sb}) \rightarrow (f0 \mathbf{U}(\mathit{sb} \mathbf{R}((\mathbf{F} f0) \wedge \neg \mathit{up}))))$
44. $\mathbf{G}((f1 \wedge \neg \mathit{sb}) \rightarrow (f1 \mathbf{U}(\mathit{sb} \mathbf{R}((\mathbf{F} f0) \wedge \neg \mathit{up}))))$
45. $\mathbf{G}((f2 \wedge \neg \mathit{sb}) \rightarrow (f2 \mathbf{U}(\mathit{sb} \mathbf{R}((\mathbf{F} f0) \wedge \neg \mathit{up}))))$

// each request will eventually be served

46. $\mathbf{G}(b0 \rightarrow \mathbf{F} f0)$
47. $\mathbf{G}(b1 \rightarrow \mathbf{F} f1)$
48. $\mathbf{G}(b2 \rightarrow \mathbf{F} f2)$

// the lift eventually leaves floor 0

49. $\mathbf{F} \neg f0$

(a) Formulas 1–48 represent a lift specification used in [WDMR08] (available from [ala]). Here we add formula 49 to express the desire to eventually leave floor 0. The specification is given here as a set of conjuncts, i.e., the overall specification is obtained by conjoining formulas 1–49. [ala] is closely modeled after the example in [Har05] instantiated for 3 floors. Our only changes w.r.t. the version from [ala] are 1. writing it as a set of formulas to be conjoined (possibly splitting biimplications), 2. following the original [Har05] in 40–42 by leaving out $b0$, and 3. adding 49. Note that changes 1. and 2. do not impact satisfiability of the conjunction of 1–49. The resulting specification turns out to be unsatisfiable (while the conjunction of 1–48 is satisfiable), hence, the lift can not leave floor 0.

5. $f0$

9. $\neg \mathit{up}$

18. $\mathbf{G}(f0 \rightarrow \mathbf{X}(f0 \vee f1))$

31. $\mathbf{G}((f0 \wedge (\mathbf{X} f0)) \rightarrow ((\mathbf{X} \mathit{up}) \rightarrow \mathit{up}))$

36. $\mathbf{G}((f0 \wedge (\mathbf{X} f1)) \rightarrow \mathit{up})$

49. $\mathbf{F} \neg f0$

(b) An IUC of the conjunction of 1–49 as an irreducible subset of the conjuncts 1–49. In other words, the conjunction of $\{5, 9, 18, 31, 36, 49\}$ is unsatisfiable and the conjunction of any proper subset of $\{5, 9, 18, 31, 36, 49\}$ is satisfiable. Applying any of the more fine-grained definitions proposed in Sect.s 4–7 does not provide additional information.

36'. $\mathbf{G}((f0 \wedge (\mathbf{X} f1)) \rightarrow \mathbf{X} \mathit{up})$

37'. $\mathbf{G}((f1 \wedge (\mathbf{X} f2)) \rightarrow \mathbf{X} \mathit{up})$

38'. $\mathbf{G}((f1 \wedge (\mathbf{X} f0)) \rightarrow \mathbf{X} \neg \mathit{up})$

39'. $\mathbf{G}((f2 \wedge (\mathbf{X} f1)) \rightarrow \mathbf{X} \neg \mathit{up})$

(c) One way to fix unsatisfiability of the specification (and the one we are following here) is by setting the the right hand sides of the implications in 36–39 to $\mathbf{X}(\neg)\mathit{up}$ (i.e., *up* observes the movement of the lift between the previous and the current time step).

Figure 5: Example of using IUCs to track down a problem. Here the notion of a UC as a subset of a set of conjuncts is sufficient.

Now it is easy to see unsatisfiability. Conjuncts 5, 7, 8, and 40' imply that sb is 0 at time step 0. Hence, the left hand side of the implication in 43" is 1 at time step 0. However, sb is 0 at time step 0 and, because of 49', $f0$ is false after time step 0. Hence, the right hand side of the implication in 43" can not be fulfilled.

8.3. A Random Formula

The third example (see Fig. 7) is a slightly simplified instance[7] of the random formulas benchmark set used in [RV10] (available from [roz]). This set of formulas was generated using the method described in [DGV99]. To obtain the formula in Fig. 7 (a) we removed double negations.

Here no (simple) rewriting into a set of conjuncts or even a formula with a sizeable Boolean component at the top is possible. Hence, the traditional notion of a UC as a subset of a set of conjuncts does not apply. By using the notion of a UC via syntax tree we obtain a formula with 35 rather than 112 nodes in its syntax tree (Fig. 7 (a)). Further simplification (using only the identities $1 \wedge \psi \equiv \psi$, $0 \vee \psi \equiv \psi$, $0\mathbf{U}\psi \equiv \psi$) results in a formula with only 17 nodes left (Fig. 7 (b)). Rewriting the formula in Fig. 7 (b) into dCNF and computing the corresponding IUC shows that $a\mathbf{U}b$ can be replaced with a weak $\mathbf{U}$ without losing unsatisfiability (Fig. 7 (c)). Finally, annotating the clauses in the dCNF with information about the time steps in which they are relevant as suggested in Sect. 6 provides further insight (Fig. 7 (c)): no clause is relevant after the second time step and only two clauses are relevant at more than one time step.

Unsatisfiability of the formula in Fig. 7 (b) can now be understood as follows. To make the formula 1 both sides of the outermost disjunction need to be 0 (conjuncts 1, 15, 16, 17 in Fig. 7 (c)). The right hand side disjunct implies a and b being 0 at time step 1 (conjuncts 12–14). Note that this makes $\neg(a\mathbf{U}b)$ 1 at time step 1 (conjuncts 6, 9). Hence, both sides of the outermost $\mathbf{U}$ formula need to be 0 at time step 0 (conjuncts 10, 11). That implies b being 1 at time step 0 (conjuncts 9, 7). On the other hand, making the left hand side of the $\mathbf{U}$ 0 requires b to be 0 at time step 0 (conjuncts 2–5).

8.4. Discussion

The first lift example shows that specifications may contain bugs and that UCs can help to track down the problem. In this example the traditional notion of a UC as a subset of a set of conjuncts turns out to be sufficient for understanding the problem and ultimately coming up with a fix. The traditional notion of a UC reduces the specification from 49 to 6 conjuncts.

In the second lift example the traditional notion again reduces the specification from 49 to 6 conjuncts. However, this time the notions via syntax trees and via dCNF yield more fine-grained information that leads to the additional removal of 2 literal occurrences, 3 operators, and simplification of another operator.

Note that for the purpose of this section we rewrote the lift specification as a set of conjuncts to make the traditional notion applicable as much as possible. This is not possible for the random example; the traditional notion provides no help in simplifying the formula. In contrast, the notion via syntax tree reduces the syntax tree of the formula from 112 to 35 nodes. Further simplification leads to a formula with a syntax tree of size 17. Applying the notion via dCNF indicates that a strong $\mathbf{U}$ can be exchanged for a weak one. Finally, along the lines of Sect. 6 we can conclude that no conjunct is relevant after the second time step and only two conjuncts are relevant at both the first and the second time steps.

The examples show that it is essential to reduce the formula size to understand why a particular formula is unsatisfiable. While there are diminishing returns of the more fine-grained notions of a UC, the more fine-grained notions did prove helpful in providing additional information useful for further simplification. We note that, while it is somewhat hard to find realistic specifications in LTL available for research (not speaking of unsatisfiable ones), none of the examples in this section was crafted by us (we did add the specific scenarios to be explored in the lift examples). Finally, we make two remarks w.r.t. the fact that the random example is the most complex one. First, we expect that specifications become more complex. Second, random examples are used frequently to debug solvers and investigating unexpected results typically accounts for a non-negligible amount of the time spent.

[7] To be precise, it is based on the sixth formula in P0.7N4L100.form.

// the lift is only at one floor at a time

1. $\mathbf{G}(f0 \rightarrow \neg f1)$
2. $\mathbf{G}(f0 \rightarrow \neg f2)$
3. $\mathbf{G}(f1 \rightarrow \neg f2)$

// initial values

4. $\neg u$
5. $f0$
6. $\neg b0$
7. $\neg b1$
8. $\neg b2$
9. $\neg up$

// the lift and the users take turns

10. $\mathbf{G}(u \rightarrow \neg \mathbf{X} u)$
11. $\mathbf{G}((\neg \mathbf{X} u) \rightarrow u)$

// when it is the users' turn, then the floor remains unchanged

12. $\mathbf{G}(u \rightarrow (f0 \rightarrow \mathbf{X} f0))$
13. $\mathbf{G}(u \rightarrow ((\mathbf{X} f0) \rightarrow f0))$
14. $\mathbf{G}(u \rightarrow (f1 \rightarrow \mathbf{X} f1))$
15. $\mathbf{G}(u \rightarrow ((\mathbf{X} f1) \rightarrow f1))$
16. $\mathbf{G}(u \rightarrow (f2 \rightarrow \mathbf{X} f2))$
17. $\mathbf{G}(u \rightarrow ((\mathbf{X} f2) \rightarrow f2))$

// the lift can move at most to one neighboring floor in one
// step

18. $\mathbf{G}(f0 \rightarrow \mathbf{X}(f0 \vee f1))$
19. $\mathbf{G}(f1 \rightarrow \mathbf{X}(f0 \vee f1 \vee f2))$
20. $\mathbf{G}(f2 \rightarrow \mathbf{X}(f1 \vee f2))$

// when it is the lift's turn, then the buttons remain unchanged

21. $\mathbf{G}((\neg u) \rightarrow (b0 \rightarrow \mathbf{X} b0))$
22. $\mathbf{G}((\neg u) \rightarrow ((\mathbf{X} b0) \rightarrow b0))$
23. $\mathbf{G}((\neg u) \rightarrow (b1 \rightarrow \mathbf{X} b1))$
24. $\mathbf{G}((\neg u) \rightarrow ((\mathbf{X} b1) \rightarrow b1))$
25. $\mathbf{G}((\neg u) \rightarrow (b2 \rightarrow \mathbf{X} b2))$
26. $\mathbf{G}((\neg u) \rightarrow ((\mathbf{X} b2) \rightarrow b2))$

// the buttons remain pressed while the corresponding floor has not
// been reached

27. $\mathbf{G}((b0 \wedge \neg f0) \rightarrow \mathbf{X} b0)$
28. $\mathbf{G}((b1 \wedge \neg f1) \rightarrow \mathbf{X} b1)$
29. $\mathbf{G}((b2 \wedge \neg f2) \rightarrow \mathbf{X} b2)$

// up is true if the lift moves up, false if it moves down, and
// unchanged if it doesn't move

30. $\mathbf{G}((f0 \wedge (\mathbf{X} f0)) \rightarrow (up \rightarrow \mathbf{X} up))$
31. $\mathbf{G}((f0 \wedge (\mathbf{X} f0)) \rightarrow ((\mathbf{X} up) \rightarrow up))$
32. $\mathbf{G}((f1 \wedge (\mathbf{X} f1)) \rightarrow (up \rightarrow \mathbf{X} up))$
33. $\mathbf{G}((f1 \wedge (\mathbf{X} f1)) \rightarrow ((\mathbf{X} up) \rightarrow up))$
34. $\mathbf{G}((f2 \wedge (\mathbf{X} f2)) \rightarrow (up \rightarrow \mathbf{X} up))$
35. $\mathbf{G}((f2 \wedge (\mathbf{X} f2)) \rightarrow ((\mathbf{X} up) \rightarrow up))$

36’. $\mathbf{G}((f0 \wedge (\mathbf{X} f1)) \rightarrow \mathbf{X} up)$
37’. $\mathbf{G}((f1 \wedge (\mathbf{X} f2)) \rightarrow \mathbf{X} up)$
38’. $\mathbf{G}((f1 \wedge (\mathbf{X} f0)) \rightarrow \mathbf{X} \neg up)$
39’. $\mathbf{G}((f2 \wedge (\mathbf{X} f1)) \rightarrow \mathbf{X} \neg up)$

// sb is true iff $b1$ or $b2$ are pressed

40. $\mathbf{G}(sb \rightarrow (b1 \vee b2))$
41. $\mathbf{G}(b1 \rightarrow sb)$
42. $\mathbf{G}(b2 \rightarrow sb)$

// when it is not used, then the lift parks at floor 0

43. $\mathbf{G}((f0 \wedge \neg sb) \rightarrow (f0 \mathbf{U}(sb \mathbf{R}((\mathbf{F} f0) \wedge \neg up))))$
44. $\mathbf{G}((f1 \wedge \neg sb) \rightarrow (f1 \mathbf{U}(sb \mathbf{R}((\mathbf{F} f0) \wedge \neg up))))$
45. $\mathbf{G}((f2 \wedge \neg sb) \rightarrow (f2 \mathbf{U}(sb \mathbf{R}((\mathbf{F} f0) \wedge \neg up))))$

// each request will eventually be served

46. $\mathbf{G}(b0 \rightarrow \mathbf{F} f0)$
47. $\mathbf{G}(b1 \rightarrow \mathbf{F} f1)$
48. $\mathbf{G}(b2 \rightarrow \mathbf{F} f2)$

// the lift never goes back to floor 0

49’. $\mathbf{XG} \neg f0$

(a) The lift specification with the fix suggested in Fig. 5 (c). We also changed the scenario to be explored (49 to 49’): we would like to know whether it is possible that the lift never returns to floor 0.

5. $f0$
7. $\neg b1$
8. $\neg b2$
40. $\mathbf{G}(sb \rightarrow (b1 \vee b2))$
43. $\mathbf{G}((f0 \wedge \neg sb) \rightarrow (f0 \mathbf{U}(sb \mathbf{R}((\mathbf{F} f0) \wedge \neg up))))$

49’. $\mathbf{XG} \neg f0$

(b) An IUC of the specification in (a) as an irreducible subset of conjuncts.

5. $f0$
7. $\neg b1$
8. $\neg b2$
40. $\mathbf{G}(sb \rightarrow (b1 \vee b2))$

43’. $\mathbf{G}((f0 \wedge \neg sb) \rightarrow (\mathbf{F}(sb \mathbf{RF} f0)))$
49’. $\mathbf{XG} \neg f0$

(c) By applying the notion of an IUC via syntax tree (Def. 8) to (b) we can further simplify: in conjunct 43 we can replace the second occurrence of $f0$ and the occurrence of $\neg up$ with 1. By using $1 \mathbf{U} \psi \equiv \mathbf{F} \psi$ and $\psi \wedge 1 \equiv 1$ we obtain 43’.

5. $f0$

7a. $x_{\neg b1}$
7b. $\mathbf{G}(x_{\neg b1} \rightarrow \neg b1)$
8a. $x_{\neg b2}$
8b. $\mathbf{G}(x_{\neg b2} \rightarrow \neg b2)$
40a. $x_{\mathbf{G}(sb \rightarrow (b1 \vee b2))}$
40b. $\mathbf{G}(x_{b1 \vee b2} \rightarrow (b1 \vee b2))$
40c. $\mathbf{G}(x_{sb \rightarrow (b1 \vee b2)} \rightarrow (sb \rightarrow x_{b1 \vee b2}))$
40d. $\mathbf{G}(x_{\mathbf{G}(sb \rightarrow (b1 \vee b2))} \rightarrow x_{sb \rightarrow (b1 \vee b2)})$
40e. $\boxed{\mathbf{G}(x_{\mathbf{G}(sb \rightarrow (b1 \vee b2))} \rightarrow \mathbf{X} x_{\mathbf{G}(sb \rightarrow (b1 \vee b2))})}$
43’a. $x_{\mathbf{G}((f0 \wedge \neg sb) \rightarrow (\mathbf{F}(sb \mathbf{RF} f0)))}$
43’b. $\mathbf{G}(x_{\neg sb} \leftarrow \neg sb)$
43’c. $\mathbf{G}(x_{f0 \wedge \neg sb} \leftarrow (f0 \wedge x_{\neg sb}))$
43’d. $\mathbf{G}(x_{\mathbf{F} f0} \rightarrow \mathbf{F} f0)$
43’e. $\mathbf{G}(x_{sb \mathbf{RF} f0} \rightarrow x_{\mathbf{F} f0})$
43’f. $\mathbf{G}(x_{sb \mathbf{RF} f0} \rightarrow (sb \vee \mathbf{X} x_{sb \mathbf{RF} f0}))$
43’g. $\mathbf{G}(x_{\mathbf{F}(sb \mathbf{RF} f0)} \rightarrow \mathbf{F} x_{sb \mathbf{RF} f0})$
43’h. $\mathbf{G}(x_{(f0 \wedge \neg sb) \rightarrow (\mathbf{F}(sb \mathbf{RF} f0))} \rightarrow (x_{f0 \wedge \neg sb} \rightarrow x_{\mathbf{F}(sb \mathbf{RF} f0)}))$
43’i. $\mathbf{G}(x_{\mathbf{G}((f0 \wedge \neg sb) \rightarrow (\mathbf{F}(sb \mathbf{RF} f0)))} \rightarrow x_{(f0 \wedge \neg sb) \rightarrow (\mathbf{F}(sb \mathbf{RF} f0))})$
43’j. $\boxed{\mathbf{G}(x_{\mathbf{G}((f0 \wedge \neg sb) \rightarrow (\mathbf{F}(sb \mathbf{RF} f0)))} \rightarrow \mathbf{X} x_{\mathbf{G}((f0 \wedge \neg sb) \rightarrow (\mathbf{F}(sb \mathbf{RF} f0)))})}$
49’a. $x_{\mathbf{XG} \neg f0}$
49’b. $\mathbf{G}(x_{\neg f0} \rightarrow \neg f0)$
49’c. $\mathbf{G}(x_{\mathbf{G} \neg f0} \rightarrow x_{\neg f0})$
49’d. $\mathbf{G}(x_{\mathbf{G} \neg f0} \rightarrow \mathbf{X} x_{\mathbf{G} \neg f0})$
49’e. $\mathbf{G}(x_{\mathbf{XG} \neg f0} \rightarrow \mathbf{X} x_{\mathbf{G} \neg f0})$

(d) This is (c) rewritten along the lines of[a] Def. 18. The corresponding IUC via dCNF doesn't require 40e and 43’j (marked blue boxed).

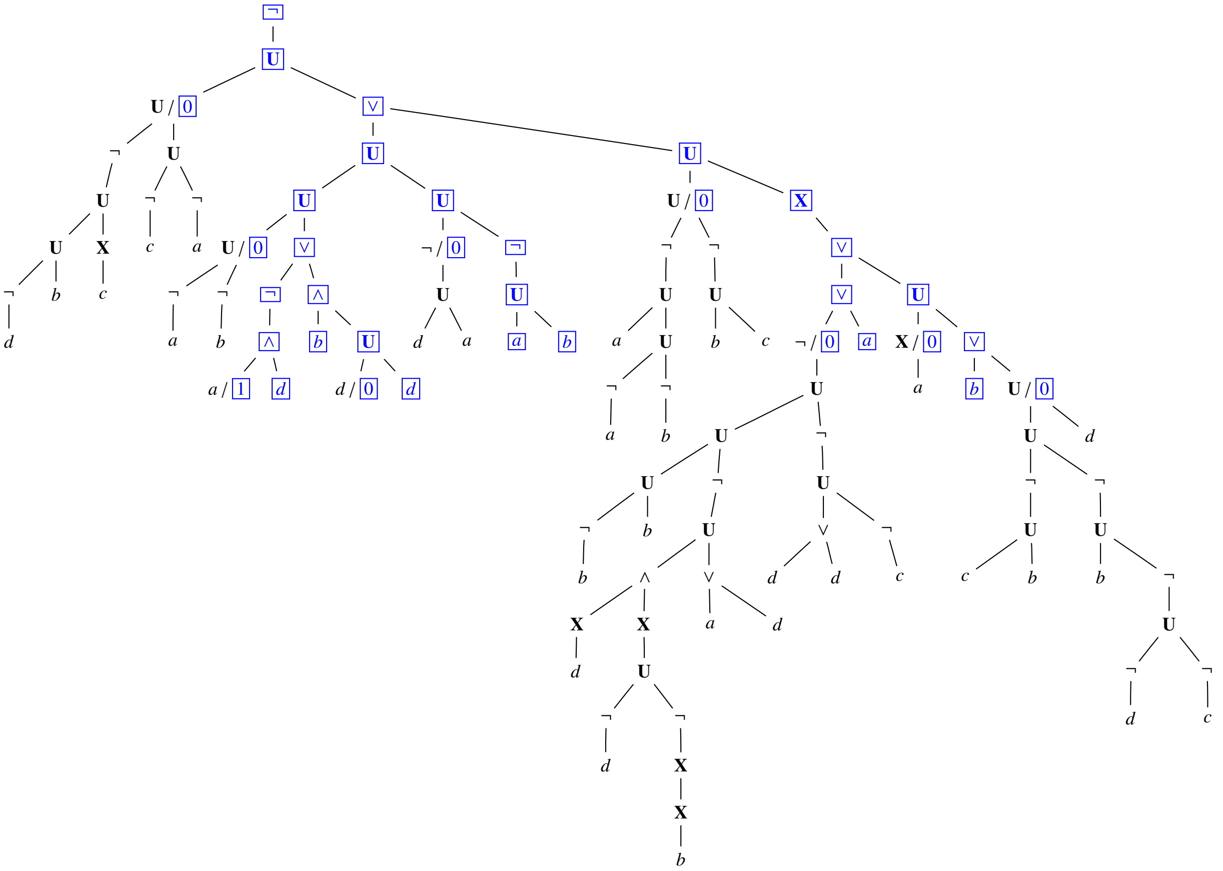
5. $f0$
7. $\neg b1$
8. $\neg b2$

40’. $sb \rightarrow (b1 \vee b2)$
43”. $(f0 \wedge \neg sb) \rightarrow (\mathbf{F}(sb \mathbf{RF} f0))$
49’. $\mathbf{XG} \neg f0$

(e) The fact that (as shown in (d)) 40e and 43’j are not needed to establish unsatisfiability makes it clear that 40 and 43’ are only relevant in the first time step. Hence, this allows to further rewrite (c) by dropping the **G** operators in 40 and 43’ (leading to 40’ and 43”).

[a] We are somewhat sloppy here by not introducing additional variables to represent the conjunction of $\{5, 7, 8, 40, 43', 49'\}$.

Figure 6: Example of using IUCs to understand impossibility of a scenario.



(a) The above syntax tree is based on an unsatisfiable instance of the random formulas benchmark set used in [RV10] (available from [roz]). The formula above was obtained by removing all occurrences of a double negation. The syntax tree induced by the blue boxed nodes forms an IUC of that formula via syntax tree. Nodes marked "$\circ$ / $\boxed{0}$" or "$\circ$ / $\boxed{1}$" indicate that in the IUC the subtree rooted at that node (with operator $\circ$) is replaced with 0 or 1, respectively. While the syntax tree of the original formula has 112 nodes that of the IUC only has 35.

$$\neg((((\neg d) \vee (b \wedge d))\mathbf{U}\neg(a\mathbf{U}b)) \vee \mathbf{X}(a \vee b))$$

(b) This is the IUC via syntax tree from (a), simplified by applying $1 \wedge \psi \equiv \psi$, $0 \vee \psi \equiv \psi$, $0\mathbf{U}\psi \equiv \psi$.

		relevant at time step
1.	$x_{\neg((((\neg d)\vee(b\wedge d))\mathbf{U}(\neg(a\mathbf{U}b)))\vee\mathbf{X}(a\vee b))}$	0
2.	$\mathbf{G}(x_{\neg d} \leftarrow \neg d)$	0
3.	$\mathbf{G}(x_{b\wedge d} \leftarrow (b \wedge d))$	0
4.	$\mathbf{G}(x_{(\neg d)\vee(b\wedge d)} \leftarrow x_{\neg d})$	0
5.	$\mathbf{G}(x_{(\neg d)\vee(b\wedge d)} \leftarrow x_{b\wedge d})$	0
6.	$\mathbf{G}(x_{a\mathbf{U}b} \rightarrow (b \vee a))$	1
7.	$\mathbf{G}(x_{a\mathbf{U}b} \rightarrow (b \vee \mathbf{X}x_{a\mathbf{U}b}))$	0
8.	$\boxed{\mathbf{G}(x_{a\mathbf{U}b} \rightarrow \mathbf{F}b)}$	—
9.	$\mathbf{G}(x_{\neg(a\mathbf{U}b)} \leftarrow \neg x_{a\mathbf{U}b})$	0,1
10.	$\mathbf{G}(x_{((\neg d)\vee(b\wedge d))\mathbf{U}(\neg(a\mathbf{U}b))} \leftarrow x_{\neg(a\mathbf{U}b)})$	0,1
11.	$\mathbf{G}(x_{((\neg d)\vee(b\wedge d))\mathbf{U}(\neg(a\mathbf{U}b))} \leftarrow (x_{(\neg d)\vee(b\wedge d)} \wedge \mathbf{X}x_{((\neg d)\vee(b\wedge d))\mathbf{U}(\neg(a\mathbf{U}b))}))$	0
12.	$\mathbf{G}(x_{a\vee b} \leftarrow a)$	1
13.	$\mathbf{G}(x_{a\vee b} \leftarrow b)$	1
14.	$\mathbf{G}(x_{\mathbf{X}(a\vee b)} \leftarrow \mathbf{X}x_{a\vee b})$	0
15.	$\mathbf{G}(x_{(((\neg d)\vee(b\wedge d))\mathbf{U}(\neg(a\mathbf{U}b)))\vee\mathbf{X}(a\vee b)} \leftarrow x_{((\neg d)\vee(b\wedge d))\mathbf{U}(\neg(a\mathbf{U}b))})$	0
16.	$\mathbf{G}(x_{(((\neg d)\vee(b\wedge d))\mathbf{U}(\neg(a\mathbf{U}b)))\vee\mathbf{X}(a\vee b)} \leftarrow x_{\mathbf{X}(a\vee b)})$	0
17.	$\mathbf{G}(x_{\neg((((\neg d)\vee(b\wedge d))\mathbf{U}(\neg(a\mathbf{U}b)))\vee\mathbf{X}(a\vee b))} \rightarrow \neg x_{(((\neg d)\vee(b\wedge d))\mathbf{U}(\neg(a\mathbf{U}b)))\vee\mathbf{X}(a\vee b)})$	0

(c) This is (b) rewritten according to Def. 18. An IUC via dCNF shows that 8 (blue boxed) is not required for unsatisfiability; in other words, the innermost **U** can be replaced with a weak **U** and the result will still be unsatisfiable. The last column adds information at which time step a conjunct is relevant for unsatisfiability as suggested in Sect. 6.

Figure 7: Example of using IUCs to understand unsatisfiability of a random formula.

9. Unrealizable Cores

In this section we consider open systems [HP85] specified in LTL, i.e., systems that interact with an environment where some signals are under the control of the system while others are under the control of the environment.

In a closed system, where all signals are controlled by the system, satisfiability of a specification is a sufficient criterion for the existence of a system that implements the specification [EC82, MW84]. However, in an open system, even if there is some behavior of the environment s.t. the resulting interaction between the system and the environment satisfies the specification, the environment may decide not to exhibit that particular behavior but a different one. Hence, the environment should be considered hostile to the system and, therefore, the system should be able to cope with all behaviors of the environment without violating its specification [PR89, ALW89].

The question of automated synthesis of open systems specified in S1S was originally stated by Church [Chu63] and later solved independently by [BL69, Rab72]. For specifications given in LTL, the problem was proved to be 2EXPTIME complete in [Ros92] (for fragments see, e.g., [AT04]), which, together with the use of the hard to implement determinization procedure [Saf88], led to the problem not receiving much attention. After some progress in overcoming [Saf88] in [KV05] and identifying somewhat easier yet powerful fragments in [PPS06] there has been renewed interest (e.g., [JB06, BGJ+07b, BGJ+07a, SSR08, CRST08a, KPP09, KS09, KHB09]).

9.1. Preliminaries

LTL Realizability. In this section we identify a Boolean variable v_p with each atomic proposition $p \in AP$ in the natural way. The set of all such variables V is partitioned into two sets of *environment variables* V^e (controlled by the environment) and *system variables* V^s (controlled by the system to be implemented): $V = V^e \uplus V^s$. A *state* $s \in S$ is a valuation of the variables: $s : V \mapsto \mathbb{B}^{|V|}$. Given a subset of variables $V' \subseteq V$, $s|_{V'}$ denotes the restriction of s to V'. An *execution* η is an infinite sequence of states: $\eta \in S^\omega$. As for infinite words, $\eta[i]$ denotes the state of η at position i and $\eta[i, \infty]$ denotes the suffix of η starting at position i (inclusive). Similarly, $\eta[0, i]$ is the prefix of η up to position i (inclusive). A *strategy* σ for the system (resp. environment) is a mapping from a finite prefix of an execution and a valuation of the environment (resp. system) variables to a valuation of the system (resp. environment) variables: $\forall i \in \mathbb{N} : (\sigma : S^i \times S|_{V^e} \mapsto S|_{V^s})$ (resp. $\sigma : S^i \times S|_{V^s} \mapsto S|_{V^e}$). An execution η is *compliant* with a system (resp. environment) strategy σ if, for all positions i, the valuation of the system (resp. environment) variables at $\eta[i]$ corresponds to the valuation prescribed by the strategy for the execution prefix up to that position: $\forall i \; . \; \eta[i]|_{V^s} = \sigma(\eta[0, i-1], \eta[i]|_{V^e})$ (resp. $\forall i \; . \; \eta[i]|_{V^e} = \sigma(\eta[0, i-1], \eta[i]|_{V^s})$). Note that in this definition of strategy both the environment and the system can take each others choices into account. Below we assume that one of the two "moves first", i.e., the first mover's strategy must not depend on the second mover's choice for the current state. The notion of *satisfaction* of an LTL formula ϕ by a word is extended to executions in the obvious way, denoted $\eta \models \phi$. Given an LTL formula ϕ, a system (resp. environment) strategy σ is *winning* for the system (resp. environment) if all executions that are compliant with σ satisfy ϕ: $\forall \eta \in S^\omega \; . \; \eta$ is compliant with $\sigma \Rightarrow \eta \models \phi$. An LTL formula ϕ is *realizable* if there exists a winning strategy for the system (*unrealizable* otherwise).

GR(1) Specifications. Generalized reactivity 1 (GR(1)) specifications represent a subclass of LTL for which the realizability problem is solvable in time cubic in the state space of the design, but which is sufficiently powerful to cover many realistic specifications [PPS06].

Given a set of Boolean variables V', we regard $\overline{V'}$ as the set of variables denoting the values of the variables in V' in their next states. Given a Boolean constraint ψ only over current state variables V we denote by $\overline{\psi}$ the version of ψ where each occurrence of a current state variable has been replaced with the corresponding next state variable. Similarly, given a Boolean constraint ψ only over next state variables $\overline{V}$ we denote by $\underline{\psi}$ the version of ψ where each occurrence of a next state variable has been replaced with the corresponding current state variable.

A GR(1) specification is given by a six tuple of sets of Boolean formulas $((I^e, R^e, B^e), (I^s, R^s, B^s))$ where 1. I^e contains variables in V^e; 2. R^e contains variables in $V^e \cup V^s \cup \overline{V^e}$; 3. B^e contains variables in $V^e \cup V^s$; 4. I^s contains variables in $V^e \cup V^s$; 5. R^s contains variables in $V^e \cup V^s \cup \overline{V^e} \cup \overline{V^s}$; and 6. B^s contains variables in $V^e \cup V^s$. Intuitively (I^e, R^e, B^e) (resp. (I^s, R^s, B^s)) describes a Büchi fair transition system by giving its initial states I^e (resp. I^s), transition relation R^e (resp. R^s), and set of Büchi fairness constraints B^e (resp. B^s). The intended semantics is then that, as long as the environment evolves according to (I^e, R^e, B^e), the system must obey (I^s, R^s, B^s) and vice versa. Formally,

$$\begin{array}{ll}
((I^e, R^e, B^e), (I^s, R^s, B^s)) \equiv & \\
\quad ((\bigwedge_{\iota^e \in I^e} \iota^e) \rightarrow (\bigwedge_{\iota^s \in I^s} \iota^s)) & \wedge \\
\quad \Big(\mathbf{G}\Big(\Big(((\mathbf{OH} \bigwedge_{\iota^s \in I^s} \iota^s) \wedge (\mathbf{YH} \bigwedge_{\rho^s \in R^s} \rho^s)) \rightarrow (\bigwedge_{\rho^e \in R^e} \rho^e)\Big) \rightarrow & \\
\qquad \Big(((\mathbf{OH} \bigwedge_{\iota^e \in I^e} \iota^e) \wedge (\mathbf{YH} \bigwedge_{\rho^e \in R^e} \rho^e)) \rightarrow (\bigwedge_{\rho^s \in R^s} \rho^s)\Big)\Big)\Big) & \wedge \\
\quad \Big(\Big(((\bigwedge_{\iota^s \in I^s} \iota^s) \wedge (\mathbf{G} \bigwedge_{\rho^s \in R^s} \rho^s)) \rightarrow (\bigwedge_{\beta^e \in B^e} \mathbf{GF}\beta^e)\Big) \rightarrow & \\
\quad\; \Big(((\bigwedge_{\iota^e \in I^e} \iota^e \wedge (\mathbf{G} \bigwedge_{\rho^e \in R^e} \rho^e)) \rightarrow (\bigwedge_{\beta^s \in B^s} \mathbf{GF}\beta^s\Big)\Big) &
\end{array} \tag{1}$$

Here **O** ("in the current or in some past state"), **H** ("in the current and in all past states"), and **Y** ("in the previous state") are past time LTL operators (see, e.g., [Eme90]) that correspond to the future time **F**, **G**, and **X** respectively. Hence, **OH** means "in the initial state" and **YH** means "in all states between the initial state and the previous state (inclusive)". Clearly, Eqn. 1 can be written more compactly; here we emphasize the idea that the system needs to comply with its obligations (only) as long as the environment does (and vice versa). Finally, note that w.l.o.g. in our formulation the environment moves first. For more details about GR(1) realizability see [PPS06].

9.2. Unrealizable Cores via Syntax Trees

We now extend the notion of a UC based on syntax trees from Sect. 4 to realizability. In fact, we argue that syntactic weakening, which is at the heart of Def. 9, fulfills the criteria outlined in Sect. 3 also for realizability. That allows us to reuse Def. 9 and obtain the definition of an unrealizable core of a syntax tree by substituting "unrealizable" for "unsatisfiable" in Def. 10.

Fulfillment of criteria 2 (unrealizability is easier to see for the user) and 3 (non-addition/preservance of reasons for unrealizability can be understood by the user) is as for unsatisfiable cores.

Syntactic weakening can be seen to fulfill criterion 1 (preservation of reasons without adding new ones) also in the case of realizability as follows. [GBJV08, Gre07] introduce[8] the notion of *open implication* to complement the standard notion of implication (termed *trace implication* in [GBJV08]) between LTL formulas in the context of open systems, i.e., realizability. Trace implication is suited to distinguish two LTL formulas ϕ and ϕ' in terms of satisfiability: ϕ implies ϕ', $\phi \rightarrow \phi'$, if the set of words satisfying ϕ is a subset of the words satisfying ϕ'. On the other hand, [GBJV08] argues that in the context of realizability it is more useful to consider the set of implementations realizing a formula, i.e., ϕ open implies ϕ', written $\phi \twoheadrightarrow\!\!\!\!\circ\;\, \phi'$, iff the set of implementations realizing ϕ is a subset of the set of implementations realizing ϕ'. Clearly, if $\phi \rightarrow \phi'$, then also $\phi \twoheadrightarrow\!\!\!\!\circ\;\, \phi'$. However, if there are some words that fulfill ϕ but not ϕ', then it may still be the case that $\phi \twoheadrightarrow\!\!\!\!\circ\;\, \phi'$. In that case the words in the difference between $L(\phi)$ and $L(\phi')$ cannot be used to construct an implementation because they require clairvoyance [GBJV08]. Hence, trace implication refines open implication. Therefore, syntactic weakening, which is based on trace implication, yields a suitable method to construct unrealizable cores, too.

We finally state

Definition 27 (Unrealizable Core of a Syntax Tree). Let pt, pt' be syntax trees. pt' is an *unrealizable core* of pt if 1. $f(pt)$ is unrealizable, 2. pt' is a core of pt, and 3. $f(pt')$ is unrealizable. pt' is an *irreducible unrealizable core* of pt if there does not exist a proper unrealizable core of pt'.

Example. Consider the following specification with environment variables v^e, $v^{e\prime}$ and system variable v^s:

$$\phi \equiv (v^e \vee v^{e\prime}) \rightarrow \mathbf{G}(((\mathbf{X}(v^e \vee v^{e\prime})) \rightarrow v^s) \wedge ((\mathbf{X}\neg(v^e \vee v^{e\prime})) \rightarrow \neg v^s))$$

In other words, if at least one of the environment variables holds in the initial state, then the system is supposed to foresee the disjunction of the two environment variables — obviously this is impossible. ϕ has the following four irreducible unrealizable cores via syntax tree:

1. $(0 \vee v^{e\prime}) \rightarrow \mathbf{G}(((\mathbf{X}(0 \vee v^{e\prime})) \rightarrow v^s) \wedge ((\mathbf{X}\neg(v^e \vee v^{e\prime})) \rightarrow \neg v^s))$,

[8] For an earlier account of an equivalence relation on properties based on having the same set of implementations see [AL93].

2. $(0 \vee v^{e\prime}) \rightarrow \mathbf{G}(((\mathbf{X}(v^e \vee 0)) \rightarrow v^s) \wedge ((\mathbf{X}\neg(v^e \vee v^{e\prime})) \rightarrow \neg v^s))$.

3. $(v^e \vee 0) \rightarrow \mathbf{G}(((\mathbf{X}(0 \vee v^{e\prime})) \rightarrow v^s) \wedge ((\mathbf{X}\neg(v^e \vee v^{e\prime})) \rightarrow \neg v^s))$, and

4. $(v^e \vee 0) \rightarrow \mathbf{G}(((\mathbf{X}(v^e \vee 0)) \rightarrow v^s) \wedge ((\mathbf{X}\neg(v^e \vee v^{e\prime})) \rightarrow \neg v^s))$.

9.3. Unrealizable Cores for GR(1) via Definitional Conjunctive Normal Form

We are not aware of a definitional conjunctive normal form that allows to obtain equirealizable formulas for arbitrary LTL formulas. GR(1) specifications have a fixed temporal structure at the top of the corresponding formula and only at the lower level a variable, Boolean structure. Moreover, users are likely to think of a GR(1) specification in terms of its 6 sets of constraints. This suggests to approach unrealizable cores for GR(1) formulas via dCNF at the level of the 6 sets of constraints, leaving the top level temporal structure untouched. Hence, the resulting notion will take a GR(1) specification and produce another GR(1) specification.

Similar approaches were presented in [CRST08a] and [KHB09]. Both approaches compute an unrealizable core by removing members of the 3 sets of constraints for the system. [CRST08a] additionally allows to reduce the 3 sets of constraints for the environment. This constitutes syntactic strengthening rather than weakening; the motivation here is to remove environment constraints which do not help to make the remaining system constraints realizable. [KHB09] not only removes system constraints but also system variables in the remaining system constraints.

Here we partially go beyond [CRST08a, KHB09] in two respects. First, we transform all constraints into Boolean CNF; this permits core extraction by removing members of the set of constraints to proceed inside the constraints of the system of the original specification. Second, we transform the specification such that there are system constraints whose removal corresponds to a weakening of the original specification on the side of the environment[9].

The combination of both steps yields a reduction that enables the use of the tools developed in [CRST08a, KHB09] to obtain unrealizable cores of improved granularity, as compared to the original versions of [CRST08a, KHB09], without modifying the original tools.

Below we first state our results and then later describe the required transformations and the accompanying theorems establishing their correctness.

Results

Definition 28 (Core of a Boolean dCNF for GR(1)). Let ϕ be a GR(1) specification and let $\phi' = ((I^{e\prime}, R^{e\prime}, B^{e\prime}), (I^{s\prime}, R^{s\prime}, B^{s\prime})) = e2s(ts(dCNF^{GR(1)}(\phi)))$. Let $\phi'' = ((I^{e\prime}, R^{e\prime}, B^{e\prime}), (I^{s\prime\prime}, R^{s\prime\prime}, B^{s\prime\prime}))$ with 1. $I^{s\prime\prime} \subseteq I^{s\prime}$, 2. $R^{s\prime\prime} \subseteq R^{s\prime}$, and 3. $B^{s\prime\prime} \subseteq B^{s\prime}$. Then ϕ'' is a *core* of ϕ' (and of ϕ). ϕ'' is a *proper core* if any of the subset relations is strict.

Definition 29 (Unrealizable Core of a Boolean dCNF for GR(1)). Let ϕ' be a core of ϕ. ϕ' is an *unrealizable core* of ϕ if both ϕ and ϕ' are unrealizable. ϕ' is an *irreducible unrealizable core* of ϕ if there does not exist a proper unrealizable core of ϕ.

We conjecture that the granularity of the unrealizable cores obtained in such a way is comparable to the granularity of unrealizable cores via syntax tree when treating repeated occurrences of the constraints in $I^e, R^e, B^e, I^s, R^s, B^s$ in Eqn. 1 as shared.

Step 1: Transforming the Constraints of a GR(1) Formula into Boolean dCNF

First note that all 6 sets of constraints that characterize a GR(1) formula are sets of Boolean formulas over current and next state environment and system variables. Clearly, by ignoring the temporal aspect inherent in the current and next state variables, i.e., regarding them as independent Boolean variables, each such constraint can be translated into an equisatisfiable Boolean dCNF by the following variant of Def. 11:

Definition 30 (Boolean Definitional Conjunctive Normal Form). Let ϕ be a Boolean formula over Boolean variables V, let $x, x', \ldots \in X$ be fresh Boolean variables not in V. $dCNF^{\mathbb{B}}_{aux}(\phi)$ is a set of conjuncts containing one conjunct for each occurrence of a subformula ψ in ϕ as follows:

[9] Note that is is different from [CRST08a] whose actions on the side of the environment syntactically strengthen the specification.

ψ	Conjunct $\in dCNF^{\mathbb{B}}_{aux}(\phi)$
b with $b \in \mathbb{B}$	$x_\psi \leftrightarrow b$
v with $v \in V$	$x_\psi \leftrightarrow v$
$\neg\psi'$	$x_\psi \leftrightarrow \neg x_{\psi'}$
$\psi' \circ_2 \psi''$ with $\circ_2 \in \{\vee, \wedge\}$	$x_\psi \leftrightarrow x_{\psi'} \circ_2 x_{\psi''}$

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

$$dCNF^{\mathbb{B}}(\phi) \equiv x_\phi \wedge \bigwedge_{c \in dCNF^{\mathbb{B}}_{aux}(\phi)} c$$

It is natural to see $dCNF^{\mathbb{B}}(\phi)$ as a set containing the root of $dCNF^{\mathbb{B}}(\phi)$ and its conjuncts in $dCNF^{\mathbb{B}}_{aux}(\phi)$. This allows for a straightforward extension of the $dCNF^{\mathbb{B}}$ to sets of Boolean formulas. Given such a set Φ of Boolean formulas, $roots(dCNF^{\mathbb{B}}(\Phi))$ and $conjs(dCNF^{\mathbb{B}}(\Phi))$ denote the set of roots and conjuncts in $dCNF^{\mathbb{B}}(\Phi)$, respectively.

Next we show how to lift the equisatisfiability at the purely Boolean level to equirealizability for a GR(1) formula. Basically it is left to decide 1. for each fresh variable, whether it should be an environment or a system variable, 2. for each fresh variable, whether it should be a current or a next state variable, and 3. for each root and for each conjunct, which of the 6 sets of constraints it should belong to.

Definition 31 (Boolean dCNF for GR(1) Realizability). Let $\phi = ((I^e, R^e, B^e), (I^s, R^s, B^s))$ be a GR(1) specification with environment variables V^e and system variables V^s. We construct the Boolean dCNF for GR(1) realizability of ϕ, $dCNF^{GR(1)}(\phi) = ((I^e_{dc}, R^e_{dc}, B^e_{dc}), (I^s_{dc}, R^s_{dc}, B^s_{dc}))$ as follows.

$$\begin{array}{lcl@{\qquad\qquad}lcl} I^e_{dc} & \equiv & dCNF^{\mathbb{B}}(I^e) & I^s_{dc} & \equiv & dCNF^{\mathbb{B}}(I^s) \\ R^e_{dc} & \equiv & dCNF^{\mathbb{B}}(R^e) \cup conjs(dCNF^{\mathbb{B}}(B^e)) & R^s_{dc} & \equiv & dCNF^{\mathbb{B}}(R^s) \cup conjs(dCNF^{\mathbb{B}}(B^s)) \\ B^e_{dc} & \equiv & \underline{roots(dCNF^{\mathbb{B}}(B^e))} & B^s_{dc} & \equiv & roots(dCNF^{\mathbb{B}}(B^s)) \end{array}$$

The fresh variables introduced when constructing the Boolean dCNFs are assigned as follows. Fresh variables originating 1. from I^e become current state environment variables, 2. from R^e become next state environment variables, 3. from B^e become next state environment variables, 4. from I^s become current state system variables, 5. from R^s become next state system variables, and 6. from B^s become current state system variables.

The intuition behind the previous definition is as follows. The fresh variables are assigned to the system (resp. environment) iff the original constraints are system (resp. environment) constraints. Moreover, the fresh variables are assigned to the earliest state at which all required information for their assignment is available. The special case here is B^e, which contains only current state variables, but from both system and environment. Hence, in this case the fresh variables need to be (environment) next state variables.[10] Finally, the conjuncts for I^e, R^e, I^s, and R^s can be added directly to I^e_{dc}, R^e_{dc}, I^s_{dc}, and R^s_{dc}. This is not possible for B^e and B^s because $\mathbf{GF}(\psi \wedge \psi') \neq (\mathbf{GF}\psi) \wedge (\mathbf{GF}\psi')$. Hence, the conjuncts for the fairness constraints are added to R^e_{dc} and R^s_{dc}. Finally, we remark that our introduction of the notion of current and next state variables in Sect. 9.1 assumes that there is one next state variable for each current state variable and vice versa; hence, we tacitly assume that for each current (resp. next) state variable added above the corresponding next (resp. current) state variable is added, too.

Theorem 32 (Equirealizability of ϕ and $dCNF^{GR(1)}(\phi)$). *Let $\phi = ((I^e, R^e, B^e), (I^s, R^s, B^s))$ be a GR(1) specification over the set of variables V and $dCNF^{GR(1)}(\phi) = ((I^e_{dc}, R^e_{dc}, B^e_{dc}), (I^s_{dc}, R^s_{dc}, B^s_{dc}))$ its Boolean dCNF for GR(1) realizability. Then*

1. *A winning strategy for the system (resp. environment) in $dCNF^{GR(1)}(\phi)$ can be generated from a corresponding winning strategy in ϕ by assigning the fresh system (resp. environment) variables the unique values determined by the biimplications of the Boolean dCNF.*

[10] More precisely, they are next state variables in R^e_{dc} but are "cast" to current state variables in B^e_{dc}.

2. *ϕ is realizable iff $dCNF^{GR(1)}(\phi)$ is realizable.*

Proof. (Sketch.) To show claim 1 let σ be a winning strategy for the system (resp. environment) in ϕ. Let σ_{dc} be the strategy for the system (resp. environment) in $dCNF^{GR(1)}(\phi)$ that assigns each fresh system (resp. environment) variable the unique value determined by the biimplications of the Boolean dCNF. We have to show that σ_{dc} is a winning strategy for the system (resp. environment) in $dCNF^{GR(1)}(\phi)$.

First note that σ_{dc} is indeed a strategy in that its computation only requires knowledge of valuations of variables available at the respective point in time.

Let η_{dc} be an execution of $dCNF^{GR(1)}(\phi)$ compliant with σ_{dc}. We show that η_{dc} is winning for the system (resp. environment).

Let $\eta \equiv \eta_{dc}|_V$ be the projection of η_{dc} onto the set of variables in ϕ. By assumption η is compliant with σ and, hence, winning for the system (resp. environment) in ϕ.

Let i be the smallest position in η s.t. the environment (resp. system) violates one of its constraints in $I^e \cup R^e$ (resp. $I^s \cup R^s$) or ∞ if such position does not exist. Furthermore, let i_{dc} be the smallest position in η_{dc} s.t. the environment (resp. system) does not assign each fresh environment (resp. system) variable the unique value determined by the biimplications of the Boolean dCNF or ∞ if such position does not exist.

First consider the case that $i < i_{dc}$ or $i = i_{dc} = \infty$. Intuitively, in this case η_{dc} is winning for the system in $dCNF^{GR(1)}(\phi)$ for the same reason that η is winning for the system in ϕ. (Slightly) more formally, by construction of σ_{dc} the system satisfies all its conjuncts in $dCNF^{GR(1)}(\phi)$ on η_{dc} at all positions and a system root in $dCNF^{GR(1)}(\phi)$ is 1 on η_{dc} at a given position iff the corresponding constraint in ϕ is 1 on η at that position. By the assumption about i_{dc} the environment satisfies all its conjuncts in $dCNF^{GR(1)}(\phi)$ on η_{dc} at all positions up to and including i. That implies that between positions 0 and i the environment satisfies an environment root in $dCNF^{GR(1)}(\phi)$ iff the corresponding constraint in ϕ is 1 on η at that position. Hence, as η is winning for the system in ϕ, so is η_{dc} in $dCNF^{GR(1)}(\phi)$. (The respective case for the environment is similar.)

Now consider the case that $i_{dc} \leq i$ and $i_{dc} < \infty$. Here, intuitively, the system wins because the environment assigns values to the fresh variables that are different from the values determined by the biimplications of the Boolean dCNF. Again, by construction of σ_{dc} the system satisfies all its conjuncts in $dCNF^{GR(1)}(\phi)$ on η_{dc} at all positions and a system root in $dCNF^{GR(1)}(\phi)$ is 1 on η_{dc} at a given position iff the corresponding constraint in ϕ is 1 on η at that position. By the assumption about i_{dc} the environment satisfies all its conjuncts in $dCNF^{GR(1)}(\phi)$ on η_{dc} at all positions up to and including $i_{dc} - 1$ but fails at least one of them at position i_{dc}. Hence, as η is winning for the system, so is η_{dc}. (The respective case for the environment is similar.)

Claim 2 follows from claim 1.

Step 2: Strengthening Environment Constraints by Removing System Constraints

Definition 33 (Time Shift of a GR(1) Specification). Let $\phi = ((I^e, R^e, B^e), (I^s, R^s, B^s))$ be a GR(1) specification over the set of variables V s.t. $init, \overline{init}$ are fresh variables. The *time shift* of ϕ, $ts(\phi) = ((I^e_{ts}, R^e_{ts}, B^e_{ts}), (I^s_{ts}, R^s_{ts}, B^s_{ts}))$ is defined as follows:

$$\begin{array}{lcl}
I^e_{ts} & \equiv & \{init\} \\
R^e_{ts} & \equiv & \{\neg\overline{init}\} \cup \{init \rightarrow (\overline{\iota^e}) \mid \iota^e \in I^e\} \cup \{(\neg init) \rightarrow \rho^e \mid \rho^e \in R^e\} \\
B^e_{ts} & \equiv & B^e \\
I^s_{ts} & \equiv & \emptyset \\
R^s_{ts} & \equiv & \{init \rightarrow (\overline{\iota^s}) \mid \iota^s \in I^s\} \cup \{(\neg init) \rightarrow \rho^s \mid \rho^s \in R^s\} \\
B^s_{ts} & \equiv & B^s
\end{array}$$

Theorem 34 (Equirealizability of ϕ and $ts(\phi)$ for GR(1)). *Let $\phi = ((I^e, R^e, B^e), (I^s, R^s, B^s))$ be a GR(1) specification over the set of variables V s.t. $init, \overline{init}$ are fresh variables and $ts(\phi) = ((I^e_{ts}, R^e_{ts}, B^e_{ts}), (I^s_{ts}, R^s_{ts}, B^s_{ts}))$ its time shift. Then*

1. *If σ is a winning strategy for the environment in ϕ, then σ_{ts} as defined below is a winning strategy for the environment in $ts(\phi)$:*

$$\begin{array}{rrlcl}
 & & \sigma_{ts}(\epsilon)(init) & = & 1 \\
 & \forall v^e \in V^e \,. & \sigma_{ts}(\epsilon)(v^e) & \in & \mathbb{B} \\
\forall i \geq 0 \,. & & \sigma_{ts}(\eta_{ts}[0,i])(init) & = & 0 \\
\forall i \geq 0 \,. & \forall v^e \in V^e \,. & \sigma_{ts}(\eta_{ts}[0,i])(v^e) & = & \sigma(\eta_{ts}[1,i]|_V)(v^e)
\end{array}$$

2. *If σ is a winning strategy for the system in ϕ, then σ_{ts} as defined below is a winning strategy for the system in $ts(\phi)$:*

$$\begin{array}{rrcl} & \forall v^s \in V^s \ . & \sigma_{ts}((\epsilon, s^e_{ts}))(v^s) & \in & \mathbb{B} \\ \forall i \geq 0 \ . & \forall v^s \in V^s \ . & \sigma_{ts}((\eta_{ts}[0,i], s^e_{ts}))(v^s) & = & \sigma((\eta_{ts}[1,i]|_V, s^e_{ts}|_V))(v^s) \end{array}$$

3. *ϕ is realizable iff $ts(\phi)$ is realizable.*

Proof. (Sketch.) Claim 1: Essentially the environment follows the same strategy in $ts(\phi)$ as in ϕ; it just delays the start of the interaction by one step. It's easy to see that any execution of $ts(\phi)$ compliant with σ_{ts} that has its first state removed and *init* projected away is an execution of ϕ compliant with σ and, hence, winning for the environment in ϕ. Moreover, resulting from the way the environment handles *init* in σ_{ts}, $ts(\phi)$ reduces to a variant of ϕ that is delayed by one time step. Hence, an execution of $ts(\phi)$ compliant with σ_{ts} is winning for the environment.

The proof of claim 2 is similar to the proof of Thm. 32 and claim 1. Either the environment complies with the restrictions imposed on in for *init* at least as long as it has not yet lost on the corresponding execution in ϕ; in that case the environment will lose for the same reason in $ts(\phi)$ as it loses in ϕ. Or, on the other hand, the environment does not comply with the restrictions for *init*; then it will lose for that reason.

Claim 3 follows from claims 1 and 2.

In the following Def. 35 we introduce system constraints that, when present, leave the environment constraints untouched, when absent, from the point of view of the environment, syntactically strengthen the environment constraints. We use $\top$ to denote 0 if a subformula ψ of some constraint $\phi \in I^e \cup R^e$ has positive polarity in ϕ and 1 otherwise.

Definition 35 (Handing Control over to the System). Let $\phi = ((I^e, R^e, B^e), (I^s, R^s, B^s))$ be a GR(1) specification over the set of variables V. Let $\phi_{dc} = dCNF^{GR(1)}(\phi) = ((I^e_{dc}, R^e_{dc}, B^e_{dc}), (I^s_{dc}, R^s_{dc}, B^s_{dc}))$ be the Boolean dCNF for GR(1) realizability. Let $\phi_{ts} = ts(\phi_{dc}) = ((I^e_{ts}, R^e_{ts}, B^e_{ts}), (I^s_{ts}, R^s_{ts}, B^s_{ts}))$ be the time shift of ϕ_{dc}. Then $\phi_{e2s} = e2s(\phi_{ts}) = ((I^e_{e2s}, R^e_{e2s}, B^e_{e2s}), (I^s_{e2s}, R^s_{e2s}, B^s_{e2s}))$ is defined as follows:

$$\begin{align} I^e_{e2s} &\equiv \{init\} \tag{1}\\ R^e_{e2s} &\equiv \{\neg\overline{init}\} \cup \tag{2}\\ &\quad \{init \rightarrow (m^s_\psi \leftrightarrow \overline{m^e_\psi}) \mid \psi \in conjs(I^e_{dc} \cup R^e_{dc})\} \cup \tag{3}\\ &\quad \{(\neg init) \rightarrow (m^e_\psi \leftrightarrow \overline{m^e_\psi}) \mid \psi \in conjs(I^e_{dc} \cup R^e_{dc})\} \cup \tag{4}\\ &\quad \{init \rightarrow (\overline{\psi}) \mid \psi \in roots(I^e_{dc})\} \cup \tag{5}\\ &\quad \{init \rightarrow ((\overline{(\neg m^e_\psi) \wedge (x_\psi \leftrightarrow \top)}) \vee (\overline{m^e_\psi \wedge (x_\psi \leftrightarrow \psi)})) \mid x_\psi \leftrightarrow \psi \in conjs(I^e_{dc})\} \cup \tag{6}\\ &\quad \{(\neg init) \rightarrow \psi \mid \psi \in roots(R^e_{dc})\} \cup \tag{7}\\ &\quad \{(\neg init) \rightarrow (((\neg m^e_\psi) \wedge (x_\psi \leftrightarrow \top)) \vee (m^e_\psi \wedge (x_\psi \leftrightarrow \psi))) \mid x_\psi \leftrightarrow \psi \in conjs(R^e_{dc})\} \tag{8}\\ B^e_{e2s} &\equiv B^e_{ts} \tag{9}\\ I^s_{e2s} &\equiv \{m^s_\psi \mid \psi \in conjs(I^e_{dc} \cup R^e_{dc})\} \tag{10}\\ R^s_{e2s} &\equiv R^s_{ts} \tag{11}\\ B^s_{e2s} &\equiv B^s_{ts} \tag{12} \end{align}$$

The intuition behind Def. 35 is as follows. For each environment conjunct ψ in ϕ_{dc} we introduce one fresh system variable m^s_ψ and one fresh environment variable m^e_ψ[11]. The environment variable m^e_ψ is used to multiplex between the original constraint (when m^e_ψ is 1) and its (from an environment point of view) strengthened version (when m^e_ψ is 0)

[11] Note that B^e_{dc} only contains roots.

in lines (6) and (8). Environment roots are left untouched (lines (5), (7)). While m^e_ψ is an environment variable, it is forced to copy the initial value of the corresponding system variable m^s_ψ (line (2)) and keep that value (line (4)). Lines (9), (11), and (12) are unchanged from ϕ_{ts}. Above the system is forced to set its variables m^s_ψ to 1 in the initial state (line (10)), thus rendering ϕ_{e2s} equirealizable to ϕ_{ts}. However, by removing one or more constraints in line (10) we can allow the system to set the initial values of the m^s_ψs to 0 and, therefore, enforce stronger constraints on the side of the environment (i.e., weaken the specification as a whole).

Theorem 36 (Equirealizability of ϕ_{ts} and $e2s(\phi_{ts})$). *Let $\phi = ((I^e, R^e, B^e), (I^s, R^s, B^s))$ be a GR(1) specification over the set of variables V. Let $\phi_{dc} = dCNF^{GR(1)}(\phi) = ((I^e_{dc}, R^e_{dc}, B^e_{dc}), (I^s_{dc}, R^s_{dc}, B^s_{dc}))$ be the Boolean dCNF for GR(1) realizability. Let $\phi_{ts} = ts(\phi_{dc}) = ((I^e_{ts}, R^e_{ts}, B^e_{ts}), (I^s_{ts}, R^s_{ts}, B^s_{ts}))$ be the time shift of ϕ_{dc}. Let $\phi_{e2s} = e2s(\phi_{ts}) = ((I^e_{e2s}, R^e_{e2s}, B^e_{e2s}), (I^s_{e2s}, R^s_{e2s}, B^s_{e2s}))$. Then*

1. *if σ_{ts} is a winning strategy for the system in ϕ_{ts}, then σ_{e2s} as defined below is a winning strategy for the system in $e2s(\phi_{ts})$:*

$$\begin{array}{lllll}
 & \forall \psi \in conjs(I^e_{dc} \cup R^e_{dc}) \;. & \sigma_{e2s}(\epsilon, s^e_{e2s})(m^s_\psi) & = & 1 \\
\forall i \geq 0 \;. & \forall \psi \in conjs(I^e_{dc} \cup R^e_{dc}) \;. & \sigma_{e2s}(\eta_{e2s}[0,i], s^e_{e2s})(m^s_\psi) & \in & \mathbb{B} \\
\forall i \geq -1 \;. & \forall v^s_{ts} \in V^s_{ts} \;. & \sigma_{e2s}(\eta_{e2s}[0,i], s^e_{e2s})(v^s_{ts}) & = & \sigma_{ts}(\eta_{e2s}[0,i]|_{V^s_{ts}}, s^e_{e2s}|_{V^e_{ts}})(v^s_{ts})
\end{array}$$

2. *if σ_{ts} is a winning strategy for the environment in ϕ_{ts}, then σ_{e2s} as defined below is a winning strategy for the environment in $e2s(\phi_{ts})$:*

$$\begin{array}{lllll}
 & \forall \psi \in conjs(I^e_{dc} \cup R^e_{dc}) \;. & \sigma_{e2s}(\epsilon)(m^e_\psi) & \in & \mathbb{B} \\
\forall i \geq 0 \;. & \forall \psi \in conjs(I^e_{dc} \cup R^e_{dc}) \;. & \sigma_{e2s}(\eta_{e2s}[0,i])(m^e_\psi) & = & \eta_{e2s}[0](m^s_\psi) \\
\forall i \geq -1 \;. & \forall v^e_{ts} \in V^e_{ts} \;. & \sigma_{e2s}(\eta_{e2s}[0,i])(v^e_{ts}) & = & \sigma_{ts}(\eta_{e2s}[0,i]|_{V^e_{ts}})(v^e_{ts})
\end{array}$$

3. *ϕ_{ts} is realizable iff $e2s(\phi_{ts})$ is realizable.*

Proof. (Sketch.) We start with 1. The system sets its multiplex variables m^s_ψ to 1 in the initial state. Either the environment adheres to its constraints for its corresponding multiplex variables and sets them to 1 (at least as long as it hasn't lost due to other reasons); in that case the environment constraints reduce to those of ϕ_{ts} and the environment loses in ϕ_{e2s} due to the same reason it lost in ϕ_{ts}. Or the environment doesn't adhere to its constraints for the multiplex variables and loses due to that reason.

Similarly, for 2, if the system obeys its initial constraints on its multiplex variables, then the environment simply replicates these values and, hence, reduces its constraints to those of ϕ_{ts}, thus winning for the same reason as in ϕ_{ts}. Otherwise the environment wins because the system doesn't fulfill its initial constraints.

Claim 3 follows from claims 1 and 2.

10. Relation to Vacuity and Some Complexity

10.1. Relation to Vacuity

Vacuity Checking. Vacuity checking [BBDER01, KV03, AFF+03, BFG+05, GC04b, GC04a, SV04, SV07, 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 a 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, CS09] 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, CS09] 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.

[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 a UC as soon as the generated CNF is unsatisfiable for some k.

[SV07] suggests to generalize the operations to strengthen a specification by considering a form of interpolants between a model and its specification. While this might lead to another possibility to derive a core from a formula, an arbitrary interpolant might not allow the user to easily see what is happening. Hence, [SV07] needs to be concretized to meet that criterion.

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 a UC of an LTL formula or considers dCNFs as we do.

Inherent Vacuity. [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 (V): occurrences of subformulas (s_V), sharing of subformulas (m_V), etc., 2. equivalence type (E): closed (c_E) or open (o_E) systems, 3. tightening type (T): equivalence (e_T) or preservance (p_T) of satisfiability/realizability, and 4. polarity type (P): strengthening (s_P) or weakening (s_W). An instance of the framework is given by a four tuple (V, E, T, P).

Our notion of UCs via syntax 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] ($V = s_V$), systems be closed ($E = c_E$), tightening type be equivalence ($T = e_T$), and polarity type be weakening ($P = w_P$). Then the following is immediate by the respective definitions

Proposition 37 (Relation between inherent vacuity and unsatisfiable cores). *Let ϕ, ϕ' be unsatisfiable LTL formulas s.t. ϕ' is derived from ϕ by replacing a single occurrence of a positive (or negative) polarity subformula ψ of ϕ with* 1 *(or* 0*); hence, ϕ' is a proper UC of ϕ by Def. 10. Then 1. ϕ is inherently vacuous of type (s_V, c_E, e_T, w_P). 2. ϕ' is an IUC iff it is not inherently vacuous of type (s_V, c_E, e_T, w_P).*

Similarly, for $E = o_E$:

Proposition 38 (Relation between inherent vacuity and unrealizable cores). *Let ϕ, ϕ' be unrealizable LTL formulas s.t. ϕ' is derived from ϕ by replacing a single occurrence of a positive (or negative) polarity subformula ψ of ϕ with* 1 *(or* 0*); hence, ϕ' is a proper unrealizable core of ϕ by Def. 27. Then 1. ϕ is inherently vacuous of type (s_V, o_E, e_T, w_P). 2. ϕ' is an irreducible unrealizable core iff it is not inherently vacuous of type (s_V, o_E, e_T, w_P).*

[FKSFV08] focuses on satisfiable/realizable instances and doesn't make a connection to the notion of unsatisfiable or unrealizable cores.

10.2. Some Complexity Results

In this paper, we are mainly concerned with the following search problems:

Definition 39 (Iuc-Search-ST). Given an LTL formula ϕ, determine an IUC ϕ' of ϕ via syntax tree (if ϕ is unsatisfiable) or output “satisfiable” (if ϕ is satisfiable).

Definition 40 (Iuc-Search-dCNF). Given an LTL formula ϕ, determine an IUC ϕ' of ϕ via dCNF (if ϕ is unsatisfiable) or output “satisfiable” (if ϕ is satisfiable).

Definition 41 (Irc-Search). Given an LTL formula ϕ, determine an irreducible unrealizable core ϕ' of ϕ via syntax tree (if ϕ is unrealizable) or output “realizable” (if ϕ is realizable).

In addition, the following decision problems (similar to, e.g., [FKSFV08, KV03, BBDER01, AFF^{+}03, BFG^{+}05, GC04a]) characterize what often constitutes an elementary step in a naive algorithm to compute an unsatisfiable or unrealizable core and whether a formula is an irreducible unsatisfiable or unrealizable core:

Definition 42 (Iuc-Step-Dec-ST). Given an LTL formula ϕ and a positive (or negative) polarity occurrence of a subformula ψ, answer “yes” if ϕ with ψ set to 1 (or 0) is a UC of ϕ via syntax tree, “no” otherwise.

Definition 43 (Iuc-Step-Dec-dCNF). Let ϕ be an LTL formula with dCNF $dCNF(\phi)$, let c be a conjunct in $dCNF_{aux}(\phi)$. Let $dCNF'$ be the largest core of $dCNF(\phi)$ with c removed from $dCNF_{aux}(\phi)$. Answer “yes” if $dCNF'$ is a UC of ϕ via dCNF, “no” otherwise.

Definition 44 (Irc-Step-Dec). Given an LTL formula ϕ and a positive (or negative) polarity occurrence of a subformula ψ, answer “yes” if ϕ with ψ set to 1 (or 0) is an unrealizable core of ϕ via syntax tree, “no” otherwise.

Definition 45 (Iuc-Dec-ST). Given an LTL formula ϕ, answer “yes” if ϕ is an IUC via syntax tree, “no” otherwise.

Definition 46 (Iuc-Dec-dCNF). Given an LTL formula ϕ, answer “yes” if ϕ is an IUC via dCNF, “no” otherwise.

Definition 47 (Irc-Dec). Given an LTL formula ϕ, answer “yes” if ϕ is an irreducible unrealizable core via syntax tree, “no” otherwise.

Theorem 48 (Complexity).

1. Iuc-Search-ST $\in$ $\mathsf{FP}^{\mathsf{PSPACE}}$.
2. Iuc-Step-Dec-ST $\in$ PSPACE-*complete.*
3. Iuc-Dec-ST $\in$ PSPACE.
4. Iuc-Search-dCNF $\in$ $\mathsf{FP}^{\mathsf{PSPACE}}$.
5. Iuc-Step-Dec-dCNF $\in$ PSPACE-*complete.*
6. Iuc-Dec-dCNF $\in$ PSPACE.
7. Irc-Search $\in$ FP^{2EXPTIME}.
8. Irc-Step-Dec $\in$ 2EXPTIME-*complete.*
9. Irc-Dec $\in$ 2EXPTIME.

Proof. We only prove 1-3. 4-9 are analogous.

We start with 2. Membership is immediate by checking unsatisfiability of the weakened variant of ϕ. Hardness for 2 is by a reduction from the set of unsatisfiable LTL formulas: Given an LTL formula ϕ', let $\phi'' \equiv \phi' \wedge 0$. Then ϕ' is unsatisfiable iff $(\phi'', 0)$ is in Iuc-Step-Dec-ST.

For 3 it’s easy to see that it is sufficient to check unsatisfiability ϕ and, if that is the case, for each occurrence of a subformula of ϕ weaken that occurrence separately and check that the result is satisfiable. This leads to a number of satisfiability checks that is linear in the size of ϕ with each checked formula being at most as long as ϕ.

For 1 note that there is a naive algorithm for Iuc-Search-ST as follows. First check unsatisfiability of ϕ. If the answer is “no”, then the algorithm stops. Otherwise, it weakens a child of the root node in the syntax tree of ϕ and determines unsatisfiability. If the resulting formula is still unsatisfiable, the weakening is made permanent; if not, the weakening is undone and the algorithm continues recursively into the children (if any) of the current node. This is repeated for all children of the root node. Clearly this algorithm performs a number of satisfiability checks that is linear in the size of the formula with each checked formula being at most as long as ϕ.

We remark that none of [AFF^{+}03, FKSFV08] provide lower bounds matching the respective upper bounds for the problems similar to Iuc-Dec-ST and Irc-Dec for LTL. Also [CS09] mostly provides no matching lower bounds for LTL.

11. Related Work

Notions of a 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, a 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, a 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. [KHB09] only reduces guarantees but proceeds inside the remaining guarantees by also removing output signals.

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

[TCJ08, Tor09] suggest 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.

Complexity

Apart from [FKSFV08, KV03, BBDER01, AFF$^+$03, BFG$^+$05, GC04a] mentioned in Sect. 10 the following works also contain results on complexity. [GC04b] discusses complexity of finding all sets of mutually vacuous subformulas. [CS09] considers a larger set of problems in mutual vacuity including optimization problems; in addition, complexity results are stated for finding a smallest subset of a set of formulas that implies the original set. For complexity results for other notions or applications of vacuity see also [BK08b, CGS08]. Going beyond vacuity, [PW85] establishes that the problem corresponding to Iuc-Dec-dCNF for 3SAT CNF is D^P-complete.

12. 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 extended some of the notions to unrealizable cores.

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 a 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 a UC suggested at the end of Sections 6 and 7, investigating the notion of a UC that results from temporal resolution proofs, and taking sharing of subformulas into account. 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 reviewers for their valuable comments. The author also thanks the research groups at FBK-irst and Verimag for helpful discussions, comments, and support, in particular, A. Cimatti, A. Ferrante, A. Mariotti, 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).

[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. 32, 33, 34, 35

[AL93] M. Abadi and L. Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, 1993. Links: ee, Google Scholar. 27

[ala] Antichains.be — The Alaska Tool — Experimental Results. `http://www.antichains.be/alaska/experiments.html`. 21, 22

[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. 1, 26, 35

[AT04] R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004. Links: ee, Google Scholar. 26

[BB94] D. Beatty and R. Bryant. Formally verifying a microprocessor using a simulation methodology. In *DAC*, pages 596–602, 1994. Links: ee, Google Scholar. 32

[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. 1, 32, 33, 34, 35

[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. 14

[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. 2, 11, 14

[BCG+10] R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Könighofer, M. Roveri, V. Schuppan, and R. Seeber. RATSY - a new requirements analysis tool with synthesis. In T. Touili, B. Cook, and P. Jackson, editors, *CAV*, volume 6174 of *LNCS*, pages 425–429. Springer, 2010. Links: ee, Google Scholar. 1

[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. 17

[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. 1

[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. 14

[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. 35

[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. 32, 34, 35

[BGJ+07a] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Automatic hardware synthesis from specifications: a case study. In R. Lauwereins and J. Madsen, editors, *DATE*, pages 1188–1193. ACM, 2007. Links: ee, Google Scholar. 26

[BGJ$^+$07b] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, compile, run: Hardware from PSL. In S. Glesner, J. Knoop, and R. Drechsler, editors, *COCV*, volume 190(4) of *ENTCS*, pages 3–16. Elsevier, 2007. Links: ee, Google Scholar. 26

[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. 14

[BK08a] C. Baier and J. Katoen. *Principles of Model Checking*. MIT Press, 2008. Links: Google Scholar. 1, 2, 3

[BK08b] T. Ball and O. Kupferman. Vacuity in testing. In B. Beckert and R. Hähnle, editors, *TAP*, volume 4966 of *LNCS*, pages 4–17. Springer, 2008. Links: ee, Google Scholar. 35

[BL69] J. Büchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, April 1969. Links: Google Scholar. 26

[Boy92] T. Boy de la Tour. An optimality result for clause form translation. *J. Symb. Comput.*, 14(4):283–302, 1992. Links: Google Scholar. 5

[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. 35

[BSS$^+$09] M. Bauland, T. Schneider, H. Schnoor, I. Schnoor, and H. Vollmer. The complexity of generalized satisfiability for linear temporal logic. *Logical Methods in Computer Science*, 5(1), 2009. Links: ee, Google Scholar. 3

[CCM$^+$09] R. Cavada, A. Cimatti, A. Mariotti, C. Mattarei, A. Micheli, S. Mover, M. Pensallorto, M. Roveri, A. Susi, and S. Tonetta. Supporting requirements validation: The EuRailCheck tool. In *ASE*, pages 665–667. IEEE Computer Society, 2009. Links: ee, Google Scholar. 1

[CCM$^+$10] A. Chiappini, A. Cimatti, L. Macchi, O. Rebollo, M. Roveri, A. Susi, S. Tonetta, and B. Vittorini. Formalization and validation of a subset of the european train control system. In J. Kramer, J. Bishop, P. Devanbu, and S. Uchitel, editors, *ICSE (2)*, pages 109–118. ACM, 2010. Links: ee, Google Scholar. 1

[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. 35

[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. 1

[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. 17

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999. Links: Google Scholar. 1

[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. 3, 35

[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. 33, 35

[Chu63] A. Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians, Stockholm, Sweden, 1962*, pages 23–35. Institut Mittag-Leffler, 1963. Links: Google Scholar. 26

[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. 14

[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. 1

[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. 11, 14

[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. 2, 3, 14, 35

[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. 2, 26, 28, 35

[CRST08b] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. From informal requirements to property-driven formal validation. In D. Cofer and A. Fantechi, editors, *FMICS*, volume 5596 of *LNCS*, pages 166–181. Springer, 2008. Links: ee, Google Scholar. 1

[CRST08c] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Object models with temporal constraints. In A. Cerone and S. Gruner, editors, *SEFM*, pages 249–258. IEEE Computer Society, 2008. Links: ee, Google Scholar. 1

[CRT09] A. Cimatti, M. Roveri, and S. Tonetta. Requirements validation for hybrid systems. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *LNCS*, pages 188–203. Springer, 2009. Links: ee, Google Scholar. 1

[CS07] H. Chockler and O. Strichman. Easier and more informative vacuity checks. In *MEMOCODE*, pages 189–198. IEEE, 2007. Links: ee, Google Scholar. 32

[CS09] H. Chockler and O. Strichman. Before and after vacuity. *Formal Methods in System Design*, 34(1):37–58, 2009. Links: ee, Google Scholar. 1, 32, 33, 34, 35

[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. 2

[DGV99] M. Daniele, F. Giunchiglia, and M. Vardi. Improved automata generation for linear temporal logic. In N. Halbwachs and D. Peled, editors, *CAV*, volume 1633 of *LNCS*, pages 249–260. Springer, 1999. Links: ee, Google Scholar. 23

[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. 35

[EC82] E. Emerson and E. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982. Links: Google Scholar. 26

[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. 1, 2, 27

[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. 5

[ES03] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. In O. Strichman and A. Biere, editors, *BMC*, volume 89(4) of *ENTCS*, pages 543–560. Elsevier, 2003. Links: ee, Google Scholar. 14

[eur] Formal verification of ETCS specifications. `http://es.fbk.eu/events/formal-etcs/`. 1

[FDP01] M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Trans. Comput. Log.*, 2(1):12–56, 2001. Links: ee, Google Scholar. 11, 14

[Fis91] M. Fisher. A resolution method for temporal logic. In *IJCAI*, pages 99–104, 1991. Links: Google Scholar. 6, 11

[FKSFV08] D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M. Vardi. A framework for inherent vacuity. In H. Chockler and A. Hu, editors, *HVC*, volume 5394 of *LNCS*, pages 7–22. Springer, 2008. Links: ee, Google Scholar. 1, 32, 33, 34, 35

[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`. 11

[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. 11

[GBJV08] K. Greimel, R. Bloem, B. Jobstmann, and M. Vardi. Open implication. In L. Aceto, I. Damgård, L. Goldberg, M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP (2)*, volume 5126 of *LNCS*, pages 361–372. Springer, 2008. Links: ee, Google Scholar. 27

[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. 32, 34, 35

[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. 32, 33, 35

[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. 35

[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. 3, 35

[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. 2, 17, 18, 35

[Gre07] K. Greimel. Open implication: A new relation on specifications given in linear temporal logic. Master's thesis, Graz University of Technology, 2007. Links: Google Scholar. 27

[Har05] A. Harding. *Symbolic Strategy Synthesis For Games With LTL Winning Conditions*. PhD thesis, University of Birmingham, 2005. Links: Google Scholar. 21, 22

[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. 15

[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`. 11

[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. 11

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In K. Apt, editor, *Logics and models of concurrent systems*, volume F 13 of *NATO ASI Series, Computer And System Sciences*, pages 477–498. Springer, 1985. Links: Google Scholar. 26

[JB06] B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *FMCAD*, pages 117–124. IEEE Computer Society, 2006. Links: ee, Google Scholar. 26

[KHB09] R. Könighofer, G. Hofferek, and R. Bloem. Debugging formal specifications using simple counterstrategies. In *FMCAD*, pages 152–159. IEEE, 2009. Links: ee, Google Scholar. 2, 26, 28, 35

[KPP09] H. Kugler, C. Plock, and A. Pnueli. Controller synthesis from LSC requirements. In M. Chechik and M. Wirsing, editors, *FASE*, volume 5503 of *LNCS*, pages 79–93. Springer, 2009. Links: ee, Google Scholar. 26

[KS08] D. Kroening and O. Strichman. *Decision Procedures*. Springer, 2008. Links: Google Scholar. 14, 35

[KS09] H. Kugler and I. Segall. Compositional synthesis of reactive systems from live sequence chart specifications. In S. Kowalewski and A. Philippou, editors, *TACAS*, volume 5505 of *LNCS*, pages 77–91. Springer, 2009. Links: ee, Google Scholar. 26

[KV03] O. Kupferman and M. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2):224–233, 2003. Links: ee, Google Scholar. 1, 4, 32, 33, 34, 35

[KV05] O. Kupferman and M. Vardi. Safraless decision procedures. In *FOCS*, pages 531–542. IEEE Computer Society, 2005. Links: ee, Google Scholar. 26

[LH09] M. Ludwig and U. Hustadt. Resolution-based model construction for PLTL. In C. Lutz and J. Raskin, editors, *TIME*, pages 73–80. IEEE Computer Society, 2009. Links: ee, Google Scholar. 3

[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. 19

[Mar04] N. Markey. Past is for free: on the complexity of verifying linear temporal properties with past. *Acta Inf.*, 40(6-7):431–458, 2004. Links: ee, Google Scholar. 3

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 6(1):68–93, 1984. Links: ee, Google Scholar. 26

[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. 35

[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. 35

[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. 14

[PG86] D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304, 1986. Links: Google Scholar. 2, 5

[Pnu77] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977. Links: Google Scholar. 1, 2

[PPS06] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In E. Emerson and K. Namjoshi, editors, *VMCAI*, volume 3855 of *LNCS*, pages 364–380. Springer, 2006. Links: ee, Google Scholar. 2, 26, 27

[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. 35

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. Links: Google Scholar. 1, 26, 35

[pro] Prosyd. `http://www.prosyd.org/`. 1

[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. 32

[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. 1, 35

[PW85] C. Papadimitriou and D. Wolfe. The complexity of facets resolved. In *FOCS*, pages 74–78. IEEE, 1985. Links: Google Scholar. 35

[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. 1

[Rab72] M. Rabin. Automata on infinite objects and church's problem. *Regional Conference Series in Mathematics*, 13, 1972. Links: Google Scholar. 26

[Ros92] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992. Links: Google Scholar. 26

[roz] Model checking benchmarking scripts. `http://ti.arc.nasa.gov/m/profile/kyrozier/benchmarking_scripts/benchmarking_scripts.html`. 21, 23, 25

[RV10] K. Rozier and M. Vardi. LTL satisfiability checking. *STTT*, 12(2):123–137, May 2010. Links: ee, Google Scholar. 3, 21, 23, 25

[Saf88] S. Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327. IEEE, 1988. Links: Google Scholar. 26

[SC85] A. Sistla and E. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. Links: ee, Google Scholar. 3

[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. 35

[Sch09a] V. Schuppan. Towards a notion of unsatisfiable cores for LTL. In F. Arbab and M. Sirjani, editors, *FSEN*, volume 5961 of *LNCS*, pages 129–145. Springer, 2009. Links: ee, Google Scholar. 1

[Sch09b] V. Schuppan. Towards a notion of unsatisfiable cores for LTL. Technical Report 200901000, Fondazione Bruno Kessler, 2009. Available from `http://www.schuppan.de/viktor/VSchuppan-FSEN-2009-full.pdf`. 1

[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. 33, 35

[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. 35

[SSR08] S. Sohail, F. Somenzi, and K. Ravi. A hybrid algorithm for LTL games. In F. Logozzo, D. Peled, and L. Zuck, editors, *VMCAI*, volume 4905 of *LNCS*, pages 309–323. Springer, 2008. Links: ee, Google Scholar. 26

[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. 14

[SV04] M. Samer and H. Veith. Parameterized vacuity. In A. Hu and A. Martin, editors, *FMCAD*, volume 3312 of *LNCS*, pages 322–336. Springer, 2004. Links: ee, Google Scholar. 32

[SV07] M. Samer and H. Veith. On the notion of vacuous truth. In N. Dershowitz and A. Voronkov, editors, *LPAR*, volume 4790 of *LNCS*, pages 2–14. Springer, 2007. Links: ee, Google Scholar. 32, 33

[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. 3, 35

[Tor09] E. Torlak. *A Constraint Solver for Software Engineering: Finding Models and Cores of Large Relational Specifications*. PhD thesis, Massachusetts Institute of Technology, 2009. Links: ee, Google Scholar. 35

[WDMR08] M. De Wulf, L. Doyen, N. Maquet, and J. Raskin. Antichains: Alternative algorithms for LTL satisfiability and model-checking. In C. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *LNCS*, pages 63–77. Springer, 2008. Links: ee, Google Scholar. 3, 21, 22

[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. 35

[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. 2

[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. 3, 35

[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`. 3, 35