On the Construction of Monitors for Temporal Logic Properties

M.C.W. Geilen [1]

Section of Information and Communication Systems
Faculty of Electrical Engineering
Eindhoven University of Technology
Eindhoven, The Netherlands

Abstract

Temporal logic is a valuable tool for specifying correctness properties of reactive programs. With the advent of temporal logic model checkers, it has become an important aid for the verification of concurrent and reactive systems. In model checking the temporal logic properties are verified against models expressed in the tool's modelling language. In addition, model-checking techniques are useful to test actual implementations or to verify models of the system that are too detailed to be analysed by a model checker, by means of, for instance, simulation.

A tableau construction is an algorithm that translates a temporal logic formula into a finite-state automaton that accepts precisely all the models of the formula. It is a key ingredient to checking satisfiability of a formula as well as to the automata-theoretic approach to model checking. An improvement to the efficiency of tableau constructions has been the development of on-the-fly versions.

In this paper, we present a particular tableau construction for the incremental analysis of execution traces during test, simulation or model-checking. The automaton forms the basis of a monitor that detects both good and bad prefix of a particular kind, namely those that are informative for the property under investigation. We elaborate on the construction of the monitor and demonstrate its correctness.

1 Introduction

Temporal logic, introduced in [11], is a popular formalism to express dynamic properties of reactive and concurrent systems. When the (abstraction of the) system is finite-state, model checking procedures can be used to verify its correctness automatically. A *tableau construction* is an algorithm that translates a temporal logic formula into a finite-state automaton (possibly on infinite

[1] Email: M.C.W.Geilen@tue.nl

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

words) that accepts precisely all the models of the formula. The *automata-theoretic* approach to model checking ([10,13]) relies on tableau algorithms to turn a temporal formula into an observer of a model's behaviours. Driven by practical needs, tableau constructions are being continuously improved and reimplemented (e.g. [7,3,5]). One such improvement has been the development of *on-the-fly* versions of tableau constructions. In general this means that the tableau automaton is constructed in a lazy way, generating states and transitions as they are needed.

Model-checking has gained a reputation for automatic verification of the correctness of (models of) real-life systems. At the same time it is recognised that similar techniques can be applied in other ways as well. One may use them not only for the verification of the formal abstract models, but also for actual software implementations or detailed simulation models and analyse their behaviour for the desired correctness properties during run-time. One particular reason to do so is to counter the effects of the state-space explosion, that makes that traditional verification techniques do not scale up well. An important aspect of traditional model-checkers is a systematic search through a system's state space. During the verification of a running system, this control over the state-space exploration is not available. Backtracking is impossible or extremely costly. Therefore monitors for the analysis of the behaviour exposed by the running system, must be able to analyse the behaviour incrementally and deterministically. Moreover, as cycles go undetected, properties cannot be inferred directly about infinite traces. For this reason, run-time model checking requires modifications to the verification approach. Such modifications are discussed in this paper.

Contribution of this paper

In this paper we present the (automatic) construction of run-time monitors for properties expressed in linear temporal logic. These monitors allow the (simultaneous) detection of both (informative) good and bad prefixes of an execution and can thus serve to monitor temporal logic properties incrementally and deterministically at run-time. We show that the transition systems belonging to a tableau automaton on infinite state sequences, the finite state automaton for informative good prefixes and the finite state automaton for informative bad prefixes (almost) coincide. The automata differ only in acceptance conditions and we show how they can be combined into a single monitor.

Related Work

This work builds on the work of Kupferman and Vardi [9]. Whereas their main objective is to simplify the model-checking procedure for safety properties using alternating automata, we study the use of their notion of informative prefixes for the construction of tableau automata and run-time monitors in particular. We focus on the construction of finite state and ultimately deterministic finite state automata. [9] also elaborates on the classification of prefixes and complexity results. Other related work includes the the papers [2,8] which give a more pragmatic treatment of run-time temporal logic verification. In [2], the basic unfolding principle of the construction of a tableau automaton is used, the main disadvantage is that formulas are manipulated directly during simulation, which may not be very efficient. Also in [8], the observation of LTL properties in simulations of System-C descriptions is discussed. Formulas are interpreted over finite state sequences and given a three-valued interpretation. Work on on-the-fly tableau constructions includes [7,4,3].

Overview of the paper

The paper is structured as follows. Section 2 introduces some general preliminaries and informative prefixes in particular. In section 3, we discuss a normal form, based on the notion of informativeness that will form the heart of the tableau constructions. The tableau construction itself is discussed in section 4. How to make run-time monitors from these tableaux is the topic of section 5, where it is also shown to be correct. Section 6 concludes.

2 Preliminaries

Finite and Infinite Words

A *word* $\bar{w} = \alpha_0\alpha_1\alpha_2 \dots \alpha_{n-1}$ (of length n) over an alphabet Σ is a sequence of symbols from Σ; An infinite word (ω-word) $\bar{w} = \alpha_0\alpha_1\alpha_2 \dots$ over an alphabet Σ is an infinite sequence of symbols from Σ; $\bar{w}(k)$ denotes α_k and $\bar{w}^k$ refers to the *tail* $\alpha_k\alpha_{k+1}\alpha_{k+2}\dots$. We use the latter notations for other kinds of sequences as well. The concatenation of a finite word $\bar{w}_1$ and a finite or infinite word $\bar{w}_2$ is denoted as $\bar{w}_1 \cdot \bar{w}_2$. A finite word $\bar{w}_1$ is said to be a *prefix* of a finite or infinite word $\bar{w}_2$ if there is some word $\bar{w}_3$ such that $\bar{w}_2 = \bar{w}_1 \cdot \bar{w}_3$. For a finite word $\bar{w}$, $|\bar{w}|$ denotes the number of symbols in the word. For an infinite word $\bar{w}$ over Σ, $\inf(\bar{w})$ denotes the symbols of Σ that occur infinitely often in $\bar{w}$. A set of words is called a *language*.

Finite State Automata

Let alphabet Σ be a set of symbols. A *labelled transition system* $L = \langle Q, Q_0, V, \delta\rangle$ over Σ consists of a finite set Q of *locations*; a finite set $Q_0 \subseteq Q$ of *initial locations*; a mapping $V : Q \to 2^\Sigma$ labelling every location with a set of symbols from the alphabet and a set $\delta \subseteq Q \times Q$ of *edges*. A *run* describes a path through the transition system. It provides the location of the transition system at any moment, by recording the sequence of locations. A run of a labelled transition system $L = \langle Q, Q_0, V, \delta\rangle$ is a (finite or infinite) sequence $\bar{q}$ of locations $\bar{q}(k) \in Q$ such that for all $k \geq 0$ (and $k < |\bar{q}|-1$ if $\bar{q}$ is finite), there is an edge $(\bar{q}(k), \bar{q}(k+1)) \in \delta$. In this case we also say that $\bar{q}$ is a run *from* location $\bar{q}(0)$, or a $\bar{q}(0)$-*run* for short. A run $\bar{q}$ is called *initial* if $\bar{q}(0) \in Q_0$.

Given a word $\bar{w}$ and a run $\bar{q}$ of equal length, $\bar{q}$ is a run *for* $\bar{w}$ (or $\bar{w}$ matches $\bar{q}$) if [2] for all $k \geq 0$, $\bar{w}(k) \in V(\bar{q}(k))$.

A *finite state automaton* $A = \langle Q, Q_0, V, \delta, f\rangle$ over Σ consists of a labelled transitions system over Σ and a set f of *final locations*. Automaton A *accepts* a *finite* word $\bar{w}$ (of length n) if it has an initial run $\bar{q}$ for $\bar{w}$ ending in a final location ($\bar{q}(n-1) \in f$). A *(generalised) Büchi automaton* $A = \langle Q, Q_0, V, \delta, F\rangle$ over Σ on the other hand is an automaton on infinite words and consists of a labelled transition system over Σ and a set F of *acceptance sets* $f \subseteq Q$. A generalised Büchi automaton A *accepts* an infinite word $\bar{w}$ if it has an initial run $\bar{q}$ for $\bar{w}$ such that for every $f \in F$, $\inf(\bar{q}) \cap f \neq \varnothing$. For a finite state automaton or Büchi automaton A, the *language* $\mathcal{L}(A)$ of A is the set of all words that it accepts.

Linear Temporal logic

We use the standard definition of Linear Temporal Logic and assume the existence of a finite set *Prop* of atomic propositions. The syntax of LTL is given by the following grammar ($p \in \mathit{Prop}$):

$$\psi ::= \text{true} \mid p \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \bigcirc\, \psi \mid \psi_1 \mathsf{U} \psi_2.$$

We let ψ, φ, ψ', φ', ψ_1, φ_1, ψ_2, φ_2, etcetera range over LTL. We use $cl(\varphi)$ to denote the subformula closure of φ. In the remainder we use the duals of the operators w.r.t. negation (false $= \neg$true, $\varphi_1 \wedge \varphi_2 = \neg((\neg\varphi_1) \vee (\neg\varphi_2))$ and $\varphi_1 \mathsf{V} \varphi_2 = \neg((\neg\varphi_1)\mathsf{U}(\neg\varphi_2)))$ to push negations inward until they occur only in front of atomic propositions, and write formulas in *positive normal form*. We shall identify formulas with the corresponding formulas in positive normal form [3]. Moreover, if Φ is a set of formulas, we write $\bigwedge \Phi$ to denote the conjunction of these formulas and we write $\bar{\sigma} \models \Phi$ to denote that $\bar{\sigma} \models \bigwedge \Phi$. The language P_φ of (infinite) state sequences that satisfy the formula φ is referred to as the *property* expressed by LTL formula φ.

Certain properties can be qualified as *safety* properties (stating that 'something bad will never happen') or *liveness* properties (stating that 'something good will eventually happen'). A property P is a liveness property if for every finite state sequence $\bar{\sigma}$ there exists some infinite state sequence $\bar{\sigma}'$ such that $\bar{\sigma} \cdot \bar{\sigma}' \in P$ (although other definitions are possible [1,12]). A property is a safety property if every infinite state sequence $\bar{\sigma} \notin P$, has a prefix $\bar{\sigma}'$ such that $\bar{\sigma}' \cdot \bar{\sigma}'' \notin P$ for every state sequence $\bar{\sigma}''$. The latter kind of prefix is called a *bad prefix*; a prefix $\bar{\sigma}$ is called a bad prefix for a property P if there is no state sequence $\bar{\sigma}'$ such that $\bar{\sigma} \cdot \bar{\sigma}' \in P$. A *good prefix* for a property P, on the other hand, is a prefix $\bar{\sigma}$ such that for every $\bar{\sigma}'$, $\bar{\sigma} \cdot \bar{\sigma}' \in P$ [9].

[2] As locations are labelled with *sets* of symbols, a single run corresponds in general to a set of words.
[3] Using $cl(\neg\psi) = \neg cl(\psi)$ rather than $cl(\neg\psi) = \neg\psi \cup cl(\psi)$ to make cl insensitive to a formula's representation.

Definition 2.1 [9] A finite word $\bar{u} \in \Sigma^*$ is called a *good prefix for the language* $\mathcal{L} \subseteq \Sigma^\omega$ iff for every infinite word $\bar{w} \in \Sigma^\omega$, $\bar{u} \cdot \bar{w} \in \mathcal{L}$. Similarly, $\bar{u}$ is called a *bad prefix for the language* $\mathcal{L}$ iff for every infinite word $\bar{w} \in \Sigma^\omega$, $\bar{u} \cdot \bar{w} \notin \mathcal{L}$. $\Box$

This paper deals with the verification of safety properties expressed by LTL formulas, however, not all safety formulas are alike. In [9], safety formulas are classified into three kinds, the *intentionally safe*, the *accidentally safe* and the *pathologically safe*, depending on the kinds of prefixes their properties possess. A prefix $\bar{\sigma}$ is called *informative* for a formula if it "tells the whole story"[9] of why the formula holds for every infinite state sequence of which $\bar{\sigma}$ is a prefix. This is made precise below. Intentionally safe formulas are formulas of which every bad prefix is informative (e.g. $\Box p$), an accidentally safe formula is a safety formula that is not intentionally safe, but of which all state sequences that violate it, do have some informative bad prefix (e.g. $\Box(p \vee (\bigcirc q \wedge \bigcirc \neg q)))$. Pathologically safe safety formulas are formulas that have computations that violate it without any informative bad prefix (e.g. $((\Box(q \vee \Box\Diamond p)) \wedge (\Box(r \vee \Box\Diamond\neg p))) \vee \Box q \vee \Box r$, examples from [9]).

A set of formulas is said to be *locally informative* if it is 'informative' in the sense that every compound formula in the set is supported by one or more of its direct subformulas. Together the formulas constitute an explanation *why* a requirement will hold. If a set contains the formula $\varphi_1 \wedge \varphi_2$, then it must also contain both φ_1 and φ_2 to demonstrate this. Similarly if a set contains $\varphi_1 \mathsf{U} \varphi_2$ then it must contain φ_1 or φ_2 as well (this only pertains to the current state, not containing φ_2 leads to extra constraints on the formulas that hold at the following moment). In the remainder of the paper we let Φ range over sets of LTL formulas.

Definition 2.2 A set Φ of formulas is *locally informative* if

- false $\notin \Phi$;
- if $\varphi_1 \vee \varphi_2 \in \Phi$ then $\varphi_1 \in \Phi$ or $\varphi_2 \in \Phi$;
- if $\varphi_1 \wedge \varphi_2 \in \Phi$ then $\varphi_1 \in \Phi$ and $\varphi_2 \in \Phi$;
- if $\varphi_1 \mathsf{U} \varphi_2 \in \Phi$ then $\varphi_1 \in \Phi$ or $\varphi_2 \in \Phi$;
- if $\varphi_1 \mathsf{V} \varphi_2 \in \Phi$ then $\varphi_2 \in \Phi$.

Local informativeness constrains the formulas that are required to hold for a particular state sequence. In the case of Until or Release operators however, constraints may also need to be imposed on the remainder of the state sequence (for instance if the set contains $\varphi_1 \mathsf{U} \varphi_2$ and φ_1, but not φ_2). If the truth of an Until or Release formula follows directly from the other formulas in the set, then such a set is said to be trivial for that Until or Release formula (if the set contains both $\varphi_1 \mathsf{U} \varphi_2$ and φ_2, or both $\varphi_1 \mathsf{V} \varphi_2$ and φ_1). It is said to be non-trivial otherwise. (Non-)trivial sets will play an important role in the tableau constructions, because they pose constraints on the remainder of the state sequence, and thus determine '*temporal informative successors*'.

Definition 2.3 A set Φ of formulas is non-trivial for

- the Until formula $\varphi_1 \mathsf{U} \varphi_2$, if $\varphi_1 \mathsf{U} \varphi_2 \in \Phi$ and $\varphi_2 \notin \Phi$, let $Next(\varphi_1 \mathsf{U} \varphi_2) = \varphi_1 \mathsf{U} \varphi_2$;
- the Release formula $\varphi_1 \mathsf{V} \varphi_2$, if $\varphi_1 \mathsf{V} \varphi_2 \in \Phi$ and $\varphi_1 \notin \Phi$, let $Next(\varphi_1 \mathsf{V} \varphi_2) = \varphi_1 \mathsf{V} \varphi_2$;
- the formula $\bigcirc\varphi$, if $\bigcirc\varphi \in \Phi$, let $Next(\bigcirc\varphi) = \varphi$. □

A set Φ' of formulas is a temporally informative successor of the set Φ of formulas if for every formula ψ such that Φ is non-trivial for ψ, Φ' contains $Next(\psi)$. Another way to formulate temporal informativeness, is to say that for Φ' to be a temporally informative successor of Φ, it must contain at least certain formulas that are determined by Φ. This is captured by the following definition.

Definition 2.4 Let Φ be a set of formulas. Then the set $Next(\Phi)$ of *temporal informativeness constraints* is the set :

$$\{Next(\psi) \mid \psi \in \Phi \text{ such that } \Phi \text{ is non-trivial for } \psi\}.$$

Φ' is a *temporally informative successor* of Φ if $Next(\Phi) \subseteq \Phi'$. This is denoted as $\Phi \to \Phi'$. □

In some of the proofs we use $Next(\Phi_1, \Phi_2)$ to denote $\{Next(\psi) \mid \psi \in \Phi_2$ such that $\Phi_1 \cup \Phi_2$ is non-trivial for $\psi\}$. We have, for instance, that $\{p\mathsf{U}q, p\} \to \{p\mathsf{U}q, q\}$ and $\{p\mathsf{U}q, q\} \to \varnothing$, but not $\{p\mathsf{U}q, p\} \to \{p\}$ and not $\{\bigcirc q\} \to \{p\}$. We can now define the notion of an *informative* good (bad) prefix.

Definition 2.5 ([9] [4]) Let $\bar{\tau}$ be a finite state sequence. $\bar{\tau}$ is informative for φ iff there exists a finite sequence $\overline{IS} \in (2^{\mathrm{LTL}})^*$ of sets of formulas, say of length $n + 1 \leq |\bar{\tau}| + 1$, such that

- $\varphi \in \overline{IS}(0)$;
- $\overline{IS}(n) = \varnothing$;
- for all $0 \leq i < n$ and $\psi \in \overline{IS}(i)$,
  - · if ψ is an atomic proposition p, then $p \in \bar{\tau}(i)$;
  - · if ψ is a negated atomic proposition $\neg p$, then $p \notin \bar{\tau}(i)$;
  - · $\overline{IS}(i)$ is locally informative;
  - · $\overline{IS}(i+1)$ is a temporally informative successor of $\overline{IS}(i)$. □

We call such a sequence $\overline{IS}$ an *informative sequence*. If such an informative sequence exists, it tells us why φ holds for any extension of the prefix $\bar{\tau}$. It indicates what formulas hold at what moment of the prefix and why. Since $\overline{IS}(i)$ is at some point empty, this reasoning is complete and thus applies to any extension of the prefix. For instance, if $\psi_1 \vee \psi_2 \in \overline{IS}(i)$, then by the

[4] we rephrase the definition of [9] in terms of our notions of local and temporal informativeness.

informativeness requirements, $\psi_1 \in \overline{IS}(i)$ or $\psi_2 \in \overline{IS}(i)$, which tells us that $\psi_1 \vee \psi_2$ holds for any extension of $\bar{\tau}^i$ (the part of $\bar{\tau}$ from state i to the end) since at least one of ψ_1 and ψ_2 holds for any extension of $\bar{\tau}^i$. If $\psi_1 \mathsf{U} \psi_2 \in \overline{IS}(i)$, $\psi_1 \in \overline{IS}(i)$, and $\psi_2 \notin \overline{IS}(i)$, then according to temporal informativeness, $\psi_1 \mathsf{U} \psi_2 \in \overline{IS}(i+1)$. This signifies that $\psi_1 \mathsf{U} \psi_2$ must hold for any extension of $\bar{\tau}^i$, because ψ_1 holds for any extension of $\bar{\tau}^i$ and $\psi_1 \mathsf{U} \psi_2$ holds for any extension of $\bar{\tau}^{i+1}$. Since $\overline{IS}(n) = \varnothing$, such a reasoning does not depend on any part of the state sequence beyond position n. It is complete and "tells the whole story"[9]. Thus, $\bar{\tau}$ is an informative good prefix for φ if it is informative for φ and $\bar{\tau}$ is an informative bad prefix for φ if it is informative for $\neg\varphi$.

3 Informative Normal Form

(On-the-fly) tableau constructions for linear temporal logic are often introduced using a rewriting procedure that rewrites formulas into 'disjunctive temporal normal form' in order to separate constraints on the current state from constraints upon the rest of the state sequence [7,4,3]. In this paper we introduce an on-the-fly tableau construction based on informativeness. Notice that although this construction is not identical, it closely resembles such constructions.

In correspondence with the disjunctive temporal normal form of traditional on-the-fly tableau constructions we define an 'informative normal form'.

Definition 3.1 A set Θ of sets of LTL formulas is in *informative normal form* if every set in Θ is locally informative.

We now introduce a number of rewrite rules, that transform any set into normal form. In the rewriting rules we represent the set of sets of formulas as a set of pairs $\langle New, Old\rangle$ (we call them terms) of sets of formulas, in order to discriminate the formulas that have been processed (*Old*) from the formulas that still need to be processed (*New*). The rules are presented in figure 1, which is interpreted as follows. Consider a set $\Theta \cup \{\langle New \cup \{\psi\}, Old\rangle\}$ of terms. The row in the table in which the Case field coincides with the shape of the LTL formula ψ determines how the set is rewritten.

Definition 3.2 The *(informative) normal form procedure* starts with a set Φ of formulas. It maintains a set Θ_n of terms $\langle New, Old\rangle$ that is initialised to $\Theta_0 = \{\langle \Phi, \varnothing\rangle\}$. Then as long as some reduction rule of table 1 applies, a rule is applied to Θ_n to obtain Θ_{n+1}. The procedure terminates when no more reduction rules apply to Θ_k for some $k \geq 0$. The result of the procedure is the set $\{Old \mid \langle \varnothing, Old\rangle \in \Theta_k\}$.

It is easy to show that the procedure terminates and that all terms in Θ_k are then of the form $\langle \varnothing, \Phi_i\rangle$ for some set Φ_i of formulas. Depending on the order in which terms from Θ and formulas from *New* are selected, different normal forms may be obtained. In the sequel, we assume the existence of a

	CASE	$\Theta \cup \{\langle New \cup \{\psi\}, Old\rangle\}$ REDUCES TO:
1	$\psi = \text{false}$	Θ
2	$\psi = \text{true}$	$\Theta \cup \{\langle New, Old \cup \{\psi\}\rangle\}$
3	$\psi = p$	$\Theta \cup \{\langle New, Old \cup \{\psi\}\rangle\}$
4	$\psi = \neg p$	$\Theta \cup \{\langle New, Old \cup \{\psi\}\rangle\}$
5	$\psi = \psi_1 \vee \psi_2$	$\Theta \cup \{\langle New \cup \{\psi_1\}, Old \cup \{\psi\}\rangle, \langle New \cup \{\psi_2\}, Old \cup \{\psi\}\rangle\}$
6	$\psi = \psi_1 \wedge \psi_2$	$\Theta \cup \{\langle New \cup \{\psi_1, \psi_2\}, Old \cup \{\psi\}\rangle\}$
7	$\psi = \bigcirc\psi'$	$\Theta \cup \{\langle New, Old \cup \{\psi\}\rangle\}$
8	$\psi = \psi_1 \mathsf{U} \psi_2$	$\Theta \cup \{\langle New \cup \{\psi_2\}, Old \cup \{\psi\}\rangle, \langle New \cup \{\psi_1\}, Old \cup \{\psi\}\rangle\}$
9	$\psi = \psi_1 \mathsf{V} \psi_2$	$\Theta \cup \{\langle New \cup \{\psi_1, \psi_2\}, Old \cup \{\psi\}\rangle, \langle New \cup \{\psi_2\}, Old \cup \{\psi\}\rangle\}$

Table 1
Local informativeness procedure

deterministic procedure *NF* that computes a particular normal form for any given set of formulas. We use $NF(\varphi)$ to denote $NF(\{\varphi\})$.

Lemma 3.3 *Let Φ be a set of* LTL *formulas. Then, $NF(\Phi)$ is in informative normal form and furthermore, if $\bar{\sigma}$ is a state sequence, such that $\bar{\sigma} \models \Phi$, then there exists a set $\Phi' \in NF(\Phi)$ such that (i) $\bar{\sigma} \models \Phi'$, (ii) $\bar{\sigma}^1 \models Next(\Phi')$ and (iii) for every Until formula $\psi = \varphi_1 \mathsf{U} \varphi_2 \in \Phi$ such that $\bar{\sigma} \models \varphi_2$, $\varphi_2 \in \Phi'$.*

Proof. The fact that Θ' is locally informative can be shown by an invariant on the sets Θ_n stating that the terms $\langle New, Old\rangle$ in Θ_n are locally informative w.r.t. the formulas in *Old*. (This means that the rules of local informativeness are interpreted as: 'false $\notin$ *Old*' and 'if $\psi \in Old$, then $\ldots \in Old \cup New$'.) When the procedure ends, all formulas are in the *Old* sets and the sets in $NF(\Phi)$ are locally informative. The second part is proved using an invariant saying that there exists a term $\langle New, Old\rangle \in \Theta_n$ such that (i) $\bar{\sigma} \models New \cup Old$, (ii) $\bar{\sigma}^1 \models Next(New, Old)$ and (iii) for every Until formula $\psi = \varphi_1 \mathsf{U} \varphi_2 \in Old$ such that $\bar{\sigma} \models \varphi_2$, $\varphi_2 \in \Phi'$. $\square$

Example

Consider the LTL formula $\Diamond p = \text{true}\mathsf{U}p$. In terms of the normal form procedure, the rewriting process of $\text{true}\mathsf{U}p$ proceeds as follows (we write $\Theta_1 \Rightarrow \Theta_2$ to express that Θ_2 is obtained from Θ_1 by one or more steps in the procedure).

$$\{\langle\{\text{true}\mathsf{U}p\}, \varnothing\rangle\} \Rightarrow$$

$$\{\langle\{p\}, \{\text{true}\mathsf{U}p, \}\rangle, \langle\{\text{true}\}, \{\text{true}\mathsf{U}p\}\rangle\} \Rightarrow$$

$$\{\langle\varnothing, \{\text{true}\mathsf{U}p, p\}\rangle, \langle\varnothing, \{\text{true}\mathsf{U}p, \text{true}\}\rangle\}$$

```
New := NF(φ), Q := ∅, Q₀ := New, δ := ∅
while New ≠ ∅ do
   Let Φ ∈ New
   New := New\{Φ}
   Q := Q ∪ {Φ}
   for every Φ' ∈ NF(Next(Φ)) do
       δ := δ ∪ {(Φ,Φ')}
       if Φ' ∉ Q then New := New ∪ {Φ'}
   od
od
```

Fig. 1. Algorithm for constructing locations and edges of the on-the-fly tableau automaton

The normal form suggests that there are two ways to demonstrate that true$\mathsf{U}p$ holds. Either demonstrate that p holds, or demonstrate that true holds (trivial) and (since $\mathit{Next}(\{\text{true}\mathsf{U}p, \text{true}\}) = \{\text{true}\mathsf{U}p\}$) that true$\mathsf{U}p$ holds at the next moment.

Complexity

One can show that the worst-case complexity of the normal form procedure $\mathit{NF}(\Phi)$ is $\mathcal{O}(2^n)$ where $n = \sum_{\psi\in\Phi} |\psi|$. Since at every step, $\sum_{\psi\in New} |\psi|$ decreases for the new terms that replace $\langle New, Old\rangle$ in the reduction and it is replaced by at most two new terms. If we further know that every $\psi \in \Phi$ is an element of $cl(\varphi)$ for some formula φ, then it follows that the complexity of NF is $\mathcal{O}(2^{|\varphi|^2})$. In that case however, a clever selection of the formula used for reduction (select the largest formulas first) reduces the complexity to $\mathcal{O}(2^{|\varphi|})$. This can be seen by considering that on any path leading from the initial term $\langle New, \varnothing\rangle$ to a final term $\langle\varnothing, Old\rangle$ every formula $\psi \in cl(\varphi)$ can be used for reduction at most once, hence such a path is of length at most $|\varphi|$ and the total number of reductions applied is $\mathcal{O}(2^{|\varphi|})$.

4 Tableau Construction

4.1 The tableau algorithm

The construction of a tableau automaton for an LTL formula φ, is based upon the normal form introduced in the previous section. The construction is closely related to the construction of [7]. Next formulas however are represented implicitly rather than explicitly. The number of formulas that may occur in the sets of the normal form terms is limited to syntactic subformulas of φ. The tableau automaton of an LTL formula φ is computed in the following way.

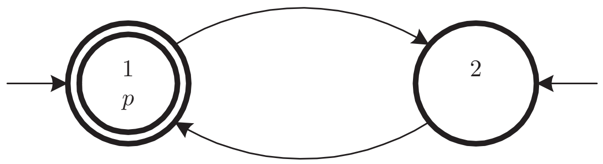

Fig. 2. Example tableau automaton of the formula $\Box\Diamond p$

Definition 4.1 Let φ be an LTL formula. The tableau automaton A_φ of φ is the automaton $\langle Q, Q_0, V, \delta, F\rangle$ over the alphabet 2^{Prop}, where

- The locations (Q), initial locations (Q_0) and transitions (δ) are computed by the procedure depicted in Figure 1. The locations $q \in Q$ are sets of LTL formulas;
- $V(q) = \{\alpha \in 2^{Prop} \mid \forall_{p\in Prop} p \in q \Rightarrow p \in \alpha, \neg p \in q \Rightarrow p \notin \alpha\}$. That is, a location q is labelled with all states that are consistent with the atomic propositions and the negated atomic propositions in q;
- F contains for every Until formula $\psi = \varphi_1 \mathsf{U} \varphi_2 \in cl(\varphi)$, a set $f_\psi = \{q \in Q \mid \psi \in q \Rightarrow \varphi_2 \in q\}$.

Example

If we take the formula $\Box\Diamond p = \text{false}\mathsf{V}(\text{true}\mathsf{U}p)$ and apply the tableau algorithm, we arrive at the automaton represented in Figure 2. Only the atomic propositions in the locations have been depicted. Location 1 is the set $\{\Box\Diamond p, \Diamond p, p\}$ and location 2 is the set $\{\Box\Diamond p, \Diamond p, \text{true}\}$. Initial locations are represented by a small arrow not originating from any location leading to the initial location. There is only one acceptance set $f_{\Diamond p}$, the locations of which are denoted with an extra circle around them.

Complexity

As all locations of the tableau automaton are subsets of $cl(\varphi)$, there are at most $2^{|\varphi|}$ different locations. For every location Φ, the normal form procedure is applied on $Next(\Phi)$. The procedure was shown to be $\mathcal{O}(2^{|\varphi|})$ in section 3. Thus the complexity of the tableau algorithm is $2^{\mathcal{O}(|\varphi|)}$.

4.2 Correctness

Here, we give a brief sketch of the proof that the tableau construction is correct, i.e. that for any LTL formula φ, the tableau automaton of φ accepts precisely those state sequences that satisfy φ. The algorithm based on informativeness constraints is very close to the algorithm of [7] and also the proof resembles those of [4,7,3].

Theorem 4.2 *Let φ be an* LTL *formula and let A_φ be the corresponding tableau automaton. Then for every state sequence $\bar{\sigma}$, A_φ accepts $\bar{\sigma}$ iff $\bar{\sigma} \models \varphi$.*

This theorem follows from soundness (every state sequence accepted by A_φ satisfies φ) and completeness (every state sequence satisfying φ is accepted by A_φ) of the construction as expressed by lemmas 4.4 and 4.7 below. In the remainder of this section, we assume that $A_\varphi = \langle Q, Q_0, V, \delta, F\rangle$ is the tableau automaton of the formula φ.

Soundness

We demonstrate that the automaton accepts only state sequences that satisfy φ. The main lemma is the following, claiming that any formula in a particular location is dealt with correctly.

Lemma 4.3 *Let $\bar{\sigma}$ be a state sequence, let $\bar{q}$ be a run of A_φ matching $\bar{\sigma}$ and let $\psi \in \bar{q}(0)$. Then $\bar{\sigma} \models \psi$.*

Proof. By induction on the structure of ψ. We only show the case related to the Until formula. If $\varphi_1 \mathsf{U} \varphi_2 \in \bar{q}(0)$, then it can be shown by the reduction of $\varphi_1 \mathsf{U} \varphi_2$ in the normal form procedure and by the construction of the automaton, that $\varphi_1 \mathsf{U} \varphi_2$ propagates at least until some location contains φ_2 (such a location is eventually reached since the run satisfies the acceptance condition related to $f_{\varphi_1 \mathsf{U} \varphi_2}$), by local informativeness, up to that point every locations contains φ_1. Thus, there is some k, such that $\varphi_2 \in \bar{q}(k)$ and for every $0 \leq m < k$, $\varphi_1 \in \bar{q}(m)$. By the induction hypothesis it follows that $\bar{\sigma} \models \varphi_1 \mathsf{U} \varphi_2$. □

One can furthermore easily show that every initial location contains the formula φ. From this and lemma 4.3, it follows immediately that every state sequence accepted by the tableau automaton A_φ satisfies φ.

Lemma 4.4 *If A_φ accepts the state sequence $\bar{\sigma}$, then $\bar{\sigma} \models \varphi$.*

Completeness

Here we demonstrate that every state sequence that satisfies φ is accepted by the tableau automaton. The normal form procedure guarantees that if a state sequence $\bar{\sigma}$ satisfies a formula ψ, then there is a term in the normal form of ψ, that is satisfied by $\bar{\sigma}$. Since the remainder of the state sequence satisfies the formulas in the corresponding *Next* set, there is a transition that can be taken by the automaton. This argument can be repeated to construct a run of the automaton for $\bar{\sigma}$. Moreover, one can show that the successor location can be chosen so as to satisfy the acceptance conditions.

The following lemma is the crux to the incremental construction of an accepting run for any state sequence $\bar{\sigma}$ that satisfies φ.

Lemma 4.5 *Let $q \in Q$ and let $\bar{\sigma}$ be a state sequence such that $\bar{\sigma} \models Next(q)$. Then there exists an edge $(q, q') \in \delta$ such that (i) $\bar{\sigma} \models q'$, (ii) $\bar{\sigma}^1 \models Next(q')$ and (iii) for every Until formula $\psi = \varphi_1 \mathsf{U} \varphi_2 \in Next(q)$ such that $\bar{\sigma} \models \varphi_2$, $q' \in f_\psi$.*

The lemma follows straightforwardly from lemma 3.3 and the construction of the tableau automaton. Similarly we can use lemma 3.3 to prove the following lemma that tells us how to select an appropriate initial location to start the construction of the run using the previous one.

Lemma 4.6 *Let $\bar{\sigma}$ be a state sequence such that $\bar{\sigma} \models \varphi$. Then there is some $q \in Q_0$ such that $\bar{\sigma} \models q$ and $\bar{\sigma}^1 \models Next(q)$.*

From lemma 4.6 and repeatedly applying lemma 4.5 to construct an accepting run, it follows that A_φ accepts all state sequences that satisfy φ.

Lemma 4.7 *If the state sequence $\bar{\sigma} \models \varphi$, then A_φ accepts $\bar{\sigma}$.*

5 Automata for Prefixes

In this section we discuss how the tableau method can be adapted to the analysis of prefixes of state sequences. It is possible to effectively construct an automaton on finite words that accepts all bad (good) prefixes for a given formula [9]. We concentrate however on automata that recognise informative prefixes only, for two reasons. Firstly, the construction of automata for all bad prefixes is doubly exponential in the length of the formula, whereas the construction of automata for informative prefixes is only singly exponential [9]. Secondly, the informative bad prefixes can be considered as the only proper counterexamples, since they demonstrate why the formula does not hold. Other bad prefixes depend on some peculiarity of the formula. For example, if ψ is a formula that is not satisfiable, then every finite state sequence is a bad prefix of the formula $\Diamond\psi$, but this finite state sequence itself provides no information why the formula does not hold.

The idea behind the construction is very simple. One creates the on-the-fly tableau automaton of the formula φ, but interprets it as an automaton on finite words. The original acceptance conditions can be forgotten, since they refer to infinite state sequences. The automaton's transition system however, has the following property. If a finite state sequence $\bar{\tau}$ is an informative bad prefix, then there is no finite run on the transition system that matches it. If on the other hand, it is an informative good prefix, then there is a run to the location $\varnothing$. To be precise, for any extension of the prefix, longer than the prefix itself, there is a matching run, the last location of which is $\varnothing$. As a consequence, if an automaton does not have a location $\varnothing$ then the formula does not have any informative good prefixes.

Definition 5.1 Let $A = \langle Q, Q_0, V, \delta, F \rangle$ be an ω-automaton over the alphabet Σ. Then $[A_\varphi]$ denotes the automaton $\langle Q, Q_0, V, \delta, Q \rangle$ on finite words over the same alphabet, i.e. the same automaton interpreted as a safety automaton (all locations are final) on finite words. $\langle A_\varphi \rangle$ denotes the automaton $\langle Q, Q_0, V, \delta, Q \cap \{\varnothing\} \rangle$ on finite words over Σ, i.e. the same automaton interpreted as an automaton on finite words with the location $\varnothing$ (if it exists) as its

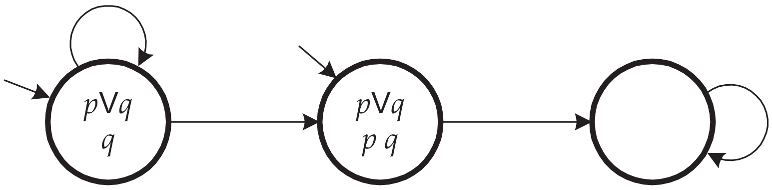

Fig. 3. Automaton for prefixes of the formula $p\vee q$

only final location.

Note that since the automata $[A_\varphi]$ and $\langle A_\varphi\rangle$ for non-bad and good prefixes respectively, are slight modifications to the Büchi tableau automaton, the complexity of their construction is the same, i.e. $2^{\mathcal{O}(|\varphi|)}$.

Example

Figure 3 shows the labelled transition system of the automaton $A_{p\vee q}$. The state sequence $\{q\}\{q\}\{p,q\}$ is an (informative) good prefix of $p\vee q$. The corresponding run to the location $\varnothing$ (the right location) is $\{p\vee q,q\}\{p\vee q,q\}$ $\{p\vee q,p,q\}\varnothing$. The run itself forms the informative sequence that establishes this. An informative bad prefix is $\{q\}\{p\}$. It can be verified that this sequence has no matching finite initial run on the transition system. A corresponding informative sequence demonstrating that the prefix is informative for $\neg(p\vee q)$ is $\{\neg(p\vee q),\neg p\}\{\neg(p\vee q),\neg q\}\varnothing$. The informative sequence can be interpreted as follows. It claims ($\neg(p\vee q)\in\overline{IS}(0)$) that there is no matching run starting from any location containing the formula $p\vee q$ (and all initial locations of the transition system contain it). The reason for this is that the first state of the prefix does not satisfy p ($\neg p\in\overline{IS}(0)$) and the remainder does not satisfy $p\vee q$ ($\neg(p\vee q)\in\overline{IS}(1)$). There is no matching run starting from the middle location, since it contains p. Any successor location of the left location contains $p\vee q$ again. According to the informative sequence, a run from such a successor location (left and middle) for the remainder $\{p\}$ does not exist since the second state of the prefix does not satisfy q ($\neg q\in\overline{IS}(1)$). This immediately rules out both locations as possible locations for a matching run and thus a matching run does not exist.

5.1 Correctness

The above example illustrates that for an informative bad prefix, there is no matching run on the tableau automaton. Vice versa, if there is no matching run for a prefix on an automaton $[A_\varphi]$, then the prefix is informative for $\neg\varphi$. This relationship between a finite state sequence being an informative bad prefix and the existence of a matching run is formalised in theorem 5.6 of this section. The example also showed the relationship between good prefixes and finite runs on the tableau automaton ending in the location $\varnothing$. Every finite run on $\langle A_\varphi\rangle$ ending in $\varnothing$ constitutes an informative sequence matching informative good prefixes. Conversely, for any informative good prefix such a run can be found. This is demonstrated with theorem 5.8. For the proof of correctness, we extend the notion of bad prefix to sets of formulas and to sets of such sets (such as the normal forms NF).

Definition 5.2 A finite state sequence $\bar{\tau}$ is an informative bad prefix of a set Φ of formulas if there is some $\psi \in \Phi$ such that $\bar{\tau}$ is an informative bad prefix for ψ or there is some $\psi \in \mathit{Next}(\Phi)$, such that $\bar{\tau}^1$ is an informative bad prefix for ψ. It is an informative bad prefix for a set Θ of such sets, if it is an informative bad prefix for every $\Phi \in \Theta$.

Automaton for Bad Prefixes

The normal form procedure preserves informative bad prefixes. If a prefix is informatively bad for a normal form of some formula, then it is also informatively bad for the formula itself.

Lemma 5.3 *If $\bar{\tau}$ is an informative bad prefix for $NF(\Phi)$, then $\bar{\tau}$ is an informative bad prefix for $\bigwedge \Phi$.*

For the proof, see appendix A. Next follows the main lemma to show that prefixes for which there is no matching run on the tableau automaton starting from some location Φ, are informatively bad for the formula corresponding to the location Φ.

Lemma 5.4 *Let A_φ be a tableau automaton, let Φ be a location of A_φ and let $\bar{\tau}$ be a finite state sequence for which there is no run on A_φ starting from Φ. Then $\bar{\tau}$ is an informative bad prefix for Φ.*

Proof. By induction on the length of the prefix $\bar{\tau}$.

- If $|\bar{\tau}| = 1$ then there is some $\psi \in \Phi$, either an atomic proposition or the negation of an atomic proposition, such that $\bar{\tau}(0) \not\models \psi$ and thus $\{\neg\psi\}\varnothing$ is an informative sequence showing that $\bar{\tau}$ is an informative bad prefix for Φ.
- If $|\bar{\tau}| > 1$ then either
  - the first symbol does not match the location Φ, which is similar to the first case, or
  - the first symbols matches the location Φ, but there is no successor location for which there is a run. By induction we have that $\bar{\tau}^1$ is an informative bad prefix for every successor location Φ_i, and thus for $NF(\mathit{Next}(\Phi))$, and by lemma 5.3 it is an informative bad prefix for $\bigwedge \mathit{Next}(\Phi)$. From this it follows that $\bar{\tau}$ is an informative bad prefix for Φ.

□

The following lemma is the main ingredient to show the converse, i.e. that informative bad prefixes have no matching run on the tableau automaton.

Lemma 5.5 *Let $\psi \in \Phi$ and let $\overline{IS}$ be an informative sequence demonstrating $\neg\psi$ for $\bar{\tau}$. Then there is no run for $\bar{\tau}$ on $[A_\varphi]$ starting from Φ.*

This lemma is proved by induction on the length of $\bar{\tau}$ and the structure of ψ. The proof is in appendix A. Now we can show that our tableau automata accept all finite sequences except the ones that are informative for $\neg\varphi$ (Kupferman and Vardi show a similar result for alternating automata in [9]).

Theorem 5.6 *Let φ be an* LTL *formula and let A_φ be a tableau automaton for φ. Then $[A_\varphi]$ accepts finite state sequence $\bar{\tau}$ iff $\bar{\tau}$ is not an informative bad prefix of φ.*

Proof. ($\Rightarrow$) Assume towards a contradiction that $\bar{\tau}$ is an informative bad prefix for φ. Any initial run starts from a location Φ such that $\varphi \in \Phi$. But by lemma 5.5 such a run cannot exist.

($\Leftarrow$) Again by contradiction. Assume that $\bar{\tau}$ is not accepted by $[A_\varphi]$. Then by lemma 5.4, for every $\Phi \in NF(\{\varphi\})$ (the initial locations of the automaton), $\bar{\tau}$ is an informative bad prefix for Φ. Thus by lemma 5.3, $\bar{\tau}$ is an informative bad prefix for φ. □

Automaton for Good Prefixes

Next, we show that informative *good* prefixes are recognised by the automaton $\langle A_\varphi \rangle$.

Lemma 5.7 *Let Φ be a set of formulas and let $\overline{IS}$ be an informative sequence with $\Phi \subseteq \overline{IS}(0)$. Then there is some $\Phi' \in NF(\Phi)$ such that $\Phi' \subseteq \overline{IS}(0)$ and $Next(\Phi') \subseteq \overline{IS}(1)$.*

The proof is given in appendix A. As a consequence, a finite state sequence is an informative good prefix iff there is a matching run leading to the location $\varnothing$.

Theorem 5.8 *Let A_φ be the on-the-fly tableau automaton of the formula φ. A finite state sequence $\bar{\tau}$ is an informative good prefix of φ iff $\langle A_\varphi \rangle$ accepts $\bar{\tau}$.*

Proof. ($\Rightarrow$) Let $\overline{IS}$ be an informative sequence with $\varphi \in \overline{IS}(0)$. By lemma 5.7, there is some $\Phi \in NF(\varphi)$ such that $\Phi \subseteq \overline{IS}(0)$ and $Next(\Phi) \subseteq \overline{IS}(1)$. Repeating the argument, we can show that there is a run $\bar{q}$ such that $\bar{q}(k) \subseteq \overline{IS}(k)$ for all $0 \leq k \leq |\overline{IS}|$. Thus $\bar{q}(|\overline{IS}|) = \varnothing$.

($\Leftarrow$) Let $\bar{q}$ be such a run. Then $\bar{q}$ itself is an informative sequence for φ since all locations are locally informative and all edges are temporally informative. □

5.2 Practical Use of the Prefix Automata

We have seen how we can construct finite state automata that recognise the informative good and bad prefixes of a particular formula φ. It has been shown that both automata share the same transition system but differ only in acceptance conditions. On the basis of these automata one can construct an observer that is linked to a running model in such a way that it can evaluate its atomic propositions defined as boolean properties of the model and is run in lock step or alternatingly with the (relevant) transitions of the model. As the monitor is made deterministic (possibly using an on-the-fly determinisation), the analysis of the increasing run can be performed incrementally. Detection of informative good or bad prefixes can be reported, possibly halting the execution of the model.

If an execution is halted without encountering either of both conditions, the encountered prefix is inconclusive w.r.t. the formula φ. Yet, further analysis of the prefix might still reveal interesting (statistical) information. How this information may be obtained however, requires further study. One would need to know what subformulas of φ have been informatively fulfilled and possibly, how many times.

6 Conclusions and Future Work

The use of temporal logic model-checking techniques on running implementations or simulations of detailed system models calls for the on-the-fly incremental analysis of finite execution traces. In this paper we have shown how to construct from a linear temporal logic formula, a finite state automaton that can act as a monitor to perform this type of analysis for the detection of (informative) satisfaction as well as violation of the formula by a finite execution of the system. These finite state automata can be determinised (possibly on-the-fly as well), to remove their non-determinism.

We are further investigating the use of similar techniques to construct runtime monitors (in the form of timed-automata) for real-time temporal logic. We will further implement the technique in a simulator for concurrent systems called SHESim[6].

References

[1] Alpern, B. and F. Schneider, *Defining liveness*, Information processing letters **21** (1985), pp. 181–185.

[2] Canfield, W., E. Emerson and A. Saha, *Checking formal specifications under simulation*, in: *Proceedings International Conference on Computer Design. VLSI in Computers and Processors* (1997), pp. 455–460.

[3] Daniele, M., F. Giunchiglia and M. Y. Vardi, *Improved automata generation for linear temporal logic*, in: N. Halbwachs and D. Peled, editors, *Computer Aided Verification: 11th International Conference Proceedings, CAV'99, Trento, Italy, July 6-10, 1999 (LNCS 1633)* (1999), pp. 249–260.

[4] D'Souza, D., "On-the-Fly Verification for Linear Time Temporal Logic," Master's thesis, SPIC Mathematical Institute, Madras (1997).

[5] Etessami, K. and G. Holzman, *Optimising büchi automata*, in: *Proceedings of the 11th Int. Conf. On Concurrency Theory (CONCUR'2000)* (2000), pp. 153–167.

[6] Geilen, M., J. Voeten, P. van der Putten, L. van Bokhoven and M. Stevens, *Object-Oriented modelling and specification using SHE*, Journal of Computer Languages, special issue for VFM'99 (to be published) (2000).

[7] Gerth, R., D. Peled, M. Vardi and P. Wolper, *Simple on-the-Fly automatic verification of linear temporal logic*, in: *Proc. IFIP/WG6.1 Symp. Protocol Specification Testing and Verification (PSTV95), Warsaw Poland* (1995), pp. 3–18.

[8] Hoffmann, D., J. Ruf, T. Kropf and W. Rosenstiel, *Simulation meets verification - checking temporal properties in SystemC*, in: F. Vajda, editor, *Proceedings of the 26th EUROMICRO Conference - Volume I, Maastricht, the Netherlands, Sept 5-7, 2000* (2000), pp. 435–438.

[9] Kupferman, O. and M. Y. Vardi, *Model checking of safety properties*, in: N. Halbwachs and D. Peled, editors, *Computer Aided Verification: 11th International Conference Proceedings, CAV'99, Trento, Italy, July 6-10, 1999 (LNCS 1633)* (1999), pp. 172–183.

[10] Lichtenstein, O. and A. Pnueli, *Checking that finite state concurrent programs satisfy their linear specification*, in: *Proc. Of Twelfth Annual ACM Symposium on Principles of Programming Languages* (1985), pp. 97–107.

[11] Pnueli, A., *The temporal logic of programs*, in: *Proc. Of the 18th Annual Symposium on Foundations of Computer Science* (1977), pp. 46–57.

[12] Sistla, A., *Safety, liveness and fairness in temporal logic*, Formal Aspects of Computing **6** (1994), pp. 495–511.

[13] Vardi, M. and P. Wolper, *An Automata-Theoretic approach to automatic program verification*, in: *Proc. Of Logic in Computing Science*, 1986.

A Proofs

Proof of lemma 5.3

Lemma 5.3 states that informativeness of prefixes is preserved by the normal form construction. To prove it, we need to define when a prefix is considered to be informative for the artifacts used during the normal form procedure.

Definition A.1 A finite state sequence $\bar{\tau}$ is an informative bad prefix for a term $\langle New, Old\rangle$ if there is some $\psi \in New \cup Old$ such that $\bar{\tau}$ is an informative bad prefix for ψ or there is some $\psi \in Next(New, Old)$ such that $\bar{\tau}^1$ is an informative bad prefix for ψ.

Definition A.2 A finite state sequence $\bar{\tau}$ is an informative bad prefix for a set Θ of terms if $\bar{\tau}$ is an informative bad prefix for every $\langle New, Old\rangle \in \Theta$.

Two informative sequences can be combined into a single new one, simply by taking the union of the corresponding sets. If $\overline{IS}_1$ and $\overline{IS}_2$ are both informative sequences, then $(\overline{IS}_1 \cup \overline{IS}_2)(k) = \overline{IS}_1(k) \cup \overline{IS}_2(k)$ for all $k \geq 0$ (taking $\overline{IS}(k) = \varnothing$ if $k > |\overline{IS}|$). It is easy to see that if $\overline{IS}_1$ and $\overline{IS}_2$ are informative sequences for $\bar{\tau}$, then $\overline{IS}_1 \cup \overline{IS}_2$ is an informative sequence for $\bar{\tau}$ as well.

The next lemma shows that reductions in the normal form procedure preserve informativeness of bad prefixes.

Lemma A.3 *Let prefix $\bar{\tau}$ be an informative bad prefix for Θ' and let $\Theta \Rightarrow \Theta'$ in the normal form procedure. Then $\bar{\tau}$ is an informative bad prefix for Θ.*

Proof. One can prove this for the reduction cases individually, which is a tedious case analysis. We only show case 5. $\Theta = \Theta'' \cup \{\langle New \cup \{\psi_1 \vee \psi_2\}, Old\rangle\}$ and $\Theta' = \Theta'' \cup \{\langle New \cup \{\psi_1\}, Old \cup \{\psi_1 \vee \psi_2\}\rangle, \langle New \cup \{\psi_2\}, Old \cup \{\psi_1 \vee \psi_2, \}\rangle\}$. If $\bar{\tau}$ is an informative bad prefix of Θ', it is a bad prefix of both $\langle New \cup \{\psi_1\}, Old \cup \{\psi_1 \vee \psi_2\}\rangle$ and $\langle New \cup \{\psi_2\}, Old \cup \{\psi_1 \vee \psi_2, \}\rangle$. If $\overline{IS}$ is an informative sequence demonstrating this (both), then $\overline{IS} \cup \{\neg(\psi_1 \vee \psi_2)\}$ is an informative sequence for $\langle New \cup \{\psi_1 \vee \psi_2\}, Old\rangle$. From this it follows straightforwardly that $\bar{\tau}$ is an informative bad prefix for Θ (note that moving $\psi_1 \vee \psi_2$ to *Old* does not add any informativeness constraints). $\square$

From this it follows immediately that the entire normal form procedure preserves informativeness of bad prefixes.

Lemma A.4 (Lemma 5.3) *If $\bar{\tau}$ is an informative bad prefix for $NF(\Phi)$, then $\bar{\tau}$ is an informative bad prefix for $\bigwedge \Phi$.*

Proof of lemma 5.5

This lemma says that an informative bad prefix cannot have a run on the on-the-fly tableau automaton.

Lemma A.5 (Lemma 5.5) *Let $\psi \in \Phi$ and let $\overline{IS}$ be an informative sequence demonstrating $\neg\psi$ for $\bar{\tau}$. Then there is no Φ-run for $\bar{\tau}$ on $[A_\varphi]$.*

Proof. By induction on the length of $\bar{\tau}$ and the structure of ψ. We show the case $\psi = \psi_1 \mathsf{U} \psi_2$, then either $\psi_2 \in \Phi$ or $\psi_1 \in \Phi$ and $\psi \in \bar{q}(1)$ for any appropriate run $\bar{q}$. Since $\neg(\psi_1 \mathsf{U} \psi_2) \in \overline{IS}(0)$, $\neg\psi_2 \in \overline{IS}(0)$ and $\neg\psi_1 \in \overline{IS}(0)$ or $\neg\psi \in \overline{IS}(1)$. That such a run $\bar{q}$ cannot exist follows by induction. Notice that the latter case can only occur if $|\bar{\tau}| > 1$ since $\overline{IS}(|\bar{\tau}|) = \varnothing$, i.e. $\neg\psi$ cannot be postponed forever. $\square$

Proof of lemma 5.7

This lemma suggests how informative sequences can be used to construct a run to the empty location. The lemma is proved using an invariant on the normal form procedure, introduced in the next definition.

Definition A.6 In the following lemma, the predicate $Inv(\Theta, \overline{IS})$ holds iff there is some term $\langle New, Old \rangle \in \Theta$ such that $New \cup Old \subseteq \overline{IS}(0)$ and $Next(New, Old) \subseteq \overline{IS}(1)$.

$Inv(\Theta, \overline{IS})$ states that $\overline{IS}$ is informative for at least one of the terms in Θ and thus for the set itself. We show that $Inv(\Theta, \overline{IS})$ is invariant under reductions in the normal form procedure.

Lemma A.7 *Let $\Theta \Rightarrow \Theta'$, let $\overline{IS}$ be an informative sequence and assume that $Inv(\Theta, \overline{IS})$ holds, then also $Inv(\Theta', \overline{IS})$ holds.*

Proof. By case analysis of the procedure. We only show case 9. $\Theta = \Theta'' \cup \{\langle New \cup \{\psi_1 \vee \psi_2\}, Old \rangle\}$ and $\Theta' = \Theta'' \cup \{\langle New \cup \{\psi_1, \psi_2\}, Old \cup \{\psi_1 \vee \psi_2\}\rangle, \langle New \cup \{\psi_2\}, Old \cup \{\psi_1 \vee \psi_2\}\rangle\}$. If there is some $\langle New', Old' \rangle \in \Theta''$ such that $New' \cup Old' \subseteq \overline{IS}(0)$ and $Next(New', Old') \subseteq \overline{IS}(1)$ then the result is trivial. Otherwise, the term satisfying the property is $\langle New \cup \{\psi_1 \vee \psi_2\}, Old \rangle$. Then $\psi_1 \vee \psi_2 \in \overline{IS}(0)$ and by local informativeness $\psi_2 \in \overline{IS}(0)$.

- If $\psi_1 \in \overline{IS}(0)$ then $New \cup \{\psi_1, \psi_2\} \cup Old \cup \{\psi_1 \vee \psi_2\} \subseteq \overline{IS}(0)$ and $Next(New \cup \{\psi_1, \psi_2\}, Old \cup \{\psi_1 \vee \psi_2\}) \subseteq Next(New \cup \{\psi_1 \vee \psi_2\}, Old) \subseteq \overline{IS}(1)$.
- If $\psi_1 \notin \overline{IS}(0)$ then $New \cup \{\psi_2\} \cup Old \cup \{\psi_1 \vee \psi_2\} \subseteq \overline{IS}(0)$ and $Next(New \cup \{\psi_2\}, Old \cup \{\psi_1 \vee \psi_2\}) \subseteq Next(New \cup \{\psi_1 \vee \psi_2\}, Old) \cup \{\psi_1 \vee \psi_2\} \subseteq \overline{IS}(1)$ since $\psi_1 \vee \psi_2 \in \overline{IS}(1)$ by temporal informativeness.

$\square$

From the previous lemma it follows directly that the following holds for the entire normal form procedure.

Lemma A.8 (Lemma 5.7) *Let Φ be a set of formulas and let $\overline{IS}$ be an informative sequence with $\Phi \subseteq \overline{IS}(0)$. Then there is some $\Phi' \in NF(\Phi)$ such that $\Phi' \subseteq \overline{IS}(0)$ and $Next(\Phi') \subseteq \overline{IS}(1)$.*