FI MU

Faculty of Informatics
Masaryk University

Relating Hierarchy of Linear Temporal Properties to Model Checking

by

Ivana Černá
Radek Pelánek

FI MU Report Series **FIMU-RS-2003-03**

April 2003

Relating Hierarchy of Linear Temporal Properties to Model Checking

Ivana Černá and Radek Pelánek [⋆]

Department of Computer Science, Faculty of Informatics
Masaryk University Brno, Czech Republic
{cerna,xpelanek}@fi.muni.cz

Abstract. The hierarchy of properties as overviewed by Manna and Pnueli [23] relates language, topology, ω-automata, and linear temporal logic classifications of properties. We provide new characterisations of this hierarchy in terms of automata with Büchi, co-Büchi, and Streett acceptance condition and in terms of Σ_i^{LTL} and Π_i^{LTL} hierarchies. Afterwards, we analyse the complexity of the model checking problem for particular classes of the hierarchy and thanks to the new characterisations we identify those linear time temporal properties for which the model checking problem can be solved more efficiently than in the general case.

1 Introduction

Model checking has become a popular technique for formal verification of reactive systems. The model checking process has several phases – the major ones being modelling of the system, specification of desired properties of the system and the actual process of automatic verification. Each of these phases has its specific difficulties. In this paper we study linear temporal properties and algorithms for the automatic verification of these properties.

Reactive systems maintain an ongoing interaction with their environment and thus produce computations – infinite sequences of states. When analysing the behaviour of such a system we are interested in some finite set AP of observable propositions about states. Hence, we can view a computation of the system as an infinite word over 2^{AP}. In general, we define a temporal *property* as a language of infinite words. A reactive system S is said to have a property P if all possible computations of S belong to P.

The problem of proper and correct specification of properties the system ought to satisfy led to a careful study of theoretical aspects of properties. Manna and Pnueli [23] have proposed a classification of temporal properties into a hierarchy. They characterise the classes of the hierarchy through four views: a language-theoretic view, a topological view, a temporal logic view, and an automata view. The fact that the hierarchy can be defined in many different ways shows the robustness of this hierarchy.

[⋆] Supported by GA ČR grant no. 201/03/0509

Model checking theory is devoted to the development of efficient algorithms for the automatic verification of properties of reactive systems. A very successful approach to verifying properties expressed as linear temporal logic (LTL) formulas makes use of automata over infinite words. Here the problem of verifying a property reduces to the problem whether a given automaton recognises a non-empty language (so called non-emptiness check). The complexity of the non-emptiness check depends on the type of the automaton. Bloem, Ravi, and Somenzi [1] have studied two specialised types of automata, called *weak* and *terminal*, for which the non-emptiness check can be performed more efficiently than in the general case.

Our contribution Our aim is to classify temporal properties specifiable by linear temporal logic formulas with respect to the complexity of their verification. To this end we provide a classification of temporal properties through two new views. First, we characterise properties in terms of automata over infinite words (ω-automata) with Büchi, co-Büchi and Streett acceptance condition and in terms of weak and terminal automata. Weak and terminal automata are used in the verification process and are checked for non-emptiness.

For the second characterisation we introduce a new hierarchy (called Until-Release hierarchy) of LTL formulas based on alternation depth of temporal operators Until and Release. We provide a relationship between the Until-Release hierarchy and the hierarchy by Manna and Pnueli [23].

Our new classification provides us with an exact relationship between the type of a formula and the type of an automaton, which is checked for non-emptiness in the model checking process of the formula.

In the second part of the paper we enquire into particular automata and analyse the complexity of their non-emptiness check in connection with both explicit and implicit representation of automata. This gives us an exact relationship between types of properties and the complexity of their verification. Finally, we discuss the possibility of exact determination of the type of a formula.

Related work The previous work on verification, which takes into account a classification of properties, is partly devoted to the proof-based approach to verification [5]. Papers on specialised model checking algorithms either cover only part of the hierarchy or have a heuristic nature. Vardi and Kupferman [20] study the model checking of safety properties. Schneider [26] is concerned with a translation of persistence properties into weak automata. Bloem and Somenzi study heuristics for the translation of a formula into weak (terminal) automaton [28] and suggest specialised algorithms for the non-emptiness problem [1]. Our work covers all types of properties and brings out the correspondence between the type and the complexity of non-emptiness check.

Plan of the work Section 2 introduces the hierarchy of properties as presented in [23] and establishes new characterisations of the hierarchy. First we supplement the automata view and than we revise the characterisation through formulas of linear temporal logic. Here we find a use for the Until/Release alternation depth of formulas. Section 3 defines the model checking problem and relates the type of a formula with the type of a corresponding automaton. Section 4 draws distinctions in non-emptiness check for different types of automata used in model checking. Algorithms for non-emptiness check using both explicit and symbolic representation of the state space are displayed and their complexity is discussed. Finally, Section 5 considers the task to determine the type of a formula.

2 Hierarchy of Temporal Properties

The hierarchy studied by Manna and Pnueli [23] classifies properties into six classes: *guarantee*, *safety*, *obligation*, *persistence*, *recurrence*, and *reactivity* properties.

Definition 1 (Language-theoretic view [23]).
Let $P \subseteq \Sigma^\omega$ be a property over Σ.

- *P is a* safety *property if there exists a language of finite words $L \subseteq \Sigma^*$ such that for every $w \in P$ all finite prefixes of w belong to L.*
- *P is a* guarantee *property if there exists a language of finite words $L \subseteq \Sigma^*$ such that for every $w \in P$ there exists a finite prefix of w which belongs to L.*
- *P is an* obligation *property if P can be expressed as a positive boolean combination of safety and guarantee properties.*
- *P is a* recurrence *property if there exists a language of finite words $L \subseteq \Sigma^*$ such that for every $w \in P$ infinitely many prefixes of w belong to L.*
- *P is a* persistence *property if there exists a language of finite words $L \subseteq \Sigma^*$ such that for every $w \in P$ all but finitely many prefixes of w belong to L.*
- *P is a* reactivity *property if P can be expressed as a positive boolean combination of recurrence and persistence properties.*

In what follows, the abbreviation κ-property stands for a property of one of the six above mentioned types. Inclusions, which relate the corresponding classes into a hierarchy, are depicted in Fig. 1. Classes which are higher up strictly contain classes which are lower down.

Properties from particular classes can be intuitively viewed as making different claims about occurrences of “good” and “bad” things during the computation. Characteristics of properties from particular classes are listed below.

- *safety*: something good always occurs (nothing bad occurs)
- *guarantee*: something good happens at least once
- *obligation*: conditional occurrence of a good thing
- *recurrence*: something good occurs infinitely many times
- *persistence*: something good occurs continuously from a certain point (bad thing occurs only finitely many times)
- *reactivity*: conditional occurrence of infinitely many good things

2.1 Automata View

Manna and Pnueli [23] have defined the hierarchy of properties in terms of deterministic Streett predicate automata. Automata for considered classes of properties differ in restrictions on their transition functions and acceptance conditions. In this section we provide a new characterisation of the hierarchy in terms of deterministic ω-automata which uses only restrictions on acceptance conditions (the transition function is always the same). We find this characterisation more uniform and believe that it provides better insight into the hierarchy. On top of that we study other widely used types of ω-automata and show that each of them exactly corresponds to one class in the hierarchy.

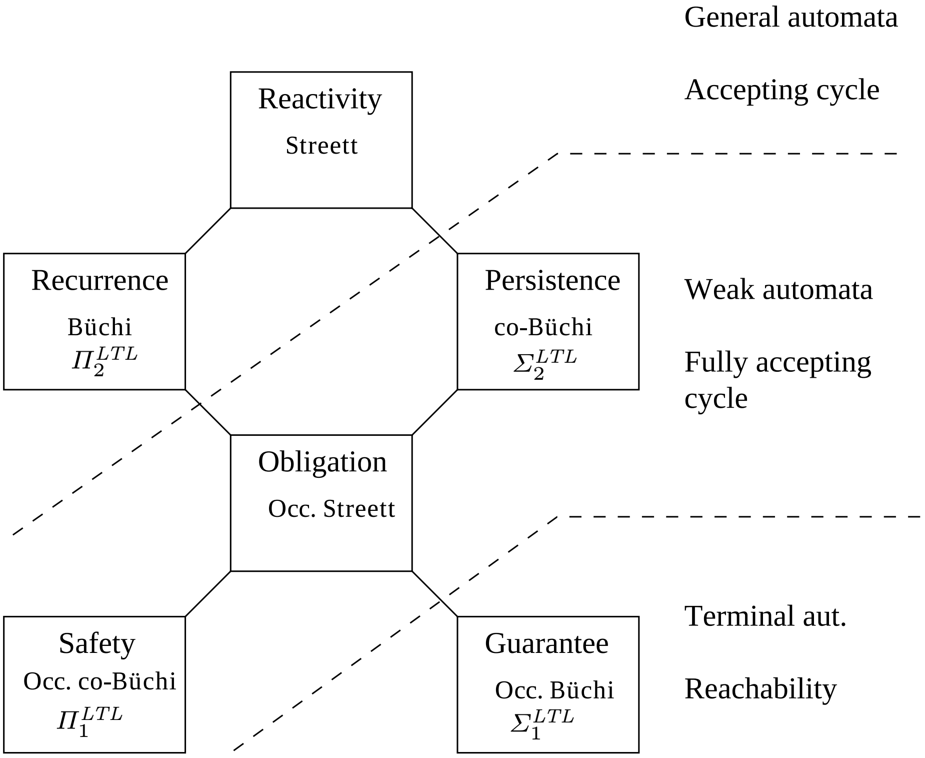

Fig. 1. Relations between classes of the hierarchy and their different characterisations. Classes which are higher up properly contain classes which are lower down. Classes on the same level are dual with respect to complementation, while the classes obligation and reactivity can be obtained by boolean combinations of properties from classes lower down.

An *ω-automaton* is a tuple $A = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, δ is a transition function, and α is an acceptance condition. The transition function determines four types of automata: *deterministic*, *nondeterministic*, *universal*, and *alternating*. A nondeterministic automaton has a transition function of the type $\delta : Q \times \Sigma \to 2^Q$. A run π of such an automaton on an infinite word $w = w(0)w(1)\ldots$ over Σ is a sequence of states $\pi = r_0, r_1, \ldots$ such that $r_0 = q_0$ and $r_{i+1} \in \delta(r_i, w(i))$ for each $i \geq 0$. A nondeterministic automaton accepts a word w if there exists an accepting run (see below) on w. Universal automata are defined in the same way, the only difference is that the universal automaton accepts a word w if all runs on w are accepting. Deterministic automata are such that $|\delta(q,a)| = 1$ for all $q \in Q, a \in \Sigma$ (there is a unique run on each word). Alternating automata form a generalisation of nondeterministic and universal automata. For the definition of alternating ω-automata see e.g. [24]. By the abuse of notation, we use $(q,r) \in \delta$ with the meaning $\exists a \in \Sigma : r \in \delta(q,a)$.

For a run π we define the *infinity set*, $\mathit{Inf}(\pi)$, to be the set of all states that appear infinitely often in π and the *occurrence set*, $\mathit{Occ}(\pi)$, to be the set of states that appear at least once in π. Acceptance conditions α are defined with respect to infinity set as follows:

- *Büchi* condition $\alpha \subseteq Q$: a run π is accepting iff $\mathit{Inf}(\pi) \cap \alpha \neq \emptyset$
- *co-Büchi* condition $\alpha \subseteq Q$: a run π is accepting iff $\mathit{Inf}(\pi) \cap \alpha = \emptyset$
- *Streett* condition $\alpha = \{\langle G_1, R_1\rangle, \ldots, \langle G_n, R_n\rangle\}, G_i, R_i \subseteq Q$: a run π is accepting iff $\forall i : (\mathit{Inf}(\pi) \cap G_i \neq \emptyset \Rightarrow \mathit{Inf}(\pi) \cap R_i \neq \emptyset)$

For every acceptance condition we can define its ”occurrence” version (also called *Staiger-Wagner* acceptance) [22, 31]:

- *occurrence Büchi* condition $\alpha \subseteq Q$: a run π is accepting iff $\mathit{Occ}(\pi) \cap \alpha \neq \emptyset$
- *occurrence co-Büchi* condition $\alpha \subseteq Q$: a run π is accepting iff $\mathit{Occ}(\pi) \cap \alpha = \emptyset$
- *occurrence Streett* condition $\alpha = \{\langle G_1, R_1\rangle, \ldots, \langle G_n, R_n\rangle\}, G_i, R_i \subseteq Q$: a run π is accepting iff $\forall i : (\mathit{Occ}(\pi) \cap G_i \neq \emptyset \Rightarrow \mathit{Occ}(\pi) \cap R_i \neq \emptyset)$

A property P is defined to be *specifiable by automata* if there is an ω-automaton A which accepts a word w if and only if $w \in P$.

The characteristic of the temporal hierarchy by Manna and Pnueli [23] makes use of following special types of automata.

Definition 2 (Automata-theoretic view [23]).
Let $A = \langle \Sigma, Q, q_0, \delta, \{\langle G_1, R_1\rangle, \ldots, \langle G_n, R_n\rangle\}\rangle$ be a deterministic infinite occurrence Streett automaton. Let $\mathit{Good} = (Q \setminus G_1) \cup R_1$ and $\mathit{Bad} = Q \setminus \mathit{Good}$.

- *A is a* safety *automaton if $n = 1$ and $\forall q \in \mathit{Bad}, q' \in \mathit{Good} : (q,q') \notin \delta$*
- *A is a* guarantee *automaton if $n = 1$ and $\forall q \in \mathit{Good}, q' \in \mathit{Bad} : (q,q') \notin \delta$*
- *A is an* obligation *automaton if $n = 1$ and there exists rank function $\sigma : Q \to \{0, \ldots, k\}$ such that:*
  - $(q,q') \in \delta \Rightarrow \sigma(q) \leq \sigma(q')$
  - $(q \in \mathit{Bad}, q' \in \mathit{Good}, (q,q') \in \delta) \Rightarrow \sigma(q) < \sigma(q')$
  - $(q \in \mathit{Good}, q' \in \mathit{Bad}, (q,q') \in \delta) \Rightarrow \sigma(q) < n$
- *A is a* recurrence *automaton if $n = 1$ and $G_1 = Q$*
- *A is a* persistence *automaton if $n = 1$ and $R_1 = \emptyset$*
- *A is a* reactivity *automaton otherwise*

A theorem relating the language and automata views follows.

Theorem 1 ([23]). *Let P be a property specifiable by automata. Then P is a κ-property if and only if it is specifiable by a κ-automaton.*

Our new characterisation of the hierarchy is based on deterministic automata as well. The most important difference is the fact that automata differ only in their acceptance condition. The characterisation is summarized in the first line of Table 1, particular relations are proved below.

Proposition 1. *Let A be a guarantee, safety, or obligation automaton with an accepting condition $\{\langle G, R\rangle\}$ and let π be a run of A. Then the run π is accepting if and only if $\mathit{Inf}(\pi) \subseteq (Q \smallsetminus G) \cup R$.*

Proof:
"$\Leftarrow$" Let $\mathit{Inf}(\pi) \subseteq (Q \smallsetminus G) \cup R$. Then the implication $\mathit{Inf}(\pi) \cap G \neq \emptyset \Rightarrow \mathit{Inf}(\pi) \cap R \neq \emptyset$ is satisfied and π is accepting.
"$\Rightarrow$" A direct consequence of the requirements on transition functions of guarantee, safety, and obligation automata. □

Lemma 1. *Property P is specifiable by a guarantee automaton if and only if it is specifiable by a deterministic occurrence Büchi automaton.*

Proof:
"$\Rightarrow$" Let $A = \langle \Sigma, Q, q_0, \delta, \{\langle G, R\rangle\}\rangle$ be a guarantee automaton. Let us define an occurrence Büchi automaton $A' = \langle \Sigma, Q, q_0, \delta, (Q \smallsetminus G) \cup R\rangle$. We claim that A' is equivalent to A. Let w be an infinite word and π the run of A on w. Since both automata have the same transition function, π is the run of A' on w as well. By Proposition 1, the automaton A accepts $w \Leftrightarrow \mathit{Inf}(\pi) \subseteq (Q \smallsetminus G) \cup R \Leftrightarrow \mathit{Occ}(\pi) \cap (Q \smallsetminus G) \cup R \neq \emptyset$ (by the definition of a guarantee automaton) $\Leftrightarrow$ A' accepts w.
"$\Leftarrow$" Let $A = \langle \Sigma, Q, q_0, \delta, \alpha\rangle$ be a deterministic occurrence Büchi automaton. Let us define an automaton $A' = \langle \Sigma, Q \cup Q', q_0, \delta', \{\langle Q, \emptyset\rangle\}\rangle$, where $Q' = \{q' \mid q \in Q\}$ and

- if $q \in \alpha$ and $\delta(q, a) = r$ then $\delta'(q, a) = r'$
- if $q \in Q \smallsetminus \alpha$ and $\delta(q, a) = r$ then $\delta'(q, a) = r$
- if $\delta(q, a) = r$ then $\delta'(q', a) = r'$

It is easy to see that A' is a guarantee automaton equivalent to A. □

Lemma 2. *Property P is specifiable by a safety automaton if and only if it is specifiable by a deterministic occurrence co-Büchi automaton.*

Proof: Analogous to the previous one. □

Lemma 3. *Property P is specifiable by an obligation automaton if and only if it is specifiable by a deterministic occurrence Streett automaton.*

Proof:
"⇒" Let $A = \langle \Sigma, Q, q_0, \delta, \{\langle G, R\rangle\}\rangle$ be an obligation automaton with ranking function $\sigma : Q \to \{0, \ldots, k\}$, and sets $Good = (Q \smallsetminus G) \cup R$, $Bad = Q \smallsetminus Good$. We define an occurrence Streett automaton $A' = \langle \Sigma, Q, q_0, \delta, \alpha'\rangle$ where $\alpha' = \{\langle G_0, R_0\rangle, \ldots, \langle G_k, R_k\rangle\}$ and for $0 \leq l \leq k$:

- $G_l = \{q \mid q \in Bad$ and $\sigma(q) = l\}$
- $R_l = \{q \mid q \in Good$ and $\sigma(q) > l\}$

We claim that A' is equivalent to A. Let w be an infinite word and π the run of A on w. Due to the monotonicity of σ there exists a number m such that $\forall q \in Inf(\pi) : \sigma(q) = m$.

- A accepts w: By Proposition 1 we have $Inf(\pi) \subseteq Good$ and due to monotonicity
  - $\forall l \geq m : Occ(\pi) \cap G_l = \emptyset$
  - $\forall l < m : Occ(\pi) \cap R_l \neq \emptyset$

  Hence π is the accepting run of A' on w.
- A does not accept w: Similar arguments get hold of $Inf(\pi) \subseteq Bad$ and
  - $Occ(\pi) \cap G_m \neq \emptyset$
  - $Occ(\pi) \cap R_m = \emptyset$

  Thus π is not accepting and A' does not accept w.

"⇐" Let A be a deterministic occurrence Streett automaton $A = \langle \Sigma, Q, q_0, \delta, \{\langle G_1, R_1\rangle, \ldots, \langle G_n, R_n\rangle\}\rangle$. We define an automaton $A' = \langle \Sigma, Q \times 2^Q, (q_0, \{q_0\}), \delta', \{\langle \emptyset, P\rangle\}\rangle$, where

- $\delta'((q, S), a) = (\delta(q, a), S \cup \{\delta(q, a)\})$
- $P = \{(q, S) \mid \forall 1 \leq i \leq n : S \cap G_i \neq \emptyset \Rightarrow S \cap R_i \neq \emptyset\}$

A' is an obligation automaton with ranking function $\sigma(q, S) = |S|$. Moreover, A accepts w iff the run π of A on w satisfies $\forall 1 \leq i \leq n : Occ(\pi) \cap G_i \neq \emptyset \Rightarrow Occ(\pi) \cap R_i \neq \emptyset$. This happens if and only if the run π' of A' on w gets eventually trapped within the set P, i.e. A' accepts w'. □

Lemma 4. *Property P is specifiable by a recurrence automaton if and only if it is specifiable by a deterministic Büchi automaton.*

Proof: A recurrence automaton with an accepting condition $\{\langle Q, R\rangle\}$ is clearly equivalent to a deterministic Büchi automaton with the accepting condition R and vice versa. □

Lemma 5. *Property P is specifiable by a persistence automaton if and only if it is specifiable by a deterministic co-Büchi automaton.*

Proof: A persistence automaton with an accepting condition $\{\langle G, \emptyset\rangle\}$ is clearly equivalent to a deterministic co-Büchi automaton with the accepting condition G and vice versa. □

Since reactivity automata are just unconstrainted deterministic Streeett automata, we obtain the following characterization of hierarchy classes specifiable by automata:

Theorem 2. *Let P be a property specifiable by automata. Then P is a guarantee, safety, obligation, persistence, recurrence, or reactivity property if and only if it is specifiable by a deterministic occurrence Büchi, occurrence co-Büchi, occurrence Streett, co-Büchi, Büchi, or Streett automaton respectively.*

To make the picture complete we have examined other types of automata as well (see Table 1). For every possible combination of transition function and acceptance condition the class of specifiable properties exactly coincides with one class in the hierarchy. Results for infinite occurrence acceptance conditions follow from [21]. Universal occurrence Büchi and nondeterministic occurrence co-Büchi automata can be determinised through the power set construction and thus they recognise the same classes as their deterministic counterparts. The other results for occurrence acceptance condition follow from [22].

	Infinite occurrence (Inf)			Occurrence (Occ)		
	Büchi	co-Büchi	Streett	Büchi	co-Büchi	Streett
Deterministic	*recurrence*	*persistence*	*reactivity*	*guarantee*	*safety*	*obligation*
Nondeterministic	*reactivity*	*persistence*	*reactivity*	*persistence*	*safety*	*persistence*
Universal	*recurrence*	*reactivity*	*reactivity*	*guarantee*	*recurrence*	*recurrence*
Alternating	*reactivity*	*reactivity*	*reactivity*	*persistence*	*recurrence*	*reactivity*

Table 1. The expressivity – each of 24 possible inter-combinations of the transition function and acceptance condition corresponds to one of the six hierarchy classes.

Several other types of acceptance conditions have been studied in the theory of automata over infinite words, in particular Müller, Rabin and parity acceptance conditions. It can be shown that automata with any of these acceptance conditions can be effectively transformed into automata with Streett acceptance conditions [31, 21]. Therefore these types of automata also exactly recognise classes from the hierarchy.

2.2 Linear Temporal Logic View

In this section we characterise the hierarchy of properties through temporal logic formulas. We revise the classification by Chang, Manna, and Pnueli [5] and propose a new hierarchy of LTL formulas based on an alternation depth.

The set of LTL formulas is defined inductively starting from a countable set AP of atomic propositions, Boolean operators, and the temporal operators **X** (Next) and **U** (Until):

$$\Psi := a \mid \neg\Psi \mid \Psi \vee \Psi \mid \Psi \wedge \Psi \mid \mathbf{X}\Psi \mid \Psi\,\mathbf{U}\,\Psi$$

LTL formulas are interpreted in the standard way [11] on infinite words over the alphabet 2^{AP}. We adopt standard abbreviations $\mathbf{R}, \mathbf{F}, \mathbf{G}$ for temporal operators Release ($\alpha \, \mathbf{R} \, \beta \equiv \neg(\neg\alpha \, \mathbf{U} \, \neg\beta)$), Future ($\mathbf{F} \, \alpha \equiv \mathit{true} \, \mathbf{U} \, \alpha$) and Globally ($\mathbf{G} \, \alpha \equiv \mathit{false} \, \mathbf{R} \, \alpha$) respectively. A property P is defined to be *specifiable by LTL* if there is an LTL formula φ such that $w \models \varphi$ if and only if $w \in P$.

Chang, Manna, and Pnueli [5] have proposed the classification of LTL formulas into 6 types: guarantee, safety, obligation, recurrence, persistence, and reactivity formulas (in the following the notation κ-formula means the formula of type κ). Every LTL formula is a reactivity formula. Guarantee formulas are defined inductively in the Ψ_G line below, similarly for other types (type names are abbreviated by they first letter).

$$\begin{array}{llllllllll}
\Psi_G := a & |\neg a & |\Psi_G \vee \Psi_G & |\Psi_G \wedge \Psi_G & |\mathbf{X}\,\Psi_G & |\mathbf{F}\,\Psi_G & |\Psi_G\,\mathbf{U}\,\Psi_G & |\neg\Psi_S & \\
\Psi_S := a & |\neg a & |\Psi_S \vee \Psi_S & |\Psi_S \wedge \Psi_S & |\mathbf{X}\,\Psi_S & |\mathbf{G}\,\Psi_S & |\Psi_S\,\mathbf{R}\,\Psi_S & |\neg\Psi_G & \\
\Psi_O := \Psi_S & |\Psi_G & |\Psi_O \vee \Psi_O & |\Psi_O \wedge \Psi_O & |\mathbf{X}\,\Psi_O & |\Psi_O\,\mathbf{U}\,\Psi_G & |\Psi_O\,\mathbf{R}\,\Psi_S & |\neg\Psi_O & \\
\Psi_P := \Psi_S & |\Psi_G & |\Psi_P \vee \Psi_P & |\Psi_P \wedge \Psi_P & |\mathbf{X}\,\Psi_P & |\mathbf{F}\,\Psi_P & |\Psi_P\,\mathbf{U}\,\Psi_P & |\Psi_P\,\mathbf{R}\,\Psi_S & |\neg\Psi_R \\
\Psi_R := \Psi_S & |\Psi_G & |\Psi_R \vee \Psi_R & |\Psi_R \wedge \Psi_R & |\mathbf{X}\,\Psi_R & |\mathbf{G}\,\Psi_R & |\Psi_R\,\mathbf{R}\,\Psi_R & |\Psi_R\,\mathbf{U}\,\Psi_G & |\neg\Psi_P
\end{array}$$

Names of formula types connote a correspondence between κ-formulas and κ-properties. However, the correspondence is not direct. For example the formula $\mathbf{G}\,p \vee (p\,\mathbf{U}\,q)$ specifies a safety property although it is not a safety formula. Yet it is equivalent to the safety formula $q\,\mathbf{R}\,(p \vee q)$. In fact, the following correspondence holds:

Theorem 3 ([5]). *A property that is specifiable by LTL is a κ-property if and only if it is specifiable by a κ-formula.*

In recent years, considerable effort has been devoted to the study of LTL hierarchies which were defined with respect to the number of nested temporal operators Until, Since and Next ([14, 30, 19]). These hierarchies provide interesting characterizations of LTL definable languages. However, they do not seem to have a direct connection to the model checking problem. We propose a new hierarchy which is based on alternation depth instead of nested depth, and establish its connection with the hierarchy of properties. In the next Section we demonstrate that this classification directly reflects the hardness of the verification problem for particular properties.

Let us define hierarchies Σ_i^{LTL} and Π_i^{LTL}, which reflect alternations of Until and Release operators in formulas. We use the Σ / Π notation since the way the hierarchy is defined strongly resembles the quantifier alternation hierarchy of first-order logic formulas or fixpoints alternation hierarchy of μ-calculus formulas [12].

Definition 3.

The class $\Sigma_0^{LTL} = \Pi_0^{LTL}$ is the least set containing all atomic propositions and closed under the application of boolean and Next operators.

The class Σ_{i+1}^{LTL} is the least set containing Π_i^{LTL} and closed under the application of conjunction, disjunction, Next and Until operators.

The class Π_{i+1}^{LTL} is the least set containing Σ_i^{LTL} and closed under the application of conjunction, disjunction, Next and Release operators.

Theorem 4. *A property that is specifiable by LTL is a guarantee (safety, persistence, recurrence respectively) property if and only if it is specifiable by a formula from the class Σ_1^{LTL} ($\Pi_1^{LTL}, \Sigma_2^{LTL}, \Pi_2^{LTL}$ respectively) (see Fig. 1).*

Proof: One can transform any guarantee (safety, persistence, recurrence respectively) formula into an equivalent Σ_1^{LTL} ($\Pi_1^{LTL}, \Sigma_2^{LTL}, \Pi_2^{LTL}$ respectively) formula using identities $\alpha\,\mathbf{R}\,\beta \equiv \mathbf{G}\,\beta \vee (\beta\,\mathbf{U}\,(\alpha \wedge \beta))$, $\alpha\,\mathbf{U}\,\beta \equiv \mathbf{F}\,\beta \wedge (\beta\,\mathbf{R}\,(\alpha \vee \beta))$, $\mathbf{F}\,\alpha \equiv \mathit{true}\,\mathbf{U}\,\alpha$, $\mathbf{G}\,\alpha \equiv \mathit{false}\,\mathbf{R}\,\alpha$ and standard identities for pushing negation "inward". The rest is a consequence of Theorem 3. □

Theorem 5. *A property is specifiable by LTL if and only if it is specifiable by a positive boolean combination of Σ_2^{LTL} and Π_2^{LTL} formulas.*

Proof: Follows from the previous Theorem and the fact that every reactivity property is a positive boolean combination of persistence and recurrence properties [23]. □

Thus we have that the type of a formula can be derived from its alternation depth. As a consequence we obtain the following, quite surprising fact:

Corollary 1. *Both Σ_i^{LTL} and Π_i^{LTL} hierarchies collapse in the sense that every LTL formula is specifiable both by a Σ_3^{LTL} and Π_3^{LTL} formula.*

3 Model Checking and Hierarchy of Properties

The model checking problem is to determine for a given reactive system K and a temporal formula φ whether the system satisfies the formula. A common approach to model checking of finite state systems and LTL formulas is to construct an automaton $A_{\neg\varphi}$ for the negation of the property and to model the system as an automaton K. The product automaton $K \times A_{\neg\varphi}$ is then checked for non-emptiness. The product automaton is a nondeterministic infinite occurrence Büchi automaton. For the formal definition of the problem and detailed description of the algorithm we refer to [6].

Our aim is to analyse the complexity of the non-emptiness check depending on the type of the verified property. As the complexity of the non-emptiness check is determined by attributes of an automaton, the question is whether for different types of formulas one can construct different types of automata. We give a comprehensive answer to this question in this section. In the next section we demonstrate how the complexity of the non-emptiness check varies depending on the type of automata.

To classify nondeterministic infinite occurences Büchi automata we adopt the criteria proposed by Bloem, Ravi, and Somenzi [1]. They differentiate *general*, *weak*, and *terminal* automata according to the following restrictions posed on their transition functions:

- *general*: no restriction
- *weak*: there exists a partition of the set Q into components Q_i and an ordering $\leq$ on these sets, such that for each $q \in Q_i, p \in Q_j$, if $\exists a \in \Sigma : q \in \delta(p, a)$ then $Q_i \leq Q_j$. Moreover for each Q_i, $Q_i \cap \alpha = \emptyset$, in which case Q_i is a rejecting component, or $Q_i \subseteq \alpha$, in which case Q_i is an accepting component.
- *terminal*: for each $q \in \alpha, a \in \Sigma$ it holds $\delta(q, a) \neq \emptyset$ and $\delta(q, a) \subseteq \alpha$.

Each transition of a weak automaton leads to a state in either the same or lower component. Consequently each run of a weak automaton gets eventually trapped within one component. The run is accepting iff this component is accepting. The transition function of a terminal automaton is even more restricted – once a run of a terminal automaton reaches an accepting state the run is accepting regardless of the suffix. Terminal and weak automata are jointly called *specialised* automata. It shows up that the classes of properties specifiable by weak and terminal automata coincide with classes of the hierarchy.

Theorem 6. *A property P specifiable by automata is a guarantee (persistence) property if and only if it is specifiable by a terminal (weak) automaton.*

Proof: A terminal automaton can be, thanks to the property of its transition function, determinised by a power-set construction. The resulting automaton can be viewed as a deterministic occurrence Büchi automaton. On the other hand, each deterministic occurrence Büchi automaton can be easily transformed into equivalent terminal automaton by a copy construction. Since deterministic occurrence Büchi automata recognise guarantee properties (Theorem 2), so do terminal automata.

In a similar way a connection between weak automata and nondeterministic infinite occurences co-Büchi automata can be established. The transformation can be found in [21]. □

Theorem 6 raises a natural question whether and how effectively one can construct for a given guarantee (persistence) formula the corresponding terminal (weak) automaton. A construction of an automaton for an LTL formula was first proposed by Wolper, Vardi and Sistla [33]. This basic construction has been improved in several papers ([16, 28, 13, 20]) where various heuristics have been used to produce automaton as small and as ”weak” as possible. Although these heuristics are quite sophisticated, they do not provide any insight into the relation between the formula and the ”weakness” of the resulting automaton.

We present a new modification of the original construction which yields for a formula from the class Σ_1^{LTL} and Σ_2^{LTL} a specialised automaton. Similar constructions were independently used by Schneider [26].

Theorem 7. *For every Σ_1^{LTL} (Σ_2^{LTL}) formula φ we can construct a terminal (weak) automaton accepting the property defined by φ.*

Proof: States of the automaton are sets of subformulas of the formula φ. The transition function is constructed in such a way that the following invariant is valid: if the automaton is in a state S then the remaining suffix of the word should satisfy all formulas in S. The acceptance condition is used to enforce the fulfillment of Until operators. For Σ_1^{LTL} and Σ_2^{LTL} formulas the acceptance condition can be simplified thanks to the special structure of alternation of Until and Release operators in the formula.

Let φ be a Σ_1^{LTL} or Σ_2^{LTL} formula over the set AP of atomic propositions. We define the automaton $A_\varphi = \langle \Sigma, Q, q_{start}, \delta, \alpha \rangle$ as follows:

$\Sigma = 2^{AP}$

$Q = q_{start} \cup Q'$, where q_{start} is a special initial state and Q' is the set of all subsets of $sub(\varphi) = \{\psi \mid \psi \text{ is a subformula of } \varphi\} \cup \{p, \neg p \mid p \in AP\}$ that do not have any propositional inconsistency, i.e. $S \in Q'$ if $S \subseteq sub(\varphi)$ and
- $p \in S \Leftrightarrow \neg p \notin S$ for all $p \in AP$
- $\psi_1 \wedge \psi_2 \in S \Rightarrow \psi_1 \in S$ and $\psi_2 \in S$
- $\psi_1 \vee \psi_2 \in S \Rightarrow \psi_1 \in S$ or $\psi_2 \in S$

δ : Let $S, S' \in Q$, then $S' \in \delta(S, A)$ if $A = S' \cap AP$ and
- $\mathbf{X}\psi \in S \Rightarrow \psi \in S'$
- $\psi_1 \mathbf{U} \psi_2 \in S \Rightarrow \psi_2 \in S$ or ($\psi_1 \in S$ and $\psi_1 \mathbf{U} \psi_2 \in S'$)
- $\psi_1 \mathbf{R} \psi_2 \in S \Rightarrow \psi_1 \wedge \psi_2 \in S$ or ($\psi_2 \in S$ and $\psi_1 \mathbf{R} \psi_2 \in S'$)
- $S \in \alpha \Rightarrow S' \in \alpha$
- $S = q_{start} \Rightarrow \varphi \in S'$

α : accepting set respects the type of the property:
- if φ is a Σ_1^{LTL} formula then $\alpha = \{S \in Q \mid S \subseteq AP\}$
- if φ is a Σ_2^{LTL} formula then $\alpha = \{S \in Q \mid \text{there is no Until formula in } S\}$

To verify the correctness of the construction we have to show that:

- the automaton A_φ accepts the property defined by φ, that is $w \models \varphi \Leftrightarrow A_\varphi$ accepts w. This can be proved by a structural induction on φ.
- the automaton A_φ is terminal (weak). This can be easily seen from the way the transition function is defined due to the fact that the formula is from the class Σ_1^{LTL} (Σ_2^{LTL}). The partition of the set Q for the weak automaton is $Q_2 = Q \smallsetminus \alpha, Q_1 = \alpha$.

The major difference in comparison to the original construction is in the way the acceptance condition is defined. The transition function is modified in order to respect the intended partition of states. □

4 Non-Emptiness Algorithms

In the previous section we showed that we can effectively construct specialised automata for formulas from lower classes of the hierarchy. Since the verified system K can be modelled as an automaton without acceptance conditions, the type of the product automaton is determined entirely by the type of the automaton $A_{\neg\varphi}$, that is even the product automaton is specialised. In this section we study algorithms for its non-emptiness check.

Every Büchi automaton can be represented as an oriented graph with vertices corresponding to states and edges corresponding to the transition function of the automaton. Every infinite path in the graph corresponds to a run of the automaton. The question whether there exists an accepting run thus can be reduced to the question whether there exists a path with some special properties in the graph. The required properties for particular automata are listed below.

Terminal non-emptiness check = reachability of an accepting state
Once a terminal automaton reaches an accepting state, it accepts the whole word. The language recognised by a terminal automaton is non-empty iff an accepting state is reachable.

Weak non-emptiness check = reachability of a ”fully accepting” cycle
States of a weak automaton are partitioned into components and therefore states from each cycle in the graph are either all accepting (the cycle is fully accepting) or all non-accepting. The language recognised by a weak automaton is non-nonempty iff a fully accepting cycle is reachable.

General non-emptiness check = reachability of an accepting cycle
In general case, cycles in the graph may contain accepting as well as non-accepting states. The language is non-nonempty iff a cycle containing at least one accepting state is reachable.

Terminal, weak, and general automata correspond to guarantee, persistence and reactivity properties respectively. One may ask whether for other types of properties and their corresponding automata the non-emptiness check reduces to different graph problems. This is not the case. A fully accepting cycle detection and an accepting cycle detection are unavoidable even for the safety and recurrence properties respectively. Thus from the model checking point of view the hierarchy splits into three stripes, as indicated on Fig. 1.

Model checking algorithms are usually divided into *explicit* and *symbolic*, according to the used representation of a considered product automaton. In the explicit representation each state is manipulated individually. The symbolic representation works with sets of states which are typically represented by binary decision diagrams (BDDs) [2, 6].

4.1 Explicit Algorithms

With the explicit representation, the product automaton is represented by adjacency lists. The algorithm traverses the graph and visits individual states.

Terminal non-emptiness: any complete traversal of the state space suffices.

Weak non-emptiness: can be solved efficiently by a depth first search (DFS). A fully accepting cycle is detected by DFS when an accepting state that is currently on the DFS stack it reached.

General non-emptiness: the most efficient explicit algorithm for this case is the nested DFS algorithm [8, 17] which traverses the state space using DFS and detects accepting cycles by nested DFS executed from accepting states.

The asymptotic time complexity of all above mentioned algorithms is the same though they differ on constant factors. The application of algorithms for specialised automata brings along several benefits as for instance a possibility for employing “guided search” heuristics [10]. The partial-order reduction can be employed more directly in the case of the simple DFS than in the case of the nested DFS [17]. For terminal automata the algorithm is not tied up with P-complete DFS [25] and hence it allows for better distribution [29]. Some of these benefits were experimentally demonstrated by Edelkamp, Lafuente and Leue [10]. They extended the model checker SPIN by a non-emptiness algorithm which to a certain extent takes the type of an automaton into consideration.

4.2 Symbolic Algorithms

Symbolic algorithms work with sets of states represented by BDDs [3] and perform basic set operations (union, intersection) and an *image* operation. The symbolic approach has been originally used for branching time logic model checking. A symbolic approach to LTL was initially based on the transformation to fair CTL model checking [7] and only recently it was reformulated directly for LTL [18]. We review this method with respect to weak and terminal automata.

Symbolic algorithm for the non-emptiness check is based on the computation of two fixpoints:

$$\begin{aligned} \mathit{Reachability}(S) &= \mu Z.(S \cup \mathit{image}(Z)) \\ \mathit{Elimination}(S) &= \nu Z.(S \cap \mathit{image}(Z)) \end{aligned}$$

The function $\mathit{Reachability}(S)$ computes the set of states that are reachable from the set S. The function $\mathit{Elimination}(S)$ computes the set of all states q, for which either q lies on a cycle in S or $q \in S$ and q is reachable in S from a cycle in S (the computation is performed by successive removal of states that do not have predecessors in S). There are several possibilities how to formulate an algorithm for non-emptiness check of general automata [15]. Algorithms displayed in Fig. 2 correspond to the “One Way Catch Them Young” strategy.

It is conventional to analyse the complexity of symbolic algorithms with respect to the number of symbolic steps (that is the number of union, intersection, and image computations). The complexity of the algorithm for general automata is quadratic, since it involves computation of nested fixpoints. The algorithms for weak and terminal automata perform only linear number of symbolic steps as they involve only simple fixpoint computations. Thus for symbolic algorithms it is even asymptotically more efficient to use specialised algorithms. Moreover, the arguments concerning heuristics and distributed computations stated for the explicit case hold for the symbolic case as well. Results of experiments performed by Bloem, Ravi, and Somenzi [1] confirm these claims.

5 Determining the Type of Formula

In the previous sections we showed that the verification of an LTL specifiable property can be more efficient if the resulting product automaton is weak or

```
proc General_Nonemptiness(A)
    $S := Reachability(q_0)$;
    $old := \emptyset$;
    while $(S \neq old)$ do
            $old := S$;
            $S := Reachability(S \cap \alpha)$;
            $S := Elimination(S)$;
    od
    return $S \neq \emptyset$;
end
```

```
proc Weak_Nonemptiness(A)
    $S := Reachability(q_0)$;
    $S := Elimination(S \cap \alpha)$;
    return $S \neq \emptyset$;
end

proc Terminal_Nonemptiness(A)
    $S := Reachability(q_0) \cap \alpha$;
    return $S \neq \emptyset$;
end
```

Fig. 2. Symbolic non-emptiness algorithms, their input is a Büchi automaton A; *Weak_Nonemptiness* and *Terminal_Nonemptiness* work correctly for weak and terminal automata respectively.

terminal. In other words, it is preferable if the property we want to verify is formulated as a recurrence or safety formula[1]. The problem is how to determine the type of a property. We remind the reader the formula $\mathbf{G}\, p \vee (p\, \mathbf{U}\, q)$ (see Section 2.2) specifying a safety property although it is not a safety formula.

Theorem 8 ([23]). *It is decidable whether a given deterministic Streett automaton specifies a property of type κ.*

Corollary 2. *It is decidable whether a given LTL formula φ specifies a property of the type κ.*

Proof: For a given formula φ we construct a nondeterministic Büchi automaton A_φ accepting models of φ [32]. The determinisation of the automaton yields an equivalent deterministic Streett automaton [21] . Theorem 8 allows us to determine the type of the property. □

Unfortunately, the decision procedure sketched in the proof of Corollary 2 is exponential due to necessary determinisation of the automaton. Moreover, Sistla [27] has shown that even the problem of deciding whether a given formula specifies a safety property is PSPACE-complete. This indicates that we cannot hope for any much more efficient algorithm for deciding whether a given LTL formula φ specifies a property of the type κ.

Thus a natural objection to model checking strategy based on the precise classification of the property type is that the effort needed for determining the type prevails over the advantage gained by the use of a specialised algorithm for the non-emptiness check. We advocate this approach as follows.

Although the algorithm for determining the type of formula has very high worst-case complexity it still may be the case that it can be well accomplished. We call to mind that formulas are usually quite "short". Moreover, it is typical to

[1] Please remember that in the verification process the negation of the formula is translated into an automaton. Therefore recurrence and safety formulas are translated into weak and terminal automata respectively.

make many tests for one fixed formula during the system development process. In such a case, the work needed for determining the type of the formula is amortised over its verification.

One can benefit from the presented classification of properties even in some other ways. First, when translating a formula into an automaton, various heuristics can be employed to generate an automaton “as weak as possible” (this approach has been used in [28, 13]). Secondly, when formulating a desired property as an LTL formula one can take into consideration the Until-Release hierarchy and avoid the combinations of temporal operators resulting in a hard non-emptiness check.

To stress the significance of the specialised algorithms we have studied the Specification Patterns System [9] that is a collection of the most often verified properties. It shows up that most of the properties are either of safety (41%) or recurrence (54%) type and thus in most cases the resulting automaton is either terminal or weak. Moreover, algorithms for specialized automata can be more effectively transformed to distributed ones [4].

6 Conclusions

The contribution of the paper is twofold. First, it provides a new classification of temporal properties through deterministic ω-automata and through the Until-Release hierarchy.

Secondly, the paper introduces a new classification of LTL properties with respect to the complexity of their verification problem. Linear temporal properties are sorted into three major classes: reactivity, recurrence and safety. The verification of safety properties reduces to the non-emptiness check of terminal automata that can be solved either by searching the state space (the explicit representation) or by simple fixpoint computation (the symbolic representation). The verification of recurrence properties reduces to the non-emptiness check of weak automata that can be solved by simple depth-first search or by simple fixpoint computation. In general, simpler non-emptiness check algorithms are not only more time efficient but also allow for more additional time / space saving methods. The paper cites several experimental works confirming these claims and observations.

References

1. R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Proc. Computer Aided Verification*, volume 1633 of *LNCS*, pages 222–235. Springer, 1999.
2. R.E. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computers*, volume C-35(8), pages 677 – 691, 1986.
3. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
4. I. Černá and R. Pelánek. Distributed explicit fair cycle detection. In *10th International SPIN Workshop on Model Checking of Software*, Portland, Oregon, 2003.

5. E. Y. Chang, Z. Manna, and A. Pnueli. Characterization of temporal property classes. In *Proc. Automata, Languages and Programming*, volume 623 of *LNCS*, pages 474–486. Springer, 1992.
6. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.
7. E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *Proc. Computer Aided Verification*, volume 818 of *LNCS*, pages 415–427. Springer, 1994.
8. C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.
9. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In *Proc. Workshop on Formal Methods in Software Practice*, pages 7–15. ACM Press, 1998.
10. S. Edelkamp, A. L. Lafuente, and S. Leue. Directed explicit model checking with HSF-SPIN. In *Proc. SPIN workshop*, volume 2057 of *LNCS*, pages 57–79. Springer, 2001.
11. E. A. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B, Formal Models and Semantics*, pages 995–1072. North-Holland Publishing Company, 1990.
12. E. A. Emerson and C. L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proc. IEEE Symposium on Logic in Comuter Science*, pages 267 – 278. Computer Society Press, 1986.
13. K. Etessami and G. J. Holzmann. Optimizing Büchi automata. In *Proc. CONCUR*, volume 1877 of *LNCS*, pages 153–167. Springer, 2000.
14. K. Etessami and T. Wilke. An Until hierarchy for temporal logic. In *Proc. IEEE Symposium on Logic in Computer Science*, pages 108–117. Computer Society Press, 1996.
15. K. Fisler, R. Fraer, G. Kamhi Y. Vardi, and Zijiang Yang. Is there a best symbolic cycle-detection algorithm? In *Proc. Tools and Algorithms for Construction and Analysis of Systems*, volume 2031 of *LNCS*, pages 420–434. Springer, 2001.
16. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.
17. G. J. Holzmann, D. Peled, and M. Yannakakis. On nested depth first search. In *Proc. SPIN Workshop*, pages 23–32. American Mathematical Society, 1996.
18. Y. Kesten, A Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In *Proc. Automata, Languages and Programming*, volume 1443 of *LNCS*, pages 1–16. Springer, 1998.
19. A. Kučera and J. Strejček. The stuttering principle revisited: On the expressiveness of nested X and U operators in the logic LTL. In *Proc. Computer Science Logic*, volume 2471 of *LNCS*, pages 276–291. Springer, 2002.
20. O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.
21. C. Löding. Methods for the transformation of omega-automata: Complexity and connection to second order logic. Master's thesis, Christian-Albrechts-University of Kiel, 1998.
22. C. Löding and W. Thomas. Alternating automata and logics over infinite words. In *Proc. IFIP International Conference on Theoretical Computer Science*, volume 1872 of *LNCS*, pages 521–535. Springer, 2000.
23. Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proc. ACM Symposium on Principles of Distributed Computing*, pages 377–410. ACM Press, 1990.

24. D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267 – 276, 1987.
25. J.H. Reif. Depth-first search is inherrently sequential. *Information Processing Letters*, 20(5):229–234, 1985.
26. K. Schneider. Improving automata generation for linear temporal logic by considering the automaton hierarchy. In *Proc. Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2250 of *LNCS*, pages 39–54. Springer, 2001.
27. A. P. Sistla. Safety, liveness, and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–512, 1994.
28. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proc. Computer Aided Verification*, volume 1855 of *LNCS*, pages 248–263. Springer, 2000.
29. U. Stern and D.L. Dill. Parallelizing the Murφ verifier. In *Proc. Computer Aided Verification*, volume 1254 of *LNCS*, pages 256–267. Springer, 1997.
30. D. Therien and T. Wilke. Nesting Until and Since in linear temporal logic. In *Proc. Symposium on Theoretical Aspects of Computer Science*, volume 2285 of *LNCS*, pages 455–464. Springer, 2002.
31. W. Thomas. Languages, automata and logic. In *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.
32. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. IEEE Symposium on Logic in Computer Science*, pages 322–331. Computer Society Press, 1986.
33. P. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about inifinite computation paths. In *Proc. Symp. on Foundations of Computer Science*, pages 185 – 194, Tuscon, 1983.

Publications in the FI MU Report Series are in general accessible via WWW and anonymous FTP:

```
http://www.fi.muni.cz/informatics/reports/
ftp  ftp.fi.muni.cz (cd pub/reports)
```

Copies may be also obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic