Characteristic Patterns for LTL*

Antonín Kučera and Jan Strejček

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
{tony,strejcek}@fi.muni.cz

Abstract. We give a new characterization of those languages that are definable in fragments of LTL where the nesting depths of X and U modalities are bounded by given constants. This brings further results about various LTL fragments. We also propose a generic method for decomposing LTL formulae into an equivalent disjunction of "semantically refined" LTL formulae, and indicate how this result can be used to improve the functionality of existing LTL model-checkers.

1 Introduction

Linear temporal logic (LTL) [6] is a popular formalism for specifying properties of (concurrent) programs. The syntax of LTL is given by the following abstract syntax equation:

$$\varphi ::= \texttt{tt} \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2$$

Here a ranges over a countable set $\Lambda = \{a, b, c, \ldots\}$ of *letters*. We also use $\mathsf{F}\varphi$ to abbreviate $\texttt{tt}\,\mathsf{U}\,\varphi$, and $\mathsf{G}\varphi$ to abbreviate $\neg\mathsf{F}\neg\varphi$. The set of all letters which appear in a given formula φ is denoted $\Lambda(\varphi)$.

The semantics of LTL is defined in terms of languages over infinite words. An *alphabet* is a finite set $\Sigma \subseteq \Lambda$. An ω*-word* over Σ is an infinite sequence $\alpha = \alpha(0)\alpha(1)\alpha(2)\ldots$ of letters from Σ. The set of all finite words over Σ is denoted by Σ^*, and the set of all ω-words by Σ^ω. The length of a given $u \in \Sigma^*$ is denoted $|u|$. In the rest of this paper we use $a, b, c, \ldots$ to range over Σ, $u, v, \ldots$ to range over Σ^*, and $\alpha, \beta, \ldots$ to range over Σ^ω. For every $i \in \mathbb{N}_0$ we denote by α_i the ith suffix of α, i.e., the word $\alpha(i)\alpha(i+1)\ldots$.

Let Σ be an alphabet. The *validity* of a formula φ for $\alpha \in \Sigma^\omega$ is defined as follows:

$$\begin{array}{lll}
\alpha \models \texttt{tt} & & \\
\alpha \models a & \text{iff} & a = \alpha(0) \\
\alpha \models \neg\varphi & \text{iff} & \alpha \not\models \varphi \\
\alpha \models \varphi_1 \wedge \varphi_2 & \text{iff} & \alpha \models \varphi_1 \wedge \alpha \models \varphi_2 \\
\alpha \models \mathsf{X}\varphi & \text{iff} & \alpha_1 \models \varphi \\
\alpha \models \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff} & \exists i \in \mathbb{N}_0 : \alpha_i \models \varphi_2 \wedge \forall\, 0 \le j < i : \alpha_j \models \varphi_1
\end{array}$$

For each alphabet Σ, a formula φ defines the ω-language $L^\Sigma_\varphi = \{\alpha \in \Sigma^\omega \mid \alpha \models \varphi\}$.

* This work has been supported by GAČR, grant No. 201/03/1161.

For every LTL formula φ and every modal operator $M \in \{\mathsf{X}, \mathsf{U}\}$ we define the *nesting depth* of M in φ, denoted M-*depth*(φ), inductively as follows (Y ranges over unary operators $\{\neg, \mathsf{X}\}$ and Z ranges over binary operators $\{\wedge, \mathsf{U}\}$).

$$M\text{-}depth(\mathtt{tt}) = M\text{-}depth(a) = 0$$
$$M\text{-}depth(Y\varphi) = \begin{cases} M\text{-}depth(\varphi) + 1 & \text{if } M = Y, \\ M\text{-}depth(\varphi) & \text{otherwise.} \end{cases}$$
$$M\text{-}depth(\varphi_1\, Z\, \varphi_2) = \begin{cases} \max\{M\text{-}depth(\varphi_1), M\text{-}depth(\varphi_2)\} + 1 & \text{if } M = Z, \\ \max\{M\text{-}depth(\varphi_1), M\text{-}depth(\varphi_2)\} & \text{otherwise.} \end{cases}$$

For all $m, n \in \mathbb{N}_0 \cup \{\infty\}$, the symbol LTL$(\mathsf{U}^m, \mathsf{X}^n)$ denotes the set of all LTL formulae φ such that U-*depth*$(\varphi) \leq m$ and X-*depth*$(\varphi) \leq n$. To simplify our notation, we omit the "∞" superscript, and if m or n equals 0, then we omit the symbol U^m or X^n in LTL$(\mathsf{U}^m, \mathsf{X}^n)$, respectively. Hence, e.g., $\mathrm{LTL}(\mathsf{U}^3, \mathsf{X})$ is a shorthand for LTL$(\mathsf{U}^3, \mathsf{X}^\infty)$.

A lot of research effort has been invested into characterizing the expressive power of LTL and its fragments. A concise survey covering basic results about LTL expressiveness can be found in [1]. A more recent survey [8] contains also results concerning some of the LTL fragments. In this paper, we give a new characterization of ω-languages that are definable in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ for given $m, n \in \mathbb{N}_0$.[1] Roughly speaking, for each alphabet Σ and all $m, n \in \mathbb{N}_0$ we design a finite set of (m, n)-*patterns*[2], where each (m, n)-pattern is a finite object representing an ω-language over Σ so that the following conditions are satisfied:

- Each $\alpha \in \Sigma^\omega$ is represented by exactly one (m, n)-pattern (consequently, the sets of ω-words represented by different patterns are disjoint).
- ω-words which are represented by the same (m, n)-pattern cannot be distinguished by any formula of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$.
- For each (m, n)-pattern p we can effectively construct a formula $\psi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ so that for each $\alpha \in \Sigma^\omega$ we have that $\alpha \models \psi$ if and only if α is represented by the pattern p.

Thus, the semantics of each formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ is fully characterized by a finite subset of (m, n)-patterns, and vice versa. Intuitively, (m, n)-patterns represent *exactly* the information about ω-words which determines the (in)validity of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae. The patterns are defined inductively on m, and the inductive step brings some insight into what is actually gained (i.e., what new properties can be expressed) by increasing the nesting depth of U by one.

Characteristic patterns can be used as a tool for proving further results about the logic LTL and its fragments. In particular, they can be used to construct a short proof of a (somewhat simplified) form of stutter invariance of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ languages introduced in [3]. This, in turn, allows to construct simpler proofs for some of the results presented in [3] (like, e.g., the strictness of the $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$, $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, and

[1] The expressiveness of these fragments has already been studied in [3]. In particular, it has been proven that the classes of languages definable by two syntactically incomparable fragments of this form are also incomparable.

[2] Let us note that (m, n)-patterns have nothing to do with the forbidden patterns of [8].

$\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ hierarchies). An interesting question (which is left open) is whether one could use characteristic patterns to demonstrate the decidability of the problem if a given ω-regular language L is definable in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$ for a given m.

Another application area for characteristic patterns is LTL model-checking. We believe that this is actually one of the most interesting parts of our work, and therefore we explain the idea in greater detail.

An instance of the LTL model-checking problem is a system and an LTL formula (called "specification formula") which defines desired properties of the system. The question is whether all runs of the system satisfy the formula. This problem can dually be reformulated as follows: for a given system and a given formula φ (representing the negation of the desired property), decide whether the system has at least one run satisfying φ. Characteristic patterns can be used to decompose a given LTL formula φ into an equivalent disjunction $\varphi \equiv \psi_1 \vee \ldots \vee \psi_n$ of mutually exclusive formulae (i.e., we have $\psi_i \Rightarrow \bigwedge_{j \neq i} \neg\psi_j$ for each i). Roughly speaking, each ψ_i corresponds to one of the patterns which define the semantics of φ. Hence, the ψ_i formulae are not necessarily smaller or simpler than φ from the syntactical point of view. The simplification is on semantical level, because each ψ_i "cuts off" a dedicated subset of runs that satisfy φ. Another advantage of this method is its scalability—the patterns can be constructed also for those n and m that are larger than the nesting depths of X and U in φ. Thus, the patterns can be repeatedly "refined", which corresponds to decomposing the constructed ψ_i formulae. Another way of refining the patterns is enlarging the alphabet Σ.

The decomposition technique enables the following model-checking strategy: First try to model-check φ. If this does not work (because of, e.g., memory overflow), then decompose φ into $\psi_1 \vee \ldots \vee \psi_n$ and try to model-check the $\psi_1, \ldots, \psi_n$ formulae. This can be done sequentially or even in parallel. If at least one subtask produces a positive answer, we are done (there is a "bad" run). Similarly, if all subtasks produce a negative answer, we are also done (there is no "bad" run). Otherwise, we go on and decompose those ψ_i for which our model-checker did not manage to answer.

Obviously, the introduced strategy can only lead to better results than checking just φ, and it is completely independent of the underlying model-checker. Moreover, some new and relevant information is obtained even in those cases when this strategy does not lead to a definite answer—we know that if there is a bad run, it must satisfy some of the subformulae we did not manage to model-check. The level of practical usability of the above discussed approach can only be measured by outcomes of practical experiments which are beyond the scope of this (mainly theoretical) paper.[3] Here we concentrate on providing basic results and identifying promising directions for applied research.

Let us note that similar decomposition techniques have been proposed in [5] and [9]. In [5], a specification formula of the form $\mathsf{G}\varphi$ is decomposed into a set of formulae $\{\mathsf{G}(x{=}v_i \Rightarrow \varphi) \mid v_i \text{ is in the range of the variable } x\}$. This decomposition technique has been implemented in the SMV system together with methods aimed at reducing the range of x. This approach has then been used for verification of specific types of infinite-state systems (see [5] for more details). In [9], a given specification formula φ is model-checked as follows: First, a finite set of formulae $\psi_1, \ldots, \psi_n$ of the form $\psi_i = \mathsf{G}(x{\neq}v_0 \Rightarrow x{=}v_i)$ is constructed such that the verified system satisfies $\psi_1 \vee$

[3] A practical implementation of the method is under preparation.

$\ldots \vee \psi_n$. The formulae $\psi_1, \ldots, \psi_n$ are either given directly by the user, or constructed automatically using methods of static analysis. The verification problem for φ is then decomposed into the problems of verifying the formulae $\psi_i \Rightarrow \varphi$. Using this approach, the peak memory in model checking has been reduced by 13–25% in the three case studies included in the paper.

It is worth mentioning that characteristic patterns could potentially be used also in a different way: we could first extract *all* patterns that can be exhibited by the system, and then check whether there is one for which φ holds. Unfortunately, the set of all patterns exhibited by a given system seems to be computable only in restricted cases, e.g., when the system has just a single path (see [4] for more information about model checking of these systems and [2] for a pattern-based algorithm).

The paper is organized as follows. Section 2 provides a formal definition of (m, n)-patterns together with basic theorems. Section 3 is devoted to detailed discussion of the indicated decomposition technique. Conclusions and directions for future research are given in Section 4. Other applications of characteristic patterns in the area of LTL model checking as well as all proofs (which were omitted due to space constraints) can be found in [2].

2 Characteristic patterns

To get some intuition about characteristic patterns, let us first consider the set of patterns constructed for the alphabet $\Sigma = \{a, b, c\}$, $m = 1$, and $n = 0$ (as we shall see, the m and n correspond to the nesting depths of U and X, respectively). Let $\alpha \in \Sigma^\omega$ be an ω-word. A letter $\alpha(i)$ is *repeated* if there is $j < i$ such that $\alpha(j) = \alpha(i)$. The $(1, 0)$-pattern of α, denoted $pat(1, 0, \alpha)$, is the finite word obtained from α by deleting all repeated letters (for reasons of consistent notation, this word is written in parenthesis). For example, if $\alpha = \underline{a}a\underline{b}bbaababab\underline{c}abccacab\ldots$, then $pat(1, 0, \alpha) = (abc)$. So, the set of all $(1, 0)$-patterns over the alphabet $\{a, b, c\}$, denoted $Pats(1, 0, \{a, b, c\})$, has exactly 15 elements which are the following:

$$(abc), (acb), (bac), (bca), (cab), (cba), (ab), (ba), (ac), (ca), (bc), (cb), (a), (b), (c)$$

Thus, the set $\{a, b, c\}^\omega$ is divided into 15 disjoint subsets, where each set consists of all ω-words that have a given pattern. It remains to explain why these patterns are interesting. The point is that $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ formulae can actually express just the order of non-repeated letters. For example, the formula $a \, \mathsf{U} \, b$ says that either the first non-repeated letter is b, or the first non-repeated letter is a and the second one is b. So, this formula holds for a given $\alpha \in \{a, b, c\}^\omega$ iff $pat(1, 0, \alpha)$ is equal to (b), (ba), (bc), (bac), (bca), (ab), or (abc). We claim (and later also prove) that ω-words of $\{a, b, c\}^\omega$ which have the same $(1, 0)$-pattern cannot be distinguished by *any* $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ formula. So, each $\varphi \in \mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$, where $\Lambda(\varphi) \subseteq \{a, b, c\}$, is fully characterized by a subset of $Pats(1, 0, \{a, b, c\})$. Moreover, for each $p \in Pats(1, 0, \{a, b, c\})$ we can construct an $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ formula φ_p such that for every $\alpha \in \{a, b, c\}^\omega$ we have that $\alpha \models \varphi_p$ iff $pat(1, 0, \alpha) = p$. For example, $\varphi_{(abc)} \; = \; a \; \wedge \; (a \, \mathsf{U} \, b) \; \wedge \; ((a \vee b) \, \mathsf{U} \, c)$.

To indicate how this can be generalized to larger m and n, we show how to extract a $(2, 0)$-pattern from a given $\alpha \in \{a, b, c\}^\omega$. We start by considering an infinite word

over the alphabet $Pats(1, 0, \{a, b, c\})$ constructed as follows:

$$pat(1, 0, \alpha_0)\ pat(1, 0, \alpha_1)\ pat(1, 0, \alpha_2)\ pat(1, 0, \alpha_3) \ldots$$

For example, for $\alpha = aabaca^\omega$ we obtain the sequence $(abc)(abc)(bac)(ac)(ca)(a)^\omega$. The pattern $pat(2, 0, \alpha)$ is obtained from the above sequence by deleting repeated letters (realize that now we consider the alphabet $Pats(1, 0, \{a, b, c\})$). Hence, $pat(2, 0, \alpha) = ((abc)(bac)(ac)(ca)(a))$. Similarly as above, it holds that those ω-words of $\{a, b, c\}^\omega$ which have the same $(2, 0)$-pattern cannot be distinguished by any $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$ formula. Moreover, for each $p \in Pats(2, 0, \{a, b, c\})$ we can construct an $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$ formula φ_p such that for every $\alpha \in \{a, b, c\}^\omega$ we have that $\alpha \models \varphi_p$ iff $pat(2, 0, \alpha) = p$.

Formally, we consider *every* finite sequence of $(1, 0)$-patterns, where no $(1, 0)$-pattern is repeated, as a $(2, 0)$-pattern. This makes the inductive definition simpler, but in this way we also introduce patterns that are not "satisfiable". For example, there is obviously no $\alpha \in \{a, b, c\}^\omega$ such that $pat(2, 0, \alpha) = ((a)(ab))$.

The last problem we have yet not addressed is how to deal with the X operator. First note that the X operator can be pushed inside using the following rules (see, e.g., [1]):

$$\mathsf{X}\mathtt{tt} \equiv \mathtt{tt} \quad \mathsf{X}\neg\varphi \equiv \neg\mathsf{X}\varphi \quad \mathsf{X}(\varphi_1 \wedge \varphi_2) \equiv \mathsf{X}\varphi_1 \wedge \mathsf{X}\varphi_2 \quad \mathsf{X}(\varphi_1 \,\mathsf{U}\, \varphi_2) \equiv \mathsf{X}\varphi_1 \,\mathsf{U}\, \mathsf{X}\varphi_2$$

Note that this transformation does not change the nesting depth of X. Hence, we can safely assume that the X operator occurs in LTL formulae only within subformulae of the form $\mathsf{X}\mathsf{X}\ldots\mathsf{X}a$. This is the reason why we can handle the X operator in the following way: the set $Pats(m, n, \Sigma)$ is defined in the same way as $Pats(m, 0, \Sigma)$. The only difference is that we start with the alphabet Σ^{n+1} instead of Σ.

Definition 1. *Let Σ be an alphabet. For all $m, n \in \mathbb{N}_0$ we define the set $Pats(m, n, \Sigma)$ inductively as follows:*

- $Pats(0, n, \Sigma) = \{w \in \Sigma^* \mid |w| = n+1\}$
- $Pats(m+1, n, \Sigma) = \{(p_1 \ldots p_k) \mid k \in \mathbb{N},\ p_1, \ldots, p_k \in Pats(m, n, \Sigma),\ p_i \neq p_j \text{ for } i \neq j\}$

The size of $Pats(m, n, \Sigma)$ and the size of its elements are estimated in our next lemma (the proof follows directly from definitions).

Lemma 2. *For every $i \in \mathbb{N}_0$, let us define the function $fac_i : \mathbb{N}_0 \to \mathbb{N}_0$ inductively as follows: $fac_0(x) = x$, $fac_{i+1}(x) = (fac_i(x) + 1)!$. The number of elements of $Pats(m, n, \Sigma)$ is bounded by $fac_m(|\Sigma|^{n+1})$, and the size of each $p \in Pats(m, n, \Sigma)$ is bounded by $(n + 1) \cdot \Pi_{i=0}^{m-1} fac_i(|\Sigma|^{n+1})$.*

The bounds given in Lemma 2 are non-elementary in m. This indicates that all of our algorithms are computationally unfeasible from the asymptotic complexity point of view. However, LTL formulae that are used in practice typically have a small nesting depth of U (usually not larger than 3 or 4), and do not contain any X operators. In this light, the bounds of Lemma 2 can actually be interpreted as "good news", because even a relatively small formula φ can be decomposed into a disjunction of many formulae which refine the meaning of φ.

To all $m, n \in \mathbb{N}_0$ and $\alpha \in \Sigma^\omega$ we associate a unique pattern of $Pats(m, n, \Sigma)$. This definition is again inductive.

Definition 3. *Let* $\alpha \in \Sigma^\omega$. *For all* $m, n \in \mathbb{N}_0$ *we define the* characteristic (m,n)-pattern *of* α, *denoted* $pat(m,n,\alpha)$, *and* (m,n)-pattern word *of* α, *denoted* $patword(m,n,\alpha)$, *inductively as follows:*

- $pat(0,n,\alpha) = \alpha(0)\ldots\alpha(n)$
- $patword(m,n,\alpha) \in Pats(m,n,\Sigma)^\omega$ *is defined by* $patword(m,n,\alpha)(i) = pat(m,n,\alpha_i)$
- $pat(m{+}1,n,\alpha)$ *is the finite word (written in parenthesis) obtained from* $patword(m,n,\alpha)$ *by deleting all repeated letters*

Words $\alpha, \beta \in \Sigma^\omega$ *are* (m,n)-equivalent, *written* $\alpha \sim_{m,n} \beta$, *iff* $pat(m,n,\alpha) = pat(m,n,\beta)$.

Example 4. Let us consider a word $\alpha = abbbacbac(ba)^\omega$. Then

$$\begin{aligned}
pat(0,0,\alpha) &= a\\
patword(0,0,\alpha) &= abbbacbac(ba)^\omega = \alpha\\
pat(1,0,\alpha) &= (abc)\\
patword(1,0,\alpha) &= (abc)(bac)(bac)(bac)(acb)(cba)(bac)(acb)(cba)((ba)(ab))^\omega\\
pat(2,0,\alpha) &= ((abc)(bac)(acb)(cba)(ba)(ab))
\end{aligned}$$

$$\begin{aligned}
pat(0,1,\alpha) &= \underline{ab}\\
patword(0,1,\alpha) &= \underline{ab}\,\underline{bb}\,\underline{bb}\,\underline{ba}\,\underline{ac}\,\underline{cb}\,\underline{ba}\,\underline{ac}\,\underline{cb}(\underline{ba}\,\underline{ab})^\omega\\
pat(1,1,\alpha) &= (\underline{ab}\,\underline{bb}\,\underline{ba}\,\underline{ac}\,\underline{cb})
\end{aligned}$$

■

Theorem 5. *Let* Σ *be an alphabet. For all* $m, n \in \mathbb{N}_0$ *and every* $p \in Pats(m,n,\Sigma)$ *there effectively exists a formula* $\varphi_p \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ *such that for every* $\alpha \in \Sigma^\omega$ *we have that* $\alpha \models \varphi_p$ *iff* $pat(m,n,\alpha) = p$.

Example 6. Let $\alpha = abbabaaabb(ac)^\omega$. Then the formula φ_p, where $p = pat(2,0,\alpha) = ((abc)(bac)(ac)(ca))$ is constructed (according to the proof of the previous theorem) as follows:

$$\begin{aligned}
\varphi_{(abc)} &= \mathsf{G}(a \vee b \vee c) \wedge a \wedge (a \,\mathsf{U}\, b) \wedge ((a \vee b) \,\mathsf{U}\, c)\\
\varphi_{(bac)} &= \mathsf{G}(b \vee a \vee c) \wedge b \wedge (b \,\mathsf{U}\, a) \wedge ((b \vee a) \,\mathsf{U}\, c)\\
\varphi_{(ac)} &= \mathsf{G}(a \vee c) \wedge a \wedge (a \,\mathsf{U}\, c)\\
\varphi_{(ca)} &= \mathsf{G}(c \vee a) \wedge c \wedge (c \,\mathsf{U}\, a)\\
\varphi_p &= \mathsf{G}\big(\varphi_{(abc)} \vee \varphi_{(bac)} \vee \varphi_{(ac)} \vee \varphi_{(ca)}\big) \wedge \varphi_{(abc)} \wedge \big(\varphi_{(abc)} \,\mathsf{U}\, \varphi_{(bac)}\big) \wedge\\
&\quad \wedge \big((\varphi_{(abc)} \vee \varphi_{(bac)}) \,\mathsf{U}\, \varphi_{(ac)}\big) \wedge \big((\varphi_{(abc)} \vee \varphi_{(bac)} \vee \varphi_{(ac)}) \,\mathsf{U}\, \varphi_{(ca)}\big)
\end{aligned}$$

■

Let us note that the size of φ_p for a given $p \in Pats(m,n,\Sigma)$ is exponential in the size of p. However, if φ_p is represented by a circuit (DAG), then the size of the circuit is only linear in the size of p.

Theorem 7. *Let* Σ *be an alphabet and let* $m, n \in \mathbb{N}_0$. *For all* $\alpha, \beta \in \Sigma^\omega$ *we have that* α *and* β *cannot be distinguished by any* $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ *formula if and only if* $\alpha \sim_{m,n} \beta$.

In other words, Theorem 7 says that the information about α which is relevant with respect to (in)validity of all $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae is exactly represented by $pat(m,n,\alpha)$. Thus, characteristic patterns provide a new characterization of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ languages which can be used to prove further results about LTL. In particular, a simplified form of (m,n)-stutter invariance of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ languages (see [3]) follows easily from the presented results on characteristic patterns:

Theorem 8. *Let $m, n \in \mathbb{N}_0$, $u, v \in \Sigma^*$ and $\alpha \in \Sigma^\omega$. If v is (m,n)-redundant in $uv\alpha$, then $uv\alpha \sim_{m,n} u\alpha$.*

Theorem 8 provides the crucial tool which was used in [3] to prove that, e.g., the $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$, $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ hierarchies are strict, that the class of ω-languages which are definable both in $\mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^n)$ and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^{n+1})$ is strictly larger than the class of languages definable in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, and so on. The proof of Theorem 8 is shorter than the one given in [3].

3 Applications in model checking

In this section, we expand the remarks about formula decomposition and pattern refinement that were sketched in the introduction. We also discuss potential benefits and drawbacks of these techniques, and provide examples illustrating the presented ideas.

Definition 9. *Let $p \in Pats(m, n, \Sigma)$ be a pattern and $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ be a formula. We say that p* satisfies *φ, written $p \models \varphi$, iff for every ω-word $\alpha \in \Sigma^\omega$ we have that if $pat(m, n, \alpha) = p$, then $\alpha \models \varphi$.*

Note that Theorem 7 implies the following: if $p \not\models \varphi$, then for every ω-word α such that $pat(m, n, \alpha) = p$ we have $\alpha \not\models \varphi$.

Theorem 10. *Given an (m,n)-pattern p and an $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formula φ, the problem whether $p \models \varphi$ can be decided in time $\mathcal{O}(|\varphi| \cdot |p|)$.*

In the rest of this section we consider the variant of LTL where formulae are built over *atomic propositions (At)* rather than over letters. The only change in the syntax is that a ranges over At. The logic is interpreted over ω-words over an alphabet $\Sigma \subseteq 2^{At}$, where $\alpha \models a$ iff $a \in \alpha(0)$. The formula $\mathsf{F}\varphi$ is to be understood just as an abbreviation for $\mathtt{tt}\,\mathsf{U}\,\varphi$, and $\mathsf{G}\varphi$ as an abbreviation for $\neg\mathsf{F}\neg\varphi$.

Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ be the negation of a property we want to verify for a given system. If our model-checker fails to verify whether the system has a run satisfying φ or not (one typical reason is memory overflow), we can proceed by decomposing the formula φ in the following way. First, we compute the set $P = \{p \in Pats(m, n, 2^{At(\varphi)}) \mid p \models \varphi\}$. Then, each $p \in P$ is translated into an equivalent LTL formula.

Example 11. We illustrate the decomposition technique on a formula $\varphi = \mathsf{FG}\neg a$ which is the negation of a typical liveness property $\mathsf{GF}a$. The alphabet is $\Sigma = 2^{\{a\}} = \{\{a\}, \emptyset\}$. To simplify our notation, we use A and B to abbreviate $\{a\}$ and $\emptyset$, respectively. The elements of $Pats(2, 0, (\{A, B\})$ are listed below (unsatisfiable patterns have been eliminated). All patterns which satisfy φ are listed in the second line.

$$((A)), ((BA)(A)), ((AB)(BA)), ((BA)(AB)), ((AB)(BA)(A)), ((BA)(AB)(A))$$
$$((B)), ((AB)(B)), ((BA)(AB)(B)), ((AB)(BA)(B))$$

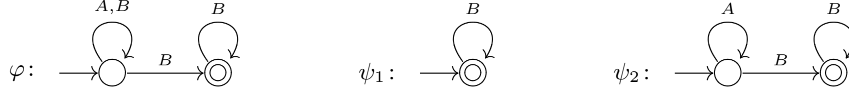

Fig. 1. Büchi automata corresponding to formulae φ, ψ_1, and ψ_2 of Example 11.

So, the formula φ is decomposed into a disjunction $\psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4$ of formulae corresponding to the patterns listed in the second line, respectively[4]:

$$\begin{array}{ll} \psi_1 = \mathsf{G}\neg a & \psi_3 = \neg a \wedge \mathsf{F}(a \wedge \mathsf{F}\neg a) \wedge \mathsf{FG}\neg a \\ \psi_2 = a \wedge a \,\mathsf{U}\, \mathsf{G}\neg a & \psi_4 = a \wedge \mathsf{F}(\neg a \wedge \mathsf{F}a) \wedge \mathsf{FG}\neg a \end{array}$$ ■

Thus, the original question whether the system has a run satisfying φ is decomposed into k questions of the same type. These can be solved using standard model-checkers.

We illustrate potential benefits of this method in the context of automata-theoretic approach to model checking [7]. Here the formula φ is translated into a corresponding Büchi automaton A_φ. Then, the model-checking algorithm computes another Büchi automaton called *product automaton*, which accepts exactly those runs of the verified system which are accepted by A_φ as well. The model-checking problem is thus reduced to the problem whether the language accepted by the product automaton is empty or not. The bottleneck of this approach is the size of the product automaton.

Example 12. Let us suppose that a given model-checking algorithm does not manage to check the formula φ of Example 11. The subtasks given by the ψ_i formulae constructed in Example 11 can be more tractable. Some of the reasons are given below.

- The size of the Büchi automaton for ψ_i can be smaller than the size of A_φ. In Example 11, this is illustrated by formula ψ_1 (see Fig. 1). The corresponding product automaton is then smaller as well.
- The size of the product automaton for ψ_i can be smaller than the one for φ, even if the size of A_{ψ_i} is larger than the size of A_φ. This can be illustrated by the formula ψ_2 of Example 11; the automata for φ and ψ_2 are almost the same (see Fig. 1), but the product automaton for ψ_2 can be much smaller as indicated in Fig. 2. ■

It is of course possible that some of the ψ_i formulae in the constructed decomposition remain intractable. Such a formula ψ_i can further be decomposed by a technique called *refinement* (since ψ_i corresponds to a unique pattern $p_i \in \mathit{Pats}(m,n,\Sigma)$, we also talk about pattern refinement). We propose two basic ways how to refine the pattern p_i. The first possibility is to compute the set of (m',n')-patterns, where $m' \geq m$ and $n' \geq n$, and identify all patterns satisfying the formula ψ_i.

Example 13. The formula ψ_3 of Example 11 corresponding to the $(2,0)$-pattern $((BA)(AB)(B))$ can be refined into two LTL($\mathsf{U}^3,\mathsf{X}^0$) formulae given by the $(3,0)$-patterns $(((BA)(AB)(B))((AB)(BA)(B))((AB)(B))((B)))$ and $(((BA)(AB)(B))((AB)(B))((B)))$. ■

[4] For notation convenience, we simplified the formulae obtained by running the algorithm of Theorem 5 into a more readable (but equivalent) form.

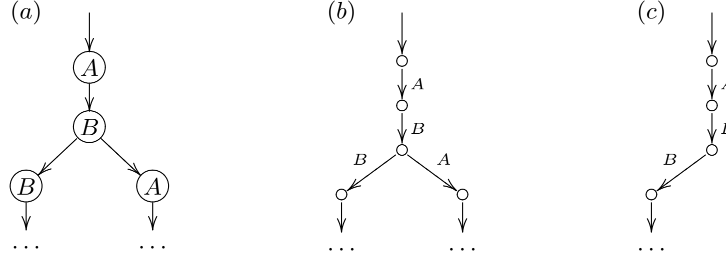

Fig. 2. An example of a verified system (a) and product automata (b) and (c) corresponding to φ and ψ_2 of Example 11, respectively.

The other refinement method is based on enlarging the alphabet before computing the patterns. We simply add a new atomic proposition to the set of atomic propositions that occur in φ. The choice of the new atomic proposition is of course important. By a "suitable" choice we mean a choice which leads to a convenient split of system's runs into more manageable units. An interesting problem (which is beyond the scope of this paper) is whether suitable new propositions can be identified effectively.

Example 14. Let us consider the formula ψ_2 of Example 11 corresponding to the $(2,0)$-pattern $((AB)(B))$. The original set of atomic propositions $At(\varphi) = \{a\}$ generates the alphabet $\Sigma = \{A, B\}$, where $A = \{a\}, B = \emptyset$. If we enrich the set of atomic propositions with b, we get a new alphabet $\Sigma' = \{C, D, E, F\}$, where $C = \{a, b\}, D = \{a\}, E = \{b\}, F = \emptyset$. Hence, the original letters A, B correspond to the pairs of letters C, D and E, F, respectively. Thus, the formula ψ_2 is refined into LTL$(\mathsf{U}^2, \mathsf{X}^0)$ formulae given by 64 $(2,0)$-patterns $((CE)(E)), ((CDE)(DE)(E)), ((CDE)(DCE)(CE)(E)), \ldots$. ■

Some of the subtasks obtained by refining intractable subtasks can be tractable. Others can be refined again and again. Observe that even if we solve only some of the subtasks, we still obtain a new piece of relevant knowledge about the system—we know that if the system has a "bad" run satisfying φ, then the run satisfies one of the formulae corresponding to the subtasks we did not manage to solve. Hence, we can (at least) classify and repeatedly refine the set of "suspicious" runs.

We finish this section by listing the benefits and drawbacks of the presented method.

+ The subtasks are formulated as standard model-checking problems. Therefore, the method can be combined with all existing algorithms and heuristics.
+ With the help of the method, we can potentially verify some systems which are beyond the reach of existing model-checkers.
+ Even if it is not possible complete the verification task, we get partial information about the structure of potential (undiscovered) bad runs. We also know which runs of the system have been successfully verified.
+ The subtasks can be solved simultaneously in a distributed environment with a very low communication overhead.

+ When we verify more formulae on the same system, the subtasks occurring in decompositions of both formulae are solved just once.
– Calculating the decomposition of a given formula can be expensive. On the other hand, this is not critical for formulae with small number of atomic propositions and small nesting depths of U and X.
– Runtime costs of the proposed algorithm are high. It can happen that all subtasks remain intractable even after several refinement rounds and we get no new information at all.

4 Conclusions and future work

The aim of this paper was to introduce the idea of characteristic patterns, develop basic results about these patterns, and indicate how they can be used in LTL model-checking. An obvious question is how the presented algorithms work in practice. This can only be answered by performing a set of experiments. We plan to implement the presented algorithms and report about their functionality in our future work.

Acknowledgement. We thank Michal Kunc for providing crucial hints which eventually led to the definition of characteristic patterns.

References

[1] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 16, pages 995–1072. Elsevier, 1990.
[2] A. Kučera and J. Strejček. Characteristic patterns for LTL. Technical Report FIMU-RS-2004-10, Faculty of Informatics, Masaryk University Brno, 2004.
[3] A. Kučera and J. Strejček. The stuttering principle revisited: On the expressiveness of nested X and U operators in the logic LTL. In *11th Annual Conference of the European Association for Computer Science Logic (CSL'02)*, volume 2471 of *LNCS*, pages 276–291. Springer, 2002.
[4] N. Markey and Ph. Schnoebelen. Model checking a path (preliminary report). In *Proc. 14th Int. Conf. Concurrency Theory (CONCUR'03)*, volume 2761 of *LNCS*, pages 251–265. Springer, 2003.
[5] K. L. McMillan. Verification of infinite state systems by compositional model checking. In *Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *LNCS*, pages 219–237. Springer, 1999.
[6] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.
[7] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society Press, 1986.
[8] Th. Wilke. Classifying discrete temporal properties. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of *LNCS*, pages 32–46. Springer, 1999.
[9] W. Zhang. Combining static analysis and case-based search space partitioning for reducing peak memory in model checking. *Journal of Computer Science and Technology*, 18(6):762–770, 2003.