FI MU

Faculty of Informatics
Masaryk University Brno

Characteristic Patterns for LTL

by

Antonín Kučera
Jan Strejček

FI MU Report Series FIMU-RS-2004-10

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

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

Here a ranges over a countable set $\Lambda = \{a, b, c, \ldots\}$ of *letters*. The set of 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 $\alpha(i)\alpha(i+1)\ldots$.

[*]This work has been supported by GAČR, grant No. 201/03/1161.
[†]This is a full version of SOFSEM'05 paper.

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

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

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

Observe that $\mathsf{F}\varphi$ is equivalent to $\mathtt{tt}\,\mathsf{U}\,\varphi$. Therefore, $\mathsf{F}\varphi$ is sometimes considered just as an abbreviation for $\mathtt{tt}\,\mathsf{U}\,\varphi$. We included the F modality into the LTL syntax explicitly because we consider various LTL fragments where the F and U modalities are distinguished. For every LTL formula φ and every modal operator $M \in \{\mathsf{X}, \mathsf{U}, \mathsf{F}\}$ we define the *nesting depth* of M in φ, denoted M-*depth*(φ), inductively as follows (Y ranges over unary operators $\{\neg, \mathsf{F}, \mathsf{X}\}$ and Z ranges over binary operators $\{\wedge, \mathsf{U}\}$).

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

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

A lot of research effort has been invested into characterizing the expressive power of LTL and its fragments. Kamp [5] proved that the logic LTL is expressively equivalent to first-order logic interpreted over ω-words. Using the results presented in [12] and [1], Perrin [9] showed that an ω-language L is definable in first-order logic iff L is ω-regular and noncounting. See [3, 13] for a more comprehensive overview. Later, various fragments of LTL have been characterized by identifying relevant structures called "forbidden patterns" in the corresponding minimal deterministic finite-state automata [15]. The semantic relationship between LTL fragments has also been studied. In [4], it is shown that the LTL($\mathsf{U}^m, \mathsf{X}, \mathsf{F}$) hierarchy is semantically strict, i.e., the class of languages definable by LTL($\mathsf{U}^{m+1}, \mathsf{X}, \mathsf{F}$) formulae is strictly larger than the class of languages definable by LTL($\mathsf{U}^m, \mathsf{X}, \mathsf{F}$) formulae (for every $m \in \mathbb{N}_0$). This proof can be adapted to show the strictness of LTL(U^m, X) hierarchy. A simpler proof for the LTL(U^m, X) hierarchy was given later in [6]. A related result [11] shows the decidability of the problem whether a given ω-regular language L is definable in LTL($\mathsf{U}^m, \mathsf{X}, \mathsf{F}$) for a given $m \in \mathbb{N}_0$.

In this paper, we give a new characterization of ω-languages that are definable in LTL($\mathsf{U}^m, \mathsf{X}^n$) for given $m, n \in \mathbb{N}_0$. Roughly speaking, for each alphabet Σ and all $m, n \in \mathbb{N}_0$ we design a finite set of (m, n)-*patterns*[1], 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 LTL($\mathsf{U}^m, \mathsf{X}^n$).
- For each (m, n)-pattern p we can effectively construct a formula $\psi \in$ 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 $\varphi \in$ LTL($\mathsf{U}^m, \mathsf{X}^n$) is fully characterized by a finite subset of (m, n)-patterns, and vice versa. Intuitively, the (m, n)-patterns represent *exactly* the information about ω-words which determines the (in)validity of 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 LTL($\mathsf{U}^m, \mathsf{X}^n$) languages introduced in [6]. This, in turn, allows to construct simpler proofs for some of the results presented in [6] (like, e.g., the strictness of the LTL(U^m, X), LTL(U, X^n), and LTL($\mathsf{U}^m, \mathsf{X}^n$) hierarchies). An interesting question (which is left open) is whether one could use char-

[1]Let us note that (m, n)-patterns have nothing to do with the forbidden patterns of [15].

acteristic patterns to demonstrate the decidability of the problem if a given ω-regular language L is definable in LTL(U^m, 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 which specifies desired properties of the system. The question is whether all runs of the system satisfy the formula. This problem can be dually 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 $\varphi \equiv \psi_1 \vee \cdots \vee \psi_n$ 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 how to refine the patterns is to enlarge 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 $\psi_1 \vee \cdots \vee \psi_n$ and try to model-check the $\psi_1, \cdots, \psi_n$ 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[2]. 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 [8] and [16]. In [8], a specification formula of the form $\mathsf{G}\varphi$ is decomposed into a set of formulae $\{\mathsf{G}(x{=}v_i \Rightarrow \varphi) \mid v_i$ 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 [8] for more details). In [16], 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 $\psi_1 \vee \ldots \vee \psi_n$. 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. This makes sense in situations when we want to check a large number of formulae on the same system. The patterns fully characterize the system's behaviour (with respect to properties expressible in a given LTL$(\mathsf{U}^m, \mathsf{X}^n)$ fragment), and this information could be re-used when checking the individual formulae. 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 (this problem is known as *model checking a path* [7]).

This paper is organized as follows. Section 2 provides a formal definition of (m, n)-patterns together with basic theorems. Section 3 is devoted to applications of characteristic patterns in model checking area. More precisely, the section contains detailed discussion of the indicated decomposition technique and the pattern-based algorithm for model checking a path. Conclusions and directions for future research are given in Section 4.

[2]Practical implementation of the method is under preparation.

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 deletion of 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 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 $\alpha \in \{a, b, c\}^\omega$ if and only if *pat*$(1, 0, \alpha)$ equals to

$$(b), (ba), (bc), (bac), (bca), (ab), \text{ or } (abc).$$

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 LTL$(\mathsf{U}^1, \mathsf{X}^0)$ formula. So, a language defined by a formula $\varphi \in$ LTL$(\mathsf{U}^1, \mathsf{X}^0)$ over alphabet $\Sigma = \{a, b, c\}$ 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 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 get the sequence

$$(abc)(abc)(bac)(ac)(ca)(a)^{\omega}.$$

The pattern $pat(2,0,\alpha)$ is obtained from the above sequence by deletion of 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 LTL($\mathsf{U}^2,\mathsf{X}^0$) formula. Moreover, for each $p \in Pats(2,0,\{a,b,c\})$ there is an 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, to be 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 towards letters using the following equivalences (see, for example, [3]):

$$\begin{array}{ll} \mathsf{X}\mathtt{tt} \Leftrightarrow \mathtt{tt} & \mathsf{X}(\neg\varphi) \Leftrightarrow \neg\mathsf{X}\varphi \\ \mathsf{X}(\varphi_1 \,\mathsf{U}\, \varphi_2) \Leftrightarrow \mathsf{X}\varphi_1 \,\mathsf{U}\, \mathsf{X}\varphi_2 & \mathsf{X}(\varphi_1 \wedge \varphi_2) \Leftrightarrow \mathsf{X}\varphi_1 \wedge \mathsf{X}\varphi_2 \end{array}$$

Note that the nesting depth of X remains unchanged by performing this transformation. 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.

Now we present a full formal definition of characteristic patterns.

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:*

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

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

Lemma 2. *For every* $\mathrm{i} \in \mathbb{N}_0$*, let us define the function* $\mathit{fac}_\mathrm{i} : \mathbb{N}_0 \to \mathbb{N}_0$ *inductively as follows:*

$$\mathit{fac}_\mathrm{i}(x) = \begin{cases} x & \textit{if } \mathrm{i} = 0 \\ (\mathit{fac}_{\mathrm{i}-\mathrm{i}}(x) + 1)! & \textit{otherwise} \end{cases}$$

The number of elements of $\mathit{Pats}(m, n, \Sigma)$ *is bounded by* $\mathit{fac}_m(|\Sigma|^{n+1})$*, and the size of each* $p \in \mathit{Pats}(m, n, \Sigma)$ *is bounded by* $(n + 1) \cdot \Pi_{i=0}^{m-1} \mathit{fac}_i(|\Sigma|^{n+1})$*.*

Proof: Directly from definitions. ■

The bounds given in Lemma 2 are non-elementary in m. This indicates that all of our algorithms are computationally unfeasible from the asymptotic analysis 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 be actually 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 $\mathit{Pats}(m, n, \Sigma)$. 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* $\mathit{pat}(m, n, \alpha)$*, and* (m, n)-pattern word *of* α*, denoted* $\mathit{patword}(m, n, \alpha)$*, inductively as follows:*

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

Words $\alpha, \beta \in \Sigma^\omega$ *are said to be* (m, n)-pattern equivalent*, written* $\alpha \sim_{m,n} \beta$*, if* $\mathit{pat}(m, n, \alpha) = \mathit{pat}(m, n, \beta)$.

Example 4. *Let us consider an ω-word $\alpha = \mathtt{abbbacbac(ba)}^\omega$. Some examples of characteristic patterns follow. Underlined sequences correspond to $(0,n)$-patterns, where $n > 0$.*

$$
\begin{aligned}
\mathit{pat}(0,0,\alpha) &= \mathtt{a} \\
\mathit{patword}(0,0,\alpha) &= \mathtt{abbbacbac(ba)}^\omega \;=\; \alpha \\
\mathit{pat}(1,0,\alpha) &= \mathtt{(abc)} \\
\mathit{patword}(1,0,\alpha) &= \mathtt{(abc)(bac)(bac)(bac)(acb)(cba)(bac)(acb)(cba)((ba)(ab))}^\omega \\
\mathit{pat}(2,0,\alpha) &= \mathtt{((abc)(bac)(acb)(cba)(ba)(ab))} \\
\mathit{pat}(0,1,\alpha) &= \underline{\mathtt{ab}} \\
\mathit{patword}(0,1,\alpha) &= \underline{\mathtt{ab}}\,\underline{\mathtt{bb}}\,\underline{\mathtt{bb}}\,\underline{\mathtt{ba}}\,\underline{\mathtt{ac}}\,\underline{\mathtt{cb}}\,\underline{\mathtt{ba}}\,\underline{\mathtt{ac}}\,\underline{\mathtt{cb}}(\underline{\mathtt{ba}}\,\underline{\mathtt{ab}})^\omega \\
\mathit{pat}(1,1,\alpha) &= (\underline{\mathtt{ab}}\,\underline{\mathtt{bb}}\,\underline{\mathtt{ba}}\,\underline{\mathtt{ac}}\,\underline{\mathtt{cb}}) \\
\mathit{pat}(0,2,\alpha) &= \underline{\mathtt{abb}}
\end{aligned}
$$

Theorem 5. *Let Σ be an alphabet. For all $m, n \in \mathbb{N}_0$ and every $p \in \mathit{Pats}(m,n,\Sigma)$ 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 $\mathit{pat}(m,n,\alpha) = p$.*

Proof: We proceed by induction on m.

- **$\mathbf{m = 0}$.** If $p \in \mathit{Pats}(0,n,\Sigma)$, then p is of the form $a_0 \dots a_n$, where each $a_i \in \Sigma$. Hence, we put

  $$\varphi_p = a_0 \wedge \mathsf{X}\,(a_1 \wedge \mathsf{X}\,(a_2 \wedge \,\dots\, \wedge \mathsf{X}\,(a_{n-1} \wedge \mathsf{X} a_n)\dots)).$$

  Obviously, $\varphi_p \in \mathrm{LTL}(\mathsf{U}^0, \mathsf{X}^n)$. Moreover, each $\alpha \in \Sigma^\omega$ such that $\alpha \models \varphi$ starts with $a_0 \dots a_n$, which means that $\mathit{pat}(0,n,\alpha) = a_0 \dots a_n$. The other direction is also trivial.

- **Induction step.** Let $p \in \mathit{Pats}(m{+}1, n, \Sigma)$. This means that p is of the form $p = (p_1 \dots p_k)$, where each $p_i \in \mathit{Pats}(m,n,\Sigma)$ and $p_i \neq p_j$ for $i \neq j$. By induction hypothesis, for each $1 \le i \le k$ there effectively exists a formula $\varphi_{p_i} \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ which satisfies the properties of our lemma. We put

  $$\varphi_p = \mathsf{G}(\varphi_{p_1} \vee \dots \vee \varphi_{p_k}) \;\wedge\; \varphi_{p_1} \;\wedge \bigwedge_{1<j\le k} (\varphi_{p_1} \vee \dots \vee \varphi_{p_{j-1}})\,\mathsf{U}\,\varphi_{p_j}.$$

  Obviously, $\varphi_p \in \mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^n)$. Now let $\alpha \in \Sigma^\omega$. By induction hypothesis, for all $i \in \mathbb{N}_0$ and $1 \le j \le k$ we have that $\alpha_i \models \varphi_{p_j}$ iff $\mathit{pat}(m,n,\alpha_i) = p_j$. First we prove that if $\alpha \models \varphi_p$, then $\mathit{pat}(m{+}1,n,\alpha) = p$. So, let $\alpha \models \varphi_p$, and let us consider the word $\mathit{patword}(m,n,\alpha)$. With the help of induction hypothesis, we can conclude that φ_p actually says that

- all patterns of $\mathit{Pats}(m, n, \Sigma)$ which appear in $\mathit{patword}(m, n, \alpha)$ are among $p_1, \ldots, p_n$ (this is expressed by the subformula $\mathsf{G}(\varphi_{p_1} \vee \ldots \vee \varphi_{p_k})$),
- each p_i appears in $\mathit{patword}(m, n, \alpha)$, and for all $1 \leq i < j \leq k$, the first occurrence of p_i precedes the first occurrence of p_j (this is expressed by the subformula $\varphi_{p_1} \wedge \bigwedge_{1<j\leq k}(\varphi_{p_1} \vee \ldots \vee \varphi_{p_{j-1}}) \mathsf{U}\, \varphi_{p_j})$

This means that $\mathit{pat}(m+1, n, \alpha) = p$ as required. The other implication (i.e. $\mathit{pat}(m+1, n, \alpha) = p$ implies $\alpha \models \varphi_p$) follows similarly. ■

Example 6. *Let $\alpha = abbabaaabb(ac)^\omega$. Then the formula φ_p, where $p = \mathit{pat}(2, 0, \alpha) = ((abc)(bac)(ac)(ca))$, is constructed as follows:*

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

Let us note that the size of φ_p for a given $p \in \mathit{Pats}(m, n, \Sigma)$ 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* LTL$(\mathsf{U}^m, \mathsf{X}^n)$ *formula if and only if $\alpha \sim_{m,n} \beta$.*

Proof: The "$\Longrightarrow$" direction follows directly from Theorem 5. We prove the other direction. Let $\varphi \in$ LTL$(\mathsf{U}^m, \mathsf{X}^n)$ be a formula. As mentioned above, we can safely assume that the X operator occurs only in subformulae of the form $\mathsf{XX}\ldots\mathsf{X}a$, where a is a letter. By induction on the structure of φ we show that for every α, β such that $\alpha \sim_{m,n} \beta$ we have that $\alpha \models \varphi \iff \beta \models \varphi$.

- $\varphi = a$. It follows directly from Definition 3 that $\alpha(0)\ldots\alpha(n) = \beta(0)\ldots\beta(n)$. This means that $\alpha \models a \iff \beta \models a$ as required.

- **Induction step.** If $\varphi = \mathsf{X}\psi$, then $\varphi = \mathsf{XX}\ldots\mathsf{X}a$, where the nesting depth of X in φ is at most n. Hence, we can argue in the same way as in the basic step. The cases when $\varphi = \neg\psi$ or $\varphi = \psi \wedge \rho$ follow directly from induction hypothesis.

Now let $\varphi = \psi \mathsf{U} \rho$. We show that if $\alpha \models \varphi$, then also $\beta \models \varphi$. So, let $\alpha \models \varphi$. This means there is $j \in \mathbb{N}_0$ such that $\alpha_j \models \rho$, and for every $i < j$ it holds that $\alpha_i \models \psi$. As $\mathit{pat}(m, n, \alpha) = \mathit{pat}(m, n, \beta)$, the sequence of first occurrences of letters in $\mathit{patword}(m-1, n, \alpha)$ is the same as in $\mathit{patword}(m-1, n, \beta)$. Let j' be the index of the first occurrence of a letter $\mathit{pat}(m-1, n, \alpha_j)$ in $\mathit{patword}(m-1, n, \beta)$, i.e. $\mathit{pat}(m-1, n, \beta_{j'}) = \mathit{pat}(m-1, n, \alpha_j)$. As $\rho \in \mathrm{LTL}(\mathsf{U}^{m-1}, \mathsf{X}^n)$, by induction hypothesis we obtain that $\beta_{j'} \models \rho$. In the same way we can show that $\beta_{i'} \models \psi$ for every $i' < j'$, because for each such i' we have that $\mathit{pat}(m-1, n, \beta_{i'}) = \mathit{pat}(m-1, n, \alpha_i)$ for some $i < j$. This means $\beta \models \psi \mathsf{U} \rho$. ■

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 $\mathit{pat}(m, n, \alpha)$. 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 [6]) follows easily from the presented results on characteristic patterns:

Lemma 8. *For all $m, n \in \mathbb{N}_0$, $v, w \in \Sigma^*$, $\alpha \in \Sigma^\omega$ it holds that if w is a prefix of v^ω and $|w| \geq |v| \cdot m - m + n + 1$ then $vw\alpha \sim_{m,n} w\alpha$.*

Proof: Let n, v, w, α satisfy the conditions of our lemma. We prove that $\mathit{pat}(m, n, vw\alpha) = \mathit{pat}(m, n, w\alpha)$. By induction on m.

- **m = 0**. It suffices to realize that $(vw)(i) = w(i)$ for $0 \leq i \leq n$. Hence, $\mathit{pat}(0, n, vw\alpha) = \mathit{pat}(0, n, w\alpha)$.

- **Induction step (m > 0).** First we prove that for every $0 \leq i < |v|$ it holds that

$$\mathit{pat}(m-1, n, v_i w\alpha) = \mathit{pat}(m-1, n, w_i\alpha). \tag{1}$$

The concatenation $v_i w$ can be seen as a concatenation $v' w_i$, where $|v'| = |v|$. Further, w_i is a prefix of $(v')^\omega$ and $|w_i| \geq |w_{|v|-1}| = |w| - |v| + 1$. Hence,

$$\begin{array}{rcl} |w_i| & \geq & |w| - |v| + 1 \\ & \geq & |v| \cdot m - m + n + 1 - |v| + 1 \\ & \geq & |v| \cdot (m-1) + |v| - m + n + 1 - |v| + 1 \\ & \geq & |v| \cdot (m-1) - (m-1) + n + 1 \end{array}$$

As $|v'| = |v|$, we obtain (1) by applying induction hypothesis.

We have proven that the first $|v|$ letters of ω-words *patword*$(m-1,n,vw\alpha)$ and *patword*$(m-1,n,w\alpha)$ are the same. Further, these letters are followed by $|v|$ repeated letters in *patword*$(m-1,n,vw\alpha)$. As $(vw\alpha)_{2|v|} = (w\alpha)_{|v|}$, the suffixes *patword*$(m-1,n,vw\alpha)_{2|v|}$ and *patword*$(m-1,n,w\alpha)_{|v|}$ are the same. Hence, *pat*$(m,n,vw\alpha) =$ *pat*$(m,n,w\alpha)$. ■

The conditions of Lemma 8 match the definition of (m,n)-redundancy of the subword v in an ω-word $vw\alpha$ as given in [6].

Theorem 9. *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* $uv\alpha \sim_{m,n} u\alpha$.

Proof: The theorem follows from Lemma 8 and the implication $\beta \sim_{m,n} \gamma \implies u\beta \sim_{m,n} u\gamma$ that can be easily proven for all $m,n \in \mathbb{N}_0$, $\beta,\gamma \in \Sigma^\omega$, and $u \in \Sigma^*$ by induction on m. ■

Theorem 9 provides the crucial tool which was used in [6] to prove that, e.g., the LTL$(\mathsf{U}^m,\mathsf{X})$, LTL$(\mathsf{U},\mathsf{X}^n)$, and LTL$(\mathsf{U}^m,\mathsf{X}^n)$ hierarchies are strict, that the class of ω-languages which are definable both in LTL$(\mathsf{U}^{m+1},\mathsf{X}^n)$ and LTL$(\mathsf{U}^m,\mathsf{X}^{n+1})$ is strictly larger than the class of languages definable in LTL$(\mathsf{U}^m,\mathsf{X}^n)$, and so on. The proof of Theorem 9 is shorter than the one given in [6].

3 Applications in model checking

In this section, the applicability of results about characteristic patterns to LTL model checking is discussed in greater detail. First of all, we introduce an algorithm deciding whether a pattern satisfies an LTL formula or not.

Definition 10. *Let* $p \in$ *Pats*(m,n,Σ) *be a pattern and* $\varphi \in$ LTL$(\mathsf{U}^m,\mathsf{X}^n)$ *be a formula. We say that* p satisfies φ, *written* $p \models \varphi$, *if 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 11. *Given an* (m,n)*-pattern* p *and an* LTL$(\mathsf{U}^m,\mathsf{X}^n)$ *formula* φ, *the problem whether* $p \models \varphi$ *can be decided in time* $\mathcal{O}(|\varphi|\cdot|p|)$.

```
proc $\mathsf{check}(\varphi, p, n)$
  if $\mathsf{U}$-depth$(()\varphi) < \mathsf{mtype}(p)$ then $\mathsf{return}(\mathsf{check}(\varphi, p(0), n))$
  elsif $\varphi = \mathsf{tt}$ then $\mathsf{return}(\mathsf{true})$
  elsif $\varphi \in \Sigma$ then $\mathsf{return}(\varphi == p(n))$
  elsif $\varphi = \neg\varphi_1$ then $\mathsf{return}(\neg\mathsf{check}(\varphi_1, p, n))$
  elsif $\varphi = \varphi_1 \wedge \varphi_2$ then $\mathsf{return}(\mathsf{check}(\varphi_1, p, n) \wedge \mathsf{check}(\varphi_2, p, n))$
  elsif $\varphi = \mathsf{X}\varphi_1$ then $\mathsf{return}(\mathsf{check}(\varphi_1, p, n + 1))$
  elsif $\varphi = \varphi_1 \,\mathsf{U}\, \varphi_2$
        then do
              $i := 0$
              while $(i < |p|) \wedge \neg\mathsf{check}(\varphi_2, p(i), n)$ do
                    if $\mathsf{check}(\varphi_1, p(i), n)$ then $i := i + 1$
                                                        else $i := |p|$
                    fi
              od
              $\mathsf{return}(i < |p|)$
           od
  fi
```

Figure 1: An algorithm deciding whether $p \models \varphi$ or not.

Proof: Consider the algorithm of Figure 1. The procedure call *check*$(\varphi, p, 0)$ decides whether $p \models \varphi$ or not. The function *mtype*(p) returns the m such that $p \in \mathit{Pats}(m, n, \Sigma)$. The algorithm is designed for all φ and p satisfying U-*depth*$(()\varphi) \leq \mathit{mtype}(p)$.

The algorithm cannot assume that the X operators in φ have been pushed inside because this transformation can lead to a formula of the size $\mathcal{O}(|\varphi|^2)$. Thus, the algorithm pushes the X operators towards letters 'virtually': the actual nesting depth of X operators is kept in the third argument of the *check* procedure and it affects the evaluation of the subformulae of the form a (see the line 4). The correctness of the algorithm follows directly from the semantics of LTL and the idea of characteristic patterns.

The complexity of our algorithm is $\mathcal{O}(|\varphi| \cdot |p|)$ as the procedure *check* is invoked at most once for every subformula and every subpattern. Let us note that values of U-*depth*$(()\varphi')$ and *mtype*(p') for all subformulae φ' of φ and all subpatterns p' of p can be pre-calculated with the complexity $\mathcal{O}(|\varphi| + |p|)$. ■

In the rest of this section we discuss potential applications of characteristic patterns mentioned in the introduction.

3.1 Decomposition technique

In this subsection 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 a formula. 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:

1. First we compute the set $P = \{p \in \mathit{Pats}(m, n, 2^{At(\varphi)}) \mid p \models \varphi\}$.

2. Then, each $p \in P$ is translated into an equivalent LTL formula (using, for example, the algorithm of Theorem 5).

A simple way how to implement the first step is to compute the set $\mathit{Pats}(m, n, 2^{At(\varphi)})$, and then for each element p decide whether $p \models \varphi$ using the algorithm of Theorem 11. In practice, this could be optimized by using a more sophisticated algorithm which takes into account the structure of φ and possibly also eliminates unsatisfiable patterns. In the second step, the patterns could be alternatively translated directly into the formalism adopted in the chosen model checker (e.g. Büchi automata or alternating automata).

Example 12. *We illustrate the decomposition technique on a formula* $\varphi = \mathsf{FG}\neg q$ *which is the negation of a typical liveness property* $\mathsf{GF}q$*. The alphabet is* $\Sigma = 2^{\{q\}} = \{\{q\}, \emptyset\}$*. To simplify our notation, we use* A *and* B *to abbreviate* $\{q\}$ *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 right column.*

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

The formulae corresponding to the patterns of the right column are listed below.[3]

$((B))$	$\psi_1 = \mathsf{G}\neg q$
$((AB)(B))$	$\psi_2 = q \wedge q\,\mathsf{U}\,\mathsf{G}\neg q$
$((BA)(AB)(B))$	$\psi_3 = \neg q \wedge \mathsf{F}(q \wedge \mathsf{F}\neg q) \wedge \mathsf{FG}\neg q$
$((AB)(BA)(B))$	$\psi_4 = q \wedge \mathsf{F}(\neg q \wedge \mathsf{F}q) \wedge \mathsf{FG}\neg q$

So, the formula φ *is decomposed into an equivalent disjunction* $\psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4$*.*

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-based approach to model checking [14]. Here the formula φ is translated into a Büchi automaton A_φ accepting the ω-language $L(\varphi)$. 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 13. *Let us suppose that a given model checking algorithm does not manage to check the formula* φ *of Example 12. The subtasks given by the* ψ_i *formulae constructed in Example 12 can be more tractable. Some of the reasons are illustrated below.*

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

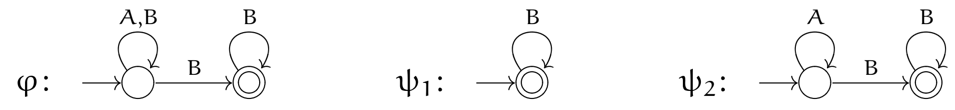

Figure 2: Büchi automata corresponding to formulae φ, ψ_1, and ψ_2 of Example 12.

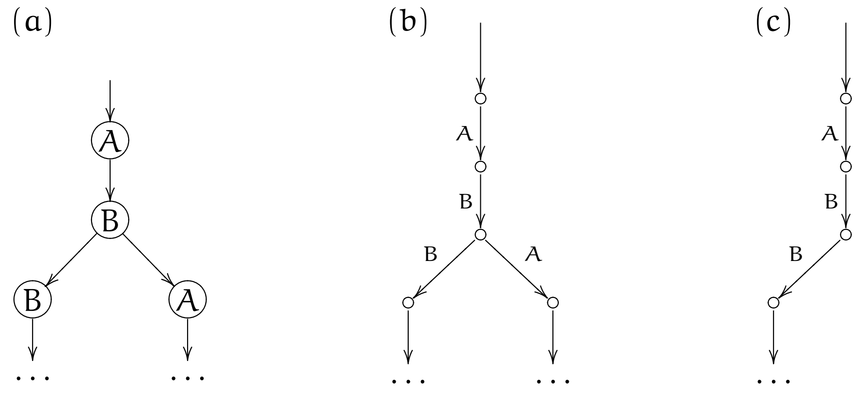

Figure 3: An example of a system to be verified (a) and product automata (b) and (c) corresponding to φ and ψ_2 of Example 12, respectively.

- *The size of the Büchi automaton for ψ_i can be smaller than the size of A_φ. In Example 12, this is illustrated by formula ψ_1 (see Figure 2). The corresponding product automaton is then smaller as well.*

- *The size of the product automaton constructed 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. In Example 12, this is illustrated by the formula ψ_2; the automata for φ and ψ_2 are almost the same (see Figure 2), but the product automaton for ψ_2 can be much smaller as indicated in Figure 3.*

It is of course possible that some of the ψ_i formulae in the constructed decomposition remain intractable. Let ψ_i be such an intractable formula. Then ψ_i can be further decomposed by a technique called *refinement* (since ψ_i corresponds to a unique pattern $p_i \in Pats(m, n, 2^{At(\varphi)})$, we can equivalently consider pattern refinement). There are two basic ways how to refine the pattern p_i. The idea of the first method is to compute the set of (m', n')-patterns, where $m' \geq m$ and $n' \geq n$, and identify all patterns that satisfy the formula ψ_i.

Example 14. *The formula* ψ_3 *of Example 12 corresponding to the* $(2,0)$*-pattern* $((\mathsf{BA})(\mathsf{AB})(\mathsf{B}))$ *can be refined into two* $\mathrm{LTL}(\mathsf{U}^3,\mathsf{X}^0)$ *formulae given by the* $(3,0)$*-patterns*

$$\begin{aligned}&(((\mathsf{BA})(\mathsf{AB})(\mathsf{B}))((\mathsf{AB})(\mathsf{B}))((\mathsf{B}))),\\&(((\mathsf{BA})(\mathsf{AB})(\mathsf{B}))((\mathsf{AB})(\mathsf{BA})(\mathsf{B}))((\mathsf{AB})(\mathsf{B}))((\mathsf{B}))).\end{aligned}$$

The other refinement method is based on enlarging the alphabet before computing the patterns. We simply expand the set $At(\varphi)$ with a new atomic proposition. 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 a potential topic for future work) is whether suitable new propositions can be identified effectively.

Example 15. *Let us consider the formula* ψ_2 *of Example 12 corresponding to the* $(2,0)$*-pattern* $((\mathsf{AB})(\mathsf{B}))$*. The original set of atomic propositions* $At(\varphi)=\{\mathsf{q}\}$ *generates the alphabet* $\Sigma=\{\mathsf{A},\mathsf{B}\}$*, where* $\mathsf{A}=\{\mathsf{q}\},\mathsf{B}=\emptyset$*. If we enrich the set of atomic propositions with* r*, we get a new alphabet* $\Sigma'=\{\mathsf{C},\mathsf{D},\mathsf{E},\mathsf{F}\}$*, where* $\mathsf{C}=\{\mathsf{q},\mathsf{r}\},\mathsf{D}=\{\mathsf{q}\},\mathsf{E}=\{\mathsf{r}\},\mathsf{F}=\emptyset$*. 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* $\mathrm{LTL}(\mathsf{U}^2,\mathsf{X}^0)$ *formulae given by the* $(2,0)$*-patterns*

$$\begin{aligned}&((\mathsf{CE})(\mathsf{E}))\\&((\mathsf{CDE})(\mathsf{DE})(\mathsf{E}))\\&((\mathsf{CDE})(\mathsf{DCE})(\mathsf{CE})(\mathsf{E}))\\&((\mathsf{CDE})(\mathsf{DCE})(\mathsf{DE})(\mathsf{E}))\\&((\mathsf{CEF})(\mathsf{EF})(\mathsf{FE}))\\&((\mathsf{CEF})(\mathsf{EF})(\mathsf{FE})(\mathsf{E}))\\&((\mathsf{CEF})(\mathsf{EF})(\mathsf{FE})(\mathsf{F}))\\&((\mathsf{CDEF})(\mathsf{DEF})(\mathsf{EF})(\mathsf{FE}))\\&((\mathsf{CDEF})(\mathsf{DEF})(\mathsf{EF})(\mathsf{FE})(\mathsf{E}))\\&((\mathsf{CDEF})(\mathsf{DEF})(\mathsf{EF})(\mathsf{FE})(\mathsf{F}))\\&((\mathsf{CDEF})(\mathsf{DCEF})(\mathsf{CEF})(\mathsf{EF})(\mathsf{FE}))\\&((\mathsf{CDEF})(\mathsf{DCEF})(\mathsf{CEF})(\mathsf{EF})(\mathsf{FE})(\mathsf{E}))\\&((\mathsf{CDEF})(\mathsf{DCEF})(\mathsf{CEF})(\mathsf{EF})(\mathsf{FE})(\mathsf{F}))\\&((\mathsf{CDEF})(\mathsf{DCEF})(\mathsf{DEF})(\mathsf{EF})(\mathsf{FE}))\\&((\mathsf{CDEF})(\mathsf{DCEF})(\mathsf{DEF})(\mathsf{EF})(\mathsf{FE})(\mathsf{E}))\\&((\mathsf{CDEF})(\mathsf{DCEF})(\mathsf{DEF})(\mathsf{EF})(\mathsf{FE})(\mathsf{F}))\end{aligned}$$

and all those patterns which can be obtained from the above given ones by either exchanging the letters C, D*, or exchanging the letters* E, F*, or by both exchanges. Hence, the formula* ψ_2 *is refined into a disjunction of* $16 \cdot 4 = 64$ *formulae.*

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 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 subsection 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) runs satisfying φ. 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.

3.2 Model checking a path using patterns

Markey and Schnoebelen [7] have recently introduced the problem of model checking a single infinite path. More precisely, they consider infinite paths of the form uv^ω, where $u, v \in \Sigma^*$ and $v \neq \varepsilon$, called *loops*. LTL model checking of a loop can be done in bilinear (i.e. $\mathcal{O}(|uv| \cdot |\varphi|)$) time using the algorithm for CTL model checking of finite-state systems [2] (the expressive power of CTL and LTL on linear structures coincides). We present a new algorithm based on characteristic patterns and argue that our algorithm can be more efficient in *some* cases.

Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ be a formula and uv^ω be a loop, where $u, v \in \Sigma^*$ and $v \neq \varepsilon$. The algorithm first computes a pattern *pat*(m, n, uv^ω) and then it decides whether the pattern satisfies φ. First we focus on the pattern extraction.

$$\begin{tikzcd}
1 \arrow[r] & 2 \arrow[r] & \cdots \arrow[r] & k \arrow[r] & k{+}1 \arrow[r] & k{+}2 \arrow[r] & \cdots \arrow[r] & k{+}l \arrow[llll, bend right]
\end{tikzcd}$$

Figure 4: A finite-state system with one infinite path.

Let $k = |u|$ and $l = |v|$. The loop can be represented by the finite structure given in Figure 4 and a function L labelling each state of the structure with the corresponding letter of the loop, i.e.

$$L(i) = \begin{cases} u(i-1) & \text{if } 1 \leq i \leq k, \\ v(i-k-1) & \text{if } k+1 \leq i \leq k+l. \end{cases}$$

By $s(i)$ we denote a *successor* of a state i defined by arrow leading from i.

The pattern extraction algorithm given in Figure 5 computes a new labelling function L'. The desired pattern *pat*(m, n, uv^ω) is stored in $L'(1)$ after the algorithm terminates. The algorithm is based directly on the definition of characteristic patterns. After the i-th iteration of the second for-loop, the labels stored in $L'(1) \ldots L'(k{+}l)$ describe the ω-word *patword*(i, n, uv^ω). More precisely, *patword*$(i, n, uv^\omega) = L'(1) \ldots L'(k)\,(L'(k{+}1) \ldots L'(k{+}l)\,)^\omega$. The time complexity of this algorithm is $\mathcal{O}(|uv| \cdot (n + m \cdot S(m, n, \Sigma)))$, where $S(m, n, \Sigma)$ is the maximal size of a pattern in *Pats*(m, n, Σ) (as given in Lemma 2).

Due to Theorem 11, the problem whether *pat*$(m, n, uv^\omega) \models \varphi$ can be solved in $\mathcal{O}(S(m, n, \Sigma) \cdot |\varphi|)$ time. Hence, the algorithm needs $\mathcal{O}(|uv|(n + m \cdot S(m, n, \Sigma)) + S(m, n, \Sigma) \cdot |\varphi|)$ time in total. In the light of this estimation, our algorithm seems to

```
for $i := 1$ to $k + l$ do
    $L'(i) := L(i)L(s(i))L(s^2(i)) \dots L(s^n(i))$
od
for $i := 1$ to $m$ do
    $L'(k{+}l)$ := the parenthesized word obtained from
                  $L'(k{+}l)L'(k{+}1)L'(k{+}2) \dots L'(k{+}l{-}1)$
                  by deletion of all repeated letters
    for $j := k + l - 1$ downto $1$ do
        $L'(j)$ := the word $L'(j{+}1)$ with the letter $L'(j)$ added to the beginning
                   and without any repetition of this letter
    od
od
```

Figure 5: An algorithm for (m, n)-pattern extraction.

be only worse than the bilinear CTL-like algorithm. However, if we bound the parameters m, n, and $|\Sigma|$ by constants (this is justifiable as these are usually "small") then our algorithm needs only $\mathcal{O}(|uv| + |\varphi|)$ time, while the CTL-like algorithm still requires $\mathcal{O}(|uv| \cdot |\varphi|)$ time. In other words, our algorithm is better in situations when m, n and $|\Sigma|$ are small, and $|uv|$ is large. Then it pays to extract the characteristic pattern from the loop and check the formulae directly on the pattern rather than on the loop itself.

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. Furthermore, we study other potential applications of characteristic patterns in model checking area, namely in state space reduction.

Acknowledgement

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

References

[1] A. Arnold. A syntactical congruence for rational ω-languages. *Theoretical Computer Science*, 39:333–335, 1985.

[2] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[3] 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.

[4] Kousha Etessami and Thomas Wilke. An until hierarchy for temporal logic. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 108–117, New Brunswick, New Jersey, 1996. IEEE Computer Society Press.

[5] Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[6] 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.

[7] 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.

[8] 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.

[9] Dominique Perrin. Recent results on automata and infinite words. In M. P. Chytil and V. Koubek, editors, *Proceedings of the 11th Symposium on Mathematical Foundations of Computer Science*, volume 176 of *LNCS*, pages 134–148. Springer, 1984.

[10] 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.

[11] Denis Thérien and Thomas Wilke. Temporal logic and semidirect products: An effective characterization of the until hierarchy. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 256–263. IEEE Computer Society Press, 1996.

[12] Wolfgang Thomas. Star-free regular sets of ω-sequences. *Information and Control*, 42(2):148–156, 1979.

[13] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, Formal models and semantics, pages 133–191. Elsevier, 1990.

[14] 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.

[15] 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.

[16] 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.