Subsequence versus substring constraints in sequence pattern languages

Peer-reviewed author version

ENGELS, Steven; TAN, Tony & VAN DEN BUSSCHE, Jan (2021) Subsequence versus substring constraints in sequence pattern languages. In: ACTA INFORMATICA, 58, p. 35-56.

DOI: 10.1007/s00236-019-00347-5
Handle: http://hdl.handle.net/1942/30404

**Acta Informatica manuscript No.**
(will be inserted by the editor)

# Subsequence versus substring constraints in sequence pattern languages

**Steven Engels · Tony Tan · Jan Van den Bussche**

Received: date / Accepted: date

**Abstract** A family of logics for expressing patterns in sequences is investigated. The logics are all fragments of first-order logic, but they are variable-free. Instead, they can use substring and subsequence constraints as basic propositions. Propositions expressing constraints on the beginning or the end of the sequence are also available. Also wildcards can be used, which is important when the alphabet is not fixed, as is typical in database applications. The maximal logic with all four features of substring, subsequence, begin–end constraints, and wildcards, turns out to be equivalent to the family of star-free regular languages of dot-depth at most one. We investigate the lattice formed by taking all possible combinations of the above four features, and show it to be strict. For instance, we formally confirm what might intuitively be expected, namely, that boolean combinations of substring constraints are not sufficient to express subsequence constraints, and vice versa. We show an expressiveness hierarchy results from allowing multiple wildcards. We also investigate what happens with regular expressions when concatenation is replaced by subsequencing. Finally, we study the expressiveness of our logic relative to first-order logic.

**Keywords** pattern language · subsequence · substring · automata · infinite alphabet

The first author acknowledges the generous financial support from Taiwan Ministry of Science and Technology under grant no. 107-2221-E-002-026-MY2.

S. Engels
Hasselt University
E-mail: steven.engels@uhasselt.be

T. Tan
National Taiwan University
E-mail: tonytan@csie.ntu.edu.tw

J. Van den Bussche
Hasselt University
E-mail: jan.vandenbussche@uhasselt.be

## 1 Introduction

A lot of data that we want to manage, query, mine, using modern database systems, is sequential in nature. Sample application areas are text [11], sequence mining [5], bioinformatics [24,6], time series [7], and of course, XML. For querying as well as mining, we want declarative languages, logics, in which we can formally express conditions on sequences. In query applications, these sequences would come straight from a database, and the conditions are meant as query conditions on the sequences that we want to retrieve; in mining, these sequences can be mined from a database, and the conditions are meant as constraints on the desired mining output (mining under constraints).

A classical family of conditions on sequences is the family of regular languages.[1] This family can be expressed using three classical and equivalent logics: finite automata (insofar this counts as a logic); regular expressions; and monadic second-order logic (Büchi's sequential calculus [1,23]). We can limit the sequential calculus to first-order, so that we get the relational calculus familiar from database theory but restricted to work on sequential structures only. When we do that, we can express exactly the star-free regular languages, which are defined by the star-free regular expressions: regular expressions extended with complementation, but limited by throwing out Kleene star.

Regular expressions or calculus formulas are natural and powerful formalisms, but not very user-friendly [9]. Indeed, users of information systems much prefer logics akin to the search expressions familiar from boolean information retrieval [3], in which one specifies keywords and combines those using and, or, and not. Note that keywords are nothing but constraints that certain patterns, namely substrings, must occur in the sequence. But apart from substring, also sub*sequence* patterns are very important in many applications (consult the references given in the first paragraph). A substring is a consecutive part of a sequence, whereas the letters of a subsequence do not need to occur consecutively. So, every substring is a subsequence, but not vice versa. For example, *aba* is a subsequence of *cacbcca*, but not a substring. In this paper, we will use comma **,** to indicate that a pattern should occur as a subsequence rather than as a substring. In contrast, we will use dots to indicate a substring pattern. Thus in the previous example we would say that the sequence *cacbcca* satisfies the pattern $a \,\mathbf{,}\, b \,\mathbf{,}\, a$ but not the pattern $a \bullet b \bullet a$.

The goal of this paper is to fill a small but worthwhile gap in the literature, namely to clarify the differences in expressive power of substring versus subsequence constraints in sequence pattern languages, and also to investigate their combination. Our points of departure are the two basic logics $\mathcal{P}^{\{\bullet\}}$ and $\mathcal{P}^{\{\mathbf{,}\}}$ consisting of boolean combinations of substring patterns and subsequence patterns, respectively. They have incomparable expressiveness, as we will show. The logic $\mathcal{P}^{\{\bullet\}}$ is "locally testable" [12], and indeed, if we additionally allow the begin-marker ^ and the end-marker $ in patterns, the extended

[1] If we identify a condition with the set of sequences (words) that satisfy it, a condition is indeed a formal language.

logic $\mathcal{P}^{\{\bullet,\hat{}\$\}}$ equals, by definition, the family of locally testable languages. On the other hand, the logic $\mathcal{P}^{\{\mathbf{,}\}}$ equals, by definition, the family of piecewise testable languages [15,19].

Of course we can also add the feature $\hat{}, \$$ to $\mathcal{P}^{\{\mathbf{,}\}}$, which will result in a logic $\mathcal{P}^{\{\mathbf{,},\hat{}\$\}}$ incomparable with $\mathcal{P}^{\{\bullet,\hat{}\$\}}$. In the end we can combine all three features in one logic $\mathcal{P}^{\{\bullet,\mathbf{,},\hat{}\$\}}$, where we can have mixed substring–subsequence patterns like $a \bullet b \mathbf{,} b \bullet c \mathbf{,} c \mathbf{,} a$.[2] We will show that $\mathcal{P}^{\{\bullet,\mathbf{,},\hat{}\$\}}$ equals the family of star-free regular languages of dot-depth at most one [4,2,22,15].

In our work, we also take into account the case when the alphabet, over which the sequences are defined, is not fixed. In the theory of formal languages, the alphabet is usually assumed to be fixed, but in database applications that is not always a reasonable assumption. Think of the one-letter wildcard ?. If the alphabet is fixed, such a wildcard can always be eliminated using disjunction, e.g., if the alphabet is $\{a, b, c\}$, then $c?a$ can be rewritten as $caa \vee cba \vee cca$. But such a rewriting is infeasible if the alphabet equals the set of data elements from some database instance, which may be large and constantly changing. In this spirit, we include the wildcard ? as an additional feature that can be added to all the logics we consider. Notably, all of our *positive* results, to the effect that one logic is at least as powerful than another, will be shown to hold uniformly over unknown alphabets; all of our *negative* results, to the effect that a certain condition cannot be expressed in some logic, will be shown to hold already for the fixed two-letter alphabet $\{a, b\}$.[3] The only exception is, of course, where we show that the logics with ? are more powerful than those without. In fact, we extend this result to the case of arbitrary number of wildcards, namely, we show that more wildcards means more expressiveness. This result is achieved via pebble automata over infinite alphabet, a well known model of computation over infinite alphabets [13,20,21].[4]

We also investigate what happens with the classical formalism of regular expressions when we replace the standard concatenation operator $\bullet$ by the comma operator $\mathbf{,}$ which allows arbitrary letters to be inserted in between two concatenated words. The semantics of Kleene star is similarly adapted. The resulting logic of, as we call them, *comma-regular expressions*, denoted by $\mathrm{disjRE}^{\mathbf{,}}$, turns out to be rather weak. Specifically, $\mathrm{disjRE}^{\mathbf{,}}$ is equivalent to the positive, even the positive–disjunctive, fragment of $\mathcal{P}^{\{\mathbf{,},\hat{}\$\}}$. In particular, adding intersection to $\mathrm{disjRE}^{\mathbf{,}}$ does not give anything more. But we will show that $\mathrm{RE}^{\mathbf{,}}$, obtained by adding complementation to $\mathrm{disjRE}^{\mathbf{,}}$, immediately gives all (and only) the star-free regular languages. Finally, we study the expressiveness of our logic relative to the standard first-order logic over finite strings.

---

[2] Note that mixing $\mathbf{,}$ and $\bullet$ is the same as allowing "globbing" wildcards $*$ in substring patterns, familiar from the Unix shell where the previous pattern would be written as $ab * bc * c * a$.

[3] Over the one-letter alphabet, all logics we consider collapse to the family of finite or cofinite languages.

[4] We note that, motivated in part by database applications, formal language theory has now been revisited to accommodate an unknown (or infinite) alphabet [18].

*Organisation.* In Section 2 we define our pattern logic based on the operations subsequence and substring and show that more operators means more expressiveness. In Section 3 we extend our pattern logic with the wildcard operators. Then in Section 4 we study what we call *comma-regular expressions*, namely, the regular expressions in which we replace the concatenation operator with the comma operator. In Section 5 we compare the expressiveness of our pattern logic with first-order logic. We conclude in Section 6.

## 2 Pattern logics

For convenience, we assume a countable infinite set $\mathbb{U}$, the elements of which are called letters/symbols. An alphabet is defined as a finite subset of $\mathbb{U}$. The set of finite sequences over an alphabet $\Sigma$ is denoted by $\Sigma^*$.

### 2.1 Patterns

Let $\Sigma$ be an alphabet. We define the set of *basic patterns* over $\Sigma$ recursively as follows:

1. $\emptyset$ is a basic pattern.
2. Every letter $a \in \Sigma$ is a basic pattern.
3. For any subset $Z \subseteq \Sigma$, the expression $\texttt{?} \setminus Z$ is a basic pattern.
4. If $\alpha$ and $\beta$ are basic patterns, so are $\alpha \bullet \beta$ and $\alpha \,\textbf{,}\, \beta$.

A *pattern* is either a basic pattern, or is of one of the three forms $\hat{}\beta$, $\beta\,\$$, or $\hat{}\beta\,\$$ with $\beta$ a basic pattern. We call $\hat{}$ and $\$$ the begin and end marker, respectively, and we assume that they do not belong to $\mathbb{U}$.

The *language generated by a basic pattern $\beta$ over $\Sigma$*, denoted by $L_\Sigma(\beta)$, is inductively defined as follows:

1. $L_\Sigma(\emptyset) := \emptyset$.
2. $L_\Sigma(a) := \{a\}$.
3. $L_\Sigma(\texttt{?} \setminus Z) := \Sigma \setminus Z$.
4. (a) $L_\Sigma(\alpha \bullet \beta) := L_\Sigma(\alpha) \bullet L_\Sigma(\beta)$, with $\bullet$ the classical concatenation operator on sequences, extended to sets of sequences in the standard way.
   (b) $L_\Sigma(\alpha \,\textbf{,}\, \beta) := L_\Sigma(\alpha) \,\textbf{,}\, L_\Sigma(\beta)$, with $\textbf{,}$ the operator on sets of sequences defined as follows:
$$S \,\textbf{,}\, T = S \bullet \Sigma^* \bullet T.$$

The *set of sequences that match pattern $\alpha$ over $\Sigma$*, denoted by $M_\Sigma(\alpha)$, is defined as follows. For any basic pattern $\beta$, we let:

$$\begin{aligned} M_\Sigma(\beta) &:= \Sigma^* \bullet L_\Sigma(\beta) \bullet \Sigma^* \\ M_\Sigma(\hat{}\beta) &:= L_\Sigma(\beta) \bullet \Sigma^* \\ M_\Sigma(\beta\,\$) &:= \Sigma^* \bullet L_\Sigma(\beta) \\ M_\Sigma(\hat{}\beta\,\$) &:= L_\Sigma(\beta) \end{aligned}$$

When the alphabet $\Sigma$ is clear from the context, we will omit $\Sigma$ and simply write basic patterns, patterns, $L(\alpha)$ and $M(\beta)$.

A pattern is also called a $\{\bullet, ,, ?, \hat{}\,\$\}$-pattern. For any subset $f$ of the set of features $\{\bullet, ,, ?, \hat{}\,\$\}$, we can consider the patterns that only use features from $f$; such patterns are called *$f$-patterns.* If such a pattern is basic, it is also called a basic $f$-pattern. To be interesting, $f$ should contain at least the dot or the comma, thus:

**Definition 1** A *feature set* is a subset of $\{\bullet, ,, ?, \hat{}\,\$\}$ containing at least the dot or the comma.

It is rather apparent that ^ and \$ are orthogonal in their semantics. So, we group them as one feature to avoid the pedantic case of a set of features that includes one, but not the other.

*Example 1* Take $\Sigma = \{a, b, c\}$.

- $a \bullet b \,, (? \setminus \{b\}) \bullet c \,, c \,, a$ is a basic pattern over $\Sigma$;
- $a \bullet b \bullet c$ is a basic $\{\bullet\}$-pattern over $\Sigma$; and
- $a \,, b \,, c\,\$$ is a $\{,, \hat{}\,\$\}$-pattern over $\Sigma$.

We will use the following terminologies and notations. We write ? to abbreviate $? \setminus \emptyset$. The *length* of a pattern $\alpha$, denoted by $|\alpha|$, is defined inductively as follows.

- $|\emptyset| = 0$.
- $|? \setminus Z| = |a| = 1$, for any set $Z$ and any letter $a$.
- $|\beta \bullet \gamma| = |\beta \,, \gamma| = |\beta| + |\gamma|$.
- $|\hat{}\beta\,\$\,| = |\hat{}\beta| = |\beta\,\$\,| = |\beta|$, for any basic pattern $\beta$.

That is, the length of a pattern is the number of letters occurring in it. Note that the length of a sequence that matches a pattern $\alpha$ must be at least $|\alpha|$.

For a $\{\bullet, ,\}$-pattern $\beta$, the *basic sequence* of $\beta$ is the sequence obtained from $\beta$ with the dot and comma signs omitted. For example, if $\beta$ is $a \bullet b \,, c \,, a$, then its basic sequence is *abca*. Obviously, the basic sequence of $\beta$ matches $\beta$.

For an integer $n \geq 1$, for a letter $a$, we write $a^n$ to denote the pattern $a \bullet \ldots \bullet a$, where $a$ appears $n$ number of times.

## 2.2 Logics

A *formula* is simply a boolean expression built from patterns over *some* alphabet $\Sigma$. If a formula $\varphi$ is built from patterns over $\Sigma$, then we say that $\varphi$ *is over* $\Sigma$.

A formula is also called a $\{\bullet, ,, ?, \hat{}\,\$\}$-formula. For any feature set $f$, we can consider the formulas that only use $f$-patterns; such formulas are called *$f$-formulas.* The logic consisting of all $f$-formulas is denoted by $\mathcal{P}^f$.

*Example 2* An example of a formula is $\neg(a \bullet b) \wedge \hat{}b \,, c\,\$$.

The set of sequences over $\Sigma$ that are *matched* by a formula $\varphi$ over $\Sigma$, denoted by $M_\Sigma(\varphi)$, is defined in the obvious manner:

– For a pattern $\alpha$, we have already defined $M_\Sigma(\alpha)$ in the previous subsection;
– $M_\Sigma(\varphi \vee \psi) := M_\Sigma(\varphi) \cup M_\Sigma(\psi)$;
– $M_\Sigma(\varphi \wedge \psi) := M_\Sigma(\varphi) \cap M_\Sigma(\psi)$;
– $M_\Sigma(\neg\varphi) := \Sigma^* \setminus M_\Sigma(\varphi)$.

Conjunction is definable in terms of disjunction and negation, but later we will also consider logics without negation. As before, when the alphabet $\Sigma$ is clear from the context, we will omit $\Sigma$ and simply write formulas and $M(\beta)$.

*Remark 1* Note that a formula is not defined over a fixed alphabet $\Sigma$ and neither is the logic $\mathcal{P}^f$. Obviously, if a formula is over an alphabet $\Sigma$, then it is also a formula over any alphabet $\Sigma' \supseteq \Sigma$. In fact, it is valid to say that a formula is over $\Sigma_0$, where $\Sigma_0$ is the set of symbols that appear in the patterns in the formula.

For this reason, we will say that an alphabet $\Sigma$ is *appropriate* for a formula, if $\Sigma$ contains all the symbols that appear in it. Of course, the set $M_\Sigma(\varphi)$ depends on the alphabet $\Sigma$. Note that the notion of "appropriate alphabet" is similar to the one in mathematical logic, where it is not uncommon to state that a vocabulary is appropriate for a first-order formula.

### 2.3 Regular expressions

From the elementary theory of computation, let us recall, for any alphabet $\Sigma$, the syntax of the regular expressions over $\Sigma$. The primitive expressions are $\emptyset$, $\varepsilon$, $a$ for any $a \in \Sigma$, and $\texttt{?} \setminus Z$ for any $Z \subseteq \Sigma$. The primitive $\texttt{?} \setminus Z$ is nonstandard in regular expressions but is included here in order to be able to work with unknown alphabets. The operators are union $e_1 \cup e_2$, intersection $e_1 \cap e_2$, complementation $e^{\mathrm{c}}$, concatenation $e_1 \bullet e_2$, and Kleene star $e^*$. The *language* over $\Sigma$ generated by an expression $e$ is denoted by $L_\Sigma(e)$ and defined in the well-known way. In particular, as for patterns, we define $L_\Sigma(\texttt{?} \setminus Z)$ as $\Sigma \setminus Z$. Note that $L_\Sigma(\emptyset^{\mathrm{c}})$ equals $\Sigma^*$.

In standard presentations of the regular expressions, intersection and complementation are not included, but they are included here so as to be able to define the family of *star-free* regular expressions simply as those expressions that do not use Kleene star. We denote this family by $\mathrm{RE}_0$. A well-known subfamily of $\mathrm{RE}_0$ is that of the star-free regular expressions of *dot-depth at most one*, denoted by $\mathrm{RE}_0^1$. Informally, these are the expressions that do not use nested applications of the $\bullet$ operator. This, however, is defined under the liberal interpretation of $\bullet$ as an associative operator that can take any number of arguments. For example, $a \bullet (b \bullet c)$ does not really count as a nested application, as we can view it simply as the concatenation $(a \bullet b \bullet c)$ of three arguments.

Formally, the dot-depth hierarchy of expressions within $\mathrm{RE}_0$ is defined inductively as follows (for a recent review see [16]).

- An $\mathrm{RE}_0$ expression is said to be of dot-depth 0 if it does not use the concatenation operator. Thus, the expression is a boolean combination (union, intersection, complementation) of primitive expressions.
- Let $k$ be a natural number. An $\mathrm{RE}_0$ expression is said to be of dot-depth at most $k+1$ if it is a boolean combination of concatenations of expressions of dot-depth at most $k$. Here, by a concatenation of expressions $e_1, \ldots, e_n$, we mean the expression $(e_1 \bullet \cdots \bullet e_n)$.

Naturally, a language over $\Sigma$ is said to be of dot-depth at most $k$ if it can be generated by an expression over $\Sigma$ of dot-depth at most $k$. Often, the notion of dot-depth is directly defined for languages, without going through expressions, but the notions are the same.

Similar to Section 2.2, given an expression $\alpha \in \mathrm{RE}_0$, we can say that *an alphabet $\Sigma$ is appropriate for $\alpha$*, if $\Sigma$ contains all the symbols appearing in $\alpha$.

*Example 3* The expression

$$e = (\emptyset^{\mathrm{c}} \bullet a \bullet b \bullet \emptyset^{\mathrm{c}})^{\mathrm{c}} \cap (b^{\mathrm{c}} \bullet \emptyset^{\mathrm{c}} \bullet c)$$

belongs to $\mathrm{RE}_0^1$, i.e., has dot-depth one. Note that for every $\Sigma \supseteq \{a, b, c\}$, $L_\Sigma(e)$ equals $M_\Sigma(\varphi)$ for the formula

$$\varphi = \neg(a \bullet b) \wedge \hat{}\,\mathtt{?} \setminus \{b\} \mathbin{,} c\,\mathtt{\$}\,.$$

The above example suggests a connection between dot-depth at most one and the logic of the previous section. We will establish this connection formally in the next section.

## 2.4 Relative expressiveness

One can think of a sequence pattern logic $\mathcal{P}$ in general to be any formal system that associates to any alphabet $\Sigma$ a set of formulas over $\Sigma$, and to each formula $\varphi$ over $\Sigma$ a set $\mathrm{Mod}_\Sigma(\varphi)$ of sequences over $\Sigma$ that "satisfy" $\varphi$. In this sense, all logics $\mathcal{P}^f$ introduced above are sequence pattern logics (with $\mathrm{Mod}_\Sigma(\varphi)$ given by $M_\Sigma(\varphi)$), and $\mathrm{RE}_0$ and $\mathrm{RE}_0^1$ are as well (with expressions playing the role of formulas and $\mathrm{Mod}_\Sigma(e)$ given by $L_\Sigma(e)$).

When a formula $\varphi$ belongs to the set of formulas associated to alphabet $\Sigma$, we say that $\Sigma$ is *appropriate* for $\varphi$. For the following definition of uniform expressiveness, it is important to note that a formula may have many appropriate alphabets.

We can now give very general notions of relative expressiveness of sequence pattern logics.

**Definition 2** Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two sequence pattern logics and let $\Sigma$ be an alphabet.

- Let $\varphi_1$ be a $\mathcal{P}_1$-formula and $\varphi_2$ be a $\mathcal{P}_2$-formula, both over $\Sigma$. We say that $\varphi_1$ *is expressible by* $\varphi_2$ *over* $\Sigma$, if $\mathrm{Mod}_\Sigma(\varphi_1) = \mathrm{Mod}_\Sigma(\varphi_2)$.

– We say that $\mathcal{P}_2$ *is more expressive than* $\mathcal{P}_1$ *over* $\Sigma$, if every $\mathcal{P}_1$-formula $\varphi_1$ over $\Sigma$ is expressible over $\Sigma$ by some $\mathcal{P}_2$-formula $\varphi_2$ over $\Sigma$.
– Let $\varphi_1$ be a $\mathcal{P}_1$-formula and $\varphi_2$ be a $\mathcal{P}_2$-formula. We say that $\varphi_1$ *is uniformly expressible by* $\varphi_2$, if there is an alphabet $\Sigma_0$ such that
  1. every alphabet $\Sigma$ containing $\Sigma_0$ that is appropriate for $\varphi_1$ is also appropriate for $\varphi_2$; and
  2. over every such alphabet $\Sigma$, $\varphi_1$ is expressible by $\varphi_2$.
– We say that $\mathcal{P}_2$ *is uniformly more expressive than* $\mathcal{P}_1$, if every $\mathcal{P}_1$-formula $\varphi_1$ is uniformly expressible by some $\mathcal{P}_2$-formula.

We say that $\mathcal{P}_2$ is *strictly* more expressive than $\mathcal{P}_1$ (over a fixed alphabet or uniformly), if $\mathcal{P}_2$ is more expressive than $\mathcal{P}_1$ but not vice versa. If $\mathcal{P}_1$ is more expressive than $\mathcal{P}_2$ and vice versa, we call $\mathcal{P}_1$ and $\mathcal{P}_2$ *equally expressive.*

Note that our definition of uniform expressibility allows some leeway in the form of a minimum alphabet $\Sigma_0$ that may be assumed by $\varphi_2$. For example, the formula $\neg\emptyset$ is uniformly expressible by the regular expression $a \cup a^c$, where $a$ is any letter. So here we would use $\Sigma_0 = \{a\}$. While we think it is only reasonable to allow this leeway, in our results, we will actually not need it.

Note also that a positive result, to the effect that one logic is more expressive than another, is stronger if proved uniformly, whereas a negative result is stronger if proved for a fixed alphabet.

*Example 4* As noted in the Introduction, it is easy to see that for any feature set $f$, the logic $\mathcal{P}^{f\cup\{?\}}$ and $\mathcal{P}^f$ are equally expressive over any fixed alphabet. But this is not true uniformly: as we will see later wildcards do add expressiveness when the alphabet is not fixed.

Our first result establishes an equivalence between the full logic and the star-free expressions of dot-depth at most one.

**Proposition 1** $\mathcal{P}^{\{\bullet,\,\boldsymbol{,},\,?,\,\hat{}\$\}}$ *and* $\mathrm{RE}_0^1$ *are equally expressive uniformly.*

*Proof* Every basic pattern $\beta$ can be very simply translated into an $\mathrm{RE}_0^1$-expression $e_\beta$ such that $L_\Sigma(\beta) = L_\Sigma(e_\beta)$ for every alphabet $\Sigma$ appropriate for $\beta$, as shown in the following table.

| $\beta$ | $e_\beta$ |
|---|---|
| $a$ | $a$ |
| $? \setminus Z$ | $? \setminus Z$ |
| $\alpha \bullet \beta$ | $e_\alpha \bullet e_\beta$ |
| $\alpha\ \boldsymbol{,}\ \beta$ | $e_\alpha \bullet \emptyset^c \bullet e_\beta$ |

We do not obtain nested dots since both $\bullet$ and $\boldsymbol{,}$ are translated in terms of concatenation; concatenation is associative, and moreover in $\mathrm{RE}_0$ we use concatenation as a multi-argument operator. For example, if $\beta$ is $(a \bullet b \,\boldsymbol{,}\, c) \bullet a$, then $e_\beta$ is $(a \bullet b \bullet \emptyset^c \bullet c \bullet a)$.

It is then obvious from the definition of $M_\Sigma(\alpha)$ that every pattern $\alpha$ can be translated into an $\mathrm{RE}_0^1$-expression $g_\alpha$ such that $M_\Sigma(\alpha) = L_\Sigma(g_\alpha)$. Finally, the boolean connectives $\vee$, $\wedge$ and $\neg$ are translated into union, intersection and complementation, respectively.

For the other direction, we begin by defining the notion of an *extended* primitive regular expression. These are either the primitive expressions we already had ($\emptyset$, $\varepsilon$, $a$, or $\texttt{?} \setminus Z$), or are of the form $\emptyset^c$, $\varepsilon^c$, or $\mathrm{len}^{\geq 2}$. Here, $\mathrm{len}^{\geq 2}$ is an abbreviation for $\texttt{?}\,\boldsymbol{.}\,\texttt{?}\,\boldsymbol{.}\,\emptyset^c$, where $\texttt{?}$ itself abbreviates $\texttt{?} \setminus \emptyset$. Recall that $L_\Sigma(\emptyset^c) = \Sigma^*$, so $L_\Sigma(\mathrm{len}^{\geq 2})$ is the set of all strings over $\Sigma$ of length at least two. Note that $L_\Sigma(\varepsilon^c)$ is the set of all nonempty strings over $\Sigma$.

We now claim that *every* $\mathrm{RE}_0$ *expression* $e$ *of dot-depth* 0 *can be equivalently written as a union of extended primitive expressions.* To prove this claim, we may assume that $e$ is in disjunctive normal form, so we may actually focus on the case where $e$ is an intersection of primitive regular expressions and their complements. If $e$ is a single primitive regular expression, the claim is trivial. If $e$ is the complement of a primitive regular expression, we can reason as follows:

- $\emptyset^c$ is itself an extended primitive expression.
- $a^c$ is equivalent to $\varepsilon \cup (\texttt{?} \setminus \{a\}) \cup \mathrm{len}^{\geq 2}$.
- $\varepsilon^c$ is itself an extended primitive expression.
- $(\texttt{?} \setminus Z)^c$ is equivalent to $\varepsilon \cup \bigcup_{a \in Z} a \cup \mathrm{len}^{\geq 2}$.

Since intersection distributes over union, the claim now follows because *the intersection of two extended primitive expressions can again be written as an extended primitive expression.* We verify the latter statement as follows.

- The intersection of $\emptyset$ with any other extended primitive expression is $\emptyset$.
- The intersection of $\varepsilon$ with $\emptyset^c$ is $\varepsilon$; otherwise, for $g$ of the form $a$, $\texttt{?} \setminus Z$, $\varepsilon^c$ or $\mathrm{len}^{\geq 2}$, we have $\varepsilon \cap g = \emptyset$.
- The intersection of $a$ with $\emptyset^c$ or $\varepsilon^c$ is $a$; the intersection of $a$ with $\texttt{?} \setminus Z$ is $a$ if $a \notin Z$ and $\emptyset$ otherwise; $a \cap b = \emptyset$ for letters $a \neq b$; and $a \cap \mathrm{len}^{\geq 2} = \emptyset$.
- The intersection of $\texttt{?} \setminus Z$ with $\emptyset^c$ and $\varepsilon^c$ is $\texttt{?} \setminus Z$; with $\mathrm{len}^{\geq 2}$, the intersection is $\emptyset$; and $(\texttt{?} \setminus Z_1) \cap (\texttt{?} \setminus Z_2) = \texttt{?} \setminus (Z_1 \cup Z_2)$.
- The intersection of $\emptyset^c$ with any regular expression $g$ is again $g$.
- Finally, $\varepsilon^c \cap \mathrm{len}^{\geq 2} = \mathrm{len}^{\geq 2}$.

By the claim, and since concatenation distributes over union, a concatenation of $\mathrm{RE}_0$-expressions of dot-depth zero can then be written as a union of concatenations of extended primitive expressions. We next argue that any such concatenation of extended primitive expressions can be expressed by a pattern. Indeed, if we have just $\varepsilon$ by itself, this can be expressed as `^$`. Otherwise $\varepsilon$ can be ignored. Now in the concatenation we perform the following modifications, in order:

1. Each occurrence of $\emptyset^c$ is replaced by a comma.
2. Each occurrence of $\varepsilon^c$ is replaced by `?,`.
3. Each occurrence of $\mathrm{len}^{\geq 2}$ is replaced by `?.?,`.
4. Repeatedly replace any two consecutive commas, or comma and dot, or dot and comma, by a single comma.

If the resulting expression begins with a comma, this comma is deleted, and the same is done with a trailing comma. If, on the other hand, there was no comma at the beginning, a ^ marker is placed there; if there was no comma at the end, a $ marker is placed there. We thus obtain the desired pattern. We conclude that any $\mathrm{RE}_0^1$-expression can be written as a boolean combination of unions of patterns, i.e., a boolean combination of patterns, i.e., a formula, and we are done. □

*Remark 2* As mentioned in the introduction, over every fixed alphabet $\mathcal{P}^{\{\bullet,\hat{}\$\}}$ and $\mathcal{P}^{\{,\}}$ express exactly the locally testable languages and the piecewise testable languages, respectively. Indeed the standard definitions of locally and piecewise testable languages [15] essentially amount to stating that the languages are expressible by a $\{\bullet, \hat{}\$\}$- and $\{,\}$-formulas, respectively. Note also that $\mathcal{P}^{\{\bullet,,,\hat{}\$\}}$ also captures the notion of locally threshold testable languages [17]. All these show yet another connection between our logic and a known family of star-free regular languages.

On the other hand, note that here is no standard uniform notion of locally testable. Our result shows that the logic $\mathcal{P}^{\{\bullet,\hat{}\$,?\}}$ and $\mathcal{P}^{\{,,?\}}$ are natural candidates for the uniform notion of locally and piecewise testable languages, respectively.

We next explore the lattice of the different logics $\mathcal{P}^f$, leaving wildcards out for the time being. So, the six possibilities for $f$ are $\{\bullet\}$; $\{\bullet, ,\}$; $\{\bullet, \hat{}\$\}$; $\{,\}$; $\{,, \hat{}\$\}$; and $\{\bullet, ,, \hat{}\$\}$. We establish that each feature strictly adds expressiveness, and that incomparable sets of features yield incomparable expressiveness.[5]

**Theorem 1** *Let $f$ and $g$ be feature sets without ?.*

1. *If $f$ is included in $g$, then $\mathcal{P}^g$ is uniformly more expressive than $\mathcal{P}^f$.*
2. *If $f$ is not included in $g$, then $\mathcal{P}^g$ is not more expressive than $\mathcal{P}^f$ already over the fixed two-letter alphabet $\{a, b\}$.*

*Proof* The first statement is clear, because if $f$ is included in $g$, then the logic $\mathcal{P}^f$ is simply syntactically contained in the logic $\mathcal{P}^g$. We establish the second statement. For each feature we exhibit a condition on sequences over $\{a, b\}$ that is expressible in the minimal logic having the feature, but not in the maximal logic not having the feature.

For the feature $\bullet$ we use the formula (actually, pattern) $a \bullet b \bullet a$. So, we show that "the sequence has a substring *aba*" is not expressible by a $\{,, \hat{}\$\}$-formula over $\{a, b\}$. Thereto, consider any such formula $\varphi$. Let $m$ be the maximal length of a pattern from $\varphi$, and consider the two sequences $s_1 = a(ba)^{m+1}$ and $s_2 = a(bba)^{m+1}$. Clearly, $s_1$ has a substring *aba* but $s_2$ does not. Nevertheless, $s_1$ and $s_2$ are indistinguishable by $\varphi$. Indeed, we will verify that $s_1$ and $s_2$ satisfy exactly the same $\{,, \hat{}\$\}$-patterns over $\{a, b\}$ of length $\leq m$.

[5] By the connections given in the previous section, this theorem includes as a special case the long-known fact that the family of locally testable languages is strictly included in the family of star-free regular languages.

Specifically, we claim that, for $i = 1, 2$, the sequence $s_i$ matches a $\{\textbf{,}, \hat{}\,\$\}$-pattern of length $\leq m$ if and only if that pattern satisfies the following properties:

1. if it begins with a begin-marker, it must begin with $\hat{}a$;
2. if it ends with an end-marker, it must end with $a\,\$$;
3. if it has both $\hat{}a$ and $a\,\$$, then it must contain at least one comma.

This claim is readily verified as follows.

Only if. The first two properties hold because $s_i$ starts and ends with $a$. The third property holds because, without at least one comma, a $\{\textbf{,}, \hat{}\,\$\}$-pattern that has both $\hat{}a$ and $a\,\$$ must be $\hat{}a\,\$$ which is not matched by $s_i$.

If. Consider a $\{\textbf{,}, \hat{}\,\$\}$-pattern of length $\leq m$ satisfying the three properties. For clarity, assume first the case when the pattern has both $\hat{}a$ and $a\,\$$. Hence, it must be of the form: $\hat{}a\,\textbf{,}\,c_1\,\textbf{,}\,\dots\,\textbf{,}\,c_n\,\textbf{,}\,a\,\$$, where $0 \leq n \leq m-2$. Since each $c_i$ is either $a$ or $b$, both $s_1$ and $s_2$ match the pattern.
For the other cases, i.e., when either $\hat{}a$, or $a\,\$$, or comma is missing, the reasoning is similar.

For the feature $\textbf{,}$ we use the pattern $a\,\textbf{,}\,b\,\textbf{,}\,a$. So, we show that "the sequence has a subsequence *aba*" is not expressible by a $\{\bullet, \hat{}\,\$\}$-formula over $\{a, b\}$. Thereto, consider any such formula $\varphi$. Let $m$ be the maximal length of a pattern from $\varphi$, and consider the two sequences $s_1 = b^{m+1}ab^{m+1}ab^{m+1}$ and $s_2 = b^{m+1}ab^{m+1}$. Clearly, $s_1$ has a subsequence *aba* but $s_2$ does not. Nevertheless, in the same manner as above, we claim that $s_1$ and $s_2$ are indistinguishable by $\varphi$, by showing for $i = 1, 2$, the sequence $s_i$ matches a $\{\bullet, \hat{}\,\$\}$-pattern over $\{a, b\}$ of length $\leq m$ if and only if that pattern satisfies the following properties:

1. It must contain at most one $a$.
2. It does not contain both $\hat{}$ and $\$$.
3. If it contains $\hat{}$ or $\$$, it must not contain any $a$.

This claim is readily verified as follows

Only if. The first property holds because the pattern has length $\leq m$, and in $s_1$, the two $a$'s are strictly more than $m$ letters away from each other. In $s_2$, there is even only one $a$. The second property holds because the pattern, having length $\leq m$, cannot match the entire $s_i$ which is strictly longer than $m$. The third property holds because the only $a$ in $s_i$ is strictly more than $m$ letters away from both the beginning and the end.

If. Consider a $\{\bullet, \hat{}\,\$\}$-pattern of length $\leq m$ satisfying the three properties. We consider two cases. If the pattern has no $a$, then it must be of the form: $\hat{}b^n$, or $b^n\,\$$, or $b^n$, where $n \leq m$. Obviously, both $s_1$ and $s_2$ match such pattern. If the pattern has an $a$, then $a$ appears only one time and the pattern is of the form: $b^n \bullet a \bullet b^k$, where $n + k \leq m$. Obviously, both $s_1$ and $s_2$ match such pattern.

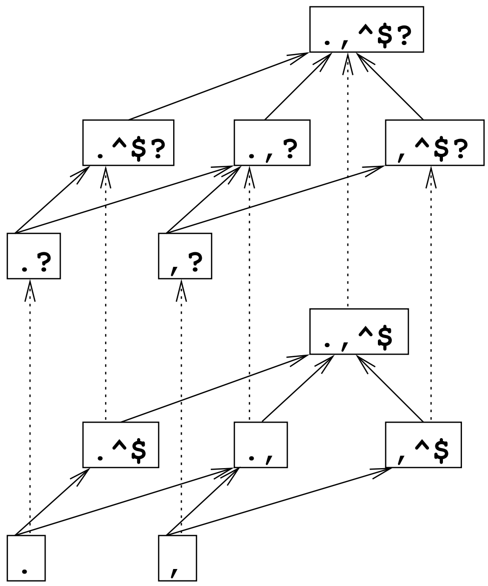


**Fig. 1** Expressiveness lattice of the feature sets. Each arrow gives strictly more expressiveness. Feature sets that are incomparable in the lattice have incomparable expressiveness. The dotted arrows are used to highlight the correspondence between the lattice of feature sets with wildcard and the lattice of feature sets without wildcards, as given by Proposition 2.

For the feature `^$` we use the pattern $\hat{}a$. Take any {`.`, `,`}-formula $\varphi$. Let $\varphi$ a boolean combination of patterns $\beta_1, \ldots, \beta_n$. Let $w_1, \ldots, w_n$ be the basic sequences of $\beta_1, \ldots, \beta_n$, respectively, and let $s_0 = w_1 \cdots w_n$. Then consider the sequences $s_1 = as_0$ and $s_2 = bs_0$. Clearly, $s_1$ begins with $a$ but $s_2$ does not. Nevertheless, $s_1$ and $s_2$ both match all the patterns from $\varphi$, so they are indistinguishable by $\varphi$. □

## 3 Wildcards

In this section, we extend our exploration of the lattice of possible feature sets by including wildcards. The final picture will be as shown in Figure 1.

First, we confirm that the sublattice formed by the feature sets that do contain wildcard is isomorphic to the sublattice formed by the feature sets that do not:

**Proposition 2** *Let $f$ and $g$ be feature sets containing* `?`*. Then the statements (1) and (2) from Theorem 1 also hold for $f$ and $g$.*

*Proof* Let $f' = f \setminus \{?\}$ and $g' = g \setminus \{?\}$. If $f$ is not a subset of $g$, we know from Theorem 1 that $\mathcal{P}^{g'}$ is not more expressive than $\mathcal{P}^{f'}$ over the fixed alphabet $\{a, b\}$. Over any fixed alphabet, however, $\mathcal{P}^f$ and $\mathcal{P}^{f'}$, and $\mathcal{P}^g$ and $\mathcal{P}^{g'}$, are equally expressive. Hence $\mathcal{P}^g$ is not more expressive than $\mathcal{P}^f$. □

It remains to show that adding wildcards adds expressiveness. Of course, this can only be shown in the uniform setting, as stated in the following theorem.

**Theorem 2** *The pattern* ^?$ *cannot be expressed uniformly by any* $\{\bullet, \mathbf{,}, \hat{}\$\}$*-formula.*

*Proof* We first claim that both the sequences $c$ and $cc$ do not match any $\{\bullet, \mathbf{,}, \hat{}\$\}$-pattern $\alpha$, when $\alpha$ does not use the letter $c$. Indeed, if $|\alpha| = 0$, then $c$ and $cc$ do not match $\alpha$. If $|\alpha| \neq 0$, then $\alpha$ contains a letter that is not $c$, which means that to match $\alpha$, a sequence must contains a letter that is not $c$.

Now, assume that ^?$ is uniformly expressed by $\{\bullet, \mathbf{,}, \hat{}\$\}$-formula $\varphi$. Let $\Sigma_0$ be such that for every $\Sigma \supseteq \Sigma_0$, $M_\Sigma(\varphi) = M_\Sigma(\hat{}?\$)$. Let $c$ be a letter that is neither in $\Sigma_0$ nor used in $\varphi$. By our claim above, either both $c$ and $cc$ are in $M_\Sigma(\varphi)$, or both are not in $M_\Sigma(\varphi)$. However, $c \in M_\Sigma(\hat{}?\$)$, but $cc \notin M_\Sigma(\hat{}?\$)$.

In fact, Theorem 2 can be generalized for formulas with arbitrary wildcards $? \setminus Z$ where $Z \neq \emptyset$. We will prove it in the sharpest sense possible:

**Theorem 3** *Let $Z$ be some non-empty finite alphabet. The pattern $?\setminus Z$ cannot be expressed uniformly by any $\{\bullet, \mathbf{,}, \hat{}\$, ?\}$-formula that only uses wildcards of the form $? \setminus Z'$ with $Z$ not a subset of $Z'$.*

*Example 5* This is indeed the sharpest result possible. As soon as $Z \subseteq Z'$, we can express $? \setminus Z$ as $? \setminus Z' \vee \bigvee_{a \in Z' \setminus Z} a$.

*Proof* (of Theorem 3) Assume, for the sake of contradiction, that $? \setminus Z$ is uniformly expressible by some $\{\bullet, \mathbf{,}, \hat{}\$, ?\}$-formula $\varphi$ where the wildcards are of the form $? \setminus Z'$ with $Z \not\subseteq Z'$. By definition, this means that there is $\Sigma_0$ such that for every alphabet $\Sigma \supseteq \Sigma_0$, $M_\Sigma(? \setminus Z) = M_\Sigma(\varphi)$. Without loss of generality, we can assume that $\Sigma_0$ contains all the letters in $\varphi$.

In the following let $\Sigma = \Sigma_0 \cup Z \cup \{c\}$, where $c \notin \Sigma_0 \cup Z$. By definition, $\varphi$ is a boolean combination of patterns $\alpha_1, \ldots, \alpha_k$. Let $\beta_1, \ldots, \beta_k$ be the basic patterns of $\alpha_1, \ldots, \alpha_k$, respectively. Recall that basic patterns are patterns without ^ and $. For each $i = 1, \ldots, k$, let $\hat{\beta}_i$ be the basic pattern obtained by replacing every occurrence of a wildcard $?\setminus Z'$ with a letter $a \in Z \setminus Z'$. Note that each $\hat{\beta}_i$ is a $\{\bullet, \mathbf{,}\}$-pattern. Let $w_i$ be the sequence of letters as they occur in $\hat{\beta}_i$, i.e., $w_i$ is the basic sequence of $\hat{\beta}_i$. Let $A = \{w_i \mid w_i \text{ uses only letters from } Z\}$ and let $s_0$ be the concatenation of the sequences in $A$ (in arbitrary order). If $A = \emptyset$, we set $s_0$ to be $\varepsilon$.

Let $m$ the maximal length of the patterns in $\varphi$ and let $a \in Z$. Then define two sequences $s_1 = a^m c s_0 c a^m$ and $s_2 = a^m s_0 a^m$.

*Claim* For each $i = 1, \ldots, k$, $s_1$ matches $\alpha_i$ if and only if $s_2$ matches $\alpha_i$.

The claim immediately implies that either $s_1, s_2 \in M_\Sigma(\varphi)$ or $s_1, s_2 \notin M_\Sigma(\varphi)$, which contradicts the assumption that $M_\Sigma(\varphi) = M_\Sigma(? \setminus Z)$, since $s_1$ matches $? \setminus Z$, but $s_2$ does not.

We now prove the claim. Suppose $s_1$ matches $\alpha_i$. We first consider the case when $\alpha_i$ neither begins with ^ nor ends with $. Let $\alpha_i$ be of the form: $a_1 \circledast_1 a_2 \circledast_2 \cdots \circledast_{n-1} a_n$ where each $\circledast_j \in \{\centerdot, ,\}$ and each $a_j$ is either a letter from $\Sigma_0$ or a wildcard $\mathtt{?} \setminus Z'$, where $Z' \nsubseteq Z$.

If some $a_j$ is a letter that is not in $Z$, then $s_1$ cannot match $\alpha_i$, since $s_1$ contains only letters from $Z \cup \{c\}$ and $c \notin \Sigma_0$. So, each $a_j$ is either a letter from $Z$ or a wildcard $\mathtt{?} \setminus Z'$. Thus, the basic sequence $w_i$ also matches $\alpha_i$. Since $w_i$ is contained inside $s_0$, and hence, also in $s_2$, it follows that $s_2$ matches $\alpha_i$ too. The proof for the converse direction that $s_2$ matches $\alpha_i$ implies $s_1$ matches $\alpha_1$ is similar.

If $\alpha_i$ begins with ^, then $\alpha_i$ requires that the first $l$ letters matches certain pattern for some $l \leq m$. Since $s_1$ and $s_2$ have the same first $m$ letters, such pattern is matched by $s_1$ if and only if it is matched by $s_2$. The proof is similar when $\alpha_i$ ends with $.

### 3.1 Patterns with Multiple Wildcards

In the semantics defined so far, different wildcard occurrences in a pattern can be independently substituted by letters. In this section we define a pattern logic equipped with multiple wildcards, where each wildcard can occur multiple times in a pattern/formula, but different occurrences of the same wildcard need to be substituted by the same letter. Note that the pattern logic in this new setting captures any property expressible by the pattern logic defined previously: When we want independent substitution as before, we simply use different wildcards and each wildcard can only occur once. In the following we write $\mathtt{?}_1, \mathtt{?}_2, \mathtt{?}_3, \ldots$ to denote the wildcards.

*Example 6* The pattern $\mathtt{?}_1 \centerdot \mathtt{?}_2$ is matched by all strings of length at least two. In contrast, the pattern $\mathtt{?}_1 \centerdot \mathtt{?}_1$ is matched by all strings containing two consecutive occurrences of some same letter.

Formally, basic patterns with multiple wildcards over the alphabet $\Sigma$ are defined recursively as follows.

1. $\emptyset$ is a basic pattern.
2. Every letter $a \in \Sigma$ is a basic pattern.
3. For every finite subset $Z \subsetneq \Sigma$, and every wildcard $\mathtt{?}_i$, the expression $\mathtt{?}_i \setminus Z$ is a basic pattern.
   As before, for succinctness, $\mathtt{?}_i \setminus \emptyset$ is abbreviated as $\mathtt{?}_i$.
4. If $\alpha$ and $\beta$ are basic patterns, then so are $\alpha \centerdot \beta$ and $\alpha , \beta$.

A pattern over the alphabet $\Sigma$ is again of one of the four forms $\beta$, ^$\beta$, $\beta$ $, or ^$\beta$ $ where $\beta$ is a basic pattern over $\Sigma$.

Next we present how a basic pattern $\beta$ with multiple wildcards defines a language $L_\Sigma(\beta)$ over $\Sigma$. Renaming the wildcards, if necessary, we may assume that the wildcards in $\beta$ are $\mathtt{?}_1, \ldots, \mathtt{?}_k$. An *interpretation of wildcards in* $\beta$ is a function $\xi : \{\mathtt{?}_1, \ldots, \mathtt{?}_k\} \to \Sigma$. The basic pattern $\beta$ defines a language $L^\xi_\Sigma(\beta)$ *with respect to a wildcard interpretation* $\xi$ as follows.

1. $L^{\xi}_{\Sigma}(\emptyset) := \emptyset$.
2. For each $a \in \Sigma$, $L^{\xi}_{\Sigma}(a) := \{a\}$.
3. For each $\texttt{?}_i \setminus Z$, $L^{\xi}_{\Sigma}(\texttt{?}_i \setminus Z) := \{\xi(\texttt{?}_i)\} \setminus Z$.
4. If $\beta = \alpha \bullet \gamma$, then $L^{\xi}_{\Sigma}(\beta) := L^{\xi}_{\Sigma}(\alpha) \bullet L^{\xi}_{\Sigma}(\gamma)$.
5. If $\beta = \alpha \,,\, \gamma$, then $L^{\xi}_{\Sigma}(\beta) := L^{\xi}_{\Sigma}(\alpha) \bullet \Sigma^* \bullet L^{\xi}_{\Sigma}(\gamma)$.

The language $L_\Sigma(\beta)$ generated by $\beta$ is now obtained by taking the union over all possible wildcard interpretations:

$$L_\Sigma(\beta) := \bigcup_{\xi:\{\texttt{?}_1,\ldots,\texttt{?}_k\}\to\Sigma} L^{\xi}_{\Sigma}(\beta).$$

Finally, as before, we use the markers ^ and $ to indicate whether a pattern should hold at the start and end of a string, respectively. That is, the patterns $\beta$, $\hat{}\beta$, $\beta\,\$$ and $\hat{}\beta\,\$$ define the following matching languages.

$$\begin{aligned} M_\Sigma(\beta) &:= \Sigma^* \bullet L_\Sigma(\beta) \bullet \Sigma^* \\ M_\Sigma(\hat{}\beta) &:= L_\Sigma(\beta) \bullet \Sigma^* \\ M_\Sigma(\beta\,\$) &:= \Sigma^* \bullet L_\Sigma(\beta) \\ M_\Sigma(\hat{}\beta\,\$) &:= L_\Sigma(\beta) \end{aligned}$$

As before, *formulas* over $\Sigma$ are boolean combinations of patterns (over $\Sigma$), and they define languages over $\Sigma$ where the boolean operators are interpreted in the same way as in Subsection 2.2. As usual, we will omit $\Sigma$ when it is clear from the context.

*Remark 3* Note that we define the semantics by iterating the interpretation $\xi$ "inside" the language $L_\Sigma(\beta)$. So, if we have a formula $\beta_1 \wedge \beta_2$, for some patterns $\beta_1$ and $\beta_2$ using the same set of wildcards, its language is defined as the intersection $M_\Sigma(\beta_1) \cap M_\Sigma(\beta_2)$, where the interpretations $\xi$ are iterated independently in $M_\Sigma(\beta_1)$ and $M_\Sigma(\beta_2)$.

*Example 7* Consider the pattern $\beta := \hat{}\texttt{?}_1 \bullet \texttt{?}_1\,\$$. For an alphabet $\Sigma$, the language $M_\Sigma(\beta)$ is $\{aa \mid a \in \Sigma\}$.

Another example is $\beta := \texttt{?}_1 \,,\, \texttt{?}_1$. Then, $M_\Sigma(\beta)$ consists of all the word $w \in \Sigma^*$ in which some letter occurs at least twice in $w$. On the other hand, $M_\Sigma(\neg\beta)$ consists of all the words in $\Sigma^*$ in which there is no letter that occurs more than once.

Yet, another example is $\beta := \neg(\hat{}\texttt{?}_1\,\$ \vee \hat{}\texttt{?}_1 \bullet \texttt{?}_2\,\$)$. Then, $M_\Sigma(\beta)$ consists of all the words in $\Sigma^*$ of length at least 3.

In the following we will show that more wildcards will yield more expressive power. We use the following notation. For every integer $k \geq 1$, let $f_k$ be the set of features $\{\bullet, \,,\, , \hat{}\$, \texttt{?}_1, \ldots, \texttt{?}_k\}$. We are going to prove the following.

**Theorem 4** *For every integer $k \geq 1$, $\mathcal{P}^{f_{k+1}}$ is strictly more expressive than $\mathcal{P}^{f_k}$ uniformly.*

Note that $\mathcal{P}^{f_1}$ is already strictly more expressive than the logic without wildcard, as implied by Theorem 3. Before presenting the formal proof of Theorem 4, we first present a brief sketch. We will need the following notation. For a formula $\varphi$, let $\mathcal{U}(\varphi)$ denote the following language:

$$\mathcal{U}(\varphi) := \bigcup_{\Sigma \text{ is appropriate for } \varphi} M_\Sigma(\varphi)$$

Note that since we assume that every alphabet $\Sigma$ is contained in the infinite set $\mathbb{U}$, the language $\mathcal{U}(\varphi)$ is a language over the infinite alphabet $\mathbb{U}$.

To separate $\mathcal{P}^{f_{k+1}}$ from $\mathcal{P}^{f_k}$, we use the following pattern:

$$\psi_{k+1} := \hat{}\ a \bullet ?_1 \, , ?_1 \bullet ?_2 \, , ?_2 \bullet ?_3 \, , ?_3 \ \ldots \ ?_k \, , ?_k \bullet ?_{k+1} \, , ?_{k+1} \bullet b \ \$$$

Intuitively, for every alphabet $\Sigma$, the language $M_\Sigma(\psi_{k+1})$ contains words of the form:

$$ac_1 \ w_1 \ c_1c_2 \ w_2 \ c_2c_3 \ w_3 \ \cdots \ w_k \ c_kc_{k+1} \ w_{k+1} \ c_{k+1}b, \tag{1}$$

where $a, b, c_1, \ldots, c_{k+1} \in \Sigma$ and $w_1, \ldots, w_{k+1} \in \Sigma^*$.

We will show that $\psi_{k+1}$ cannot be expressed with any formula in $\mathcal{P}^{f_k}$. The proof is via the notion of weak pebble automata, a well known model of computation for languages over infinite alphabets. (We will review its definition shortly.) The connection between the pattern logic $\mathcal{P}^{f_k}$ and weak pebble automata is established in the following lemma.

**Lemma 1** *For every integer $k \geq 1$, and for every formula $\varphi$ in $\mathcal{P}^{f_k}$, there exists a weak $(k+1)$ pebble automaton $\mathcal{A}_\varphi$ such that the language accepted by $\mathcal{A}_\varphi$ is $\mathcal{U}(\varphi)$.*

We note that the language $\mathcal{U}(\psi_{k+1})$ is a language that has been denoted by $\mathcal{R}^+_{k+2}$ by Tan [21]. Tan has proved that $\mathcal{R}^+_{k+2}$ cannot be recognized by any weak $(k+1)$ pebble automaton [21, Lemma 4.3].[6] Thus, proving Lemma 1 implies that $\psi_{k+1}$ is not uniformly expressible by any $\mathcal{P}^{f_k}$ formula, and hence, establishes Theorem 4.

In the rest of this section we will present the formal treatment. In Subsection 3.2 we will review the definition of weak PA. Then, in Subsection 3.3, we will formally prove Lemma 1. Finally, we will present the formal proof of Theorem 4 in Subsection 3.4.

---

[6] The definition of $\mathcal{R}^+_{k+2}$ considered in [21] contains all words of the form (1) where each $c_i$ does not appear in $w_i$. However, this is merely a more restricted version of the language $\mathcal{U}(\psi_{k+1})$. The proof in [21, Lemma 4.3] still trivially holds even if we drop the requirement that each $c_i$ does not appear in $w_i$.

### 3.2 Review of weak pebble automata

Recall that we assume an infinite alphabet $\mathbb{U}$. We reserve two special symbols $\triangleleft$ and $\triangleright$, which we assume do not belong to $\mathbb{U}$. They will be used as the start and end markers of the input strings of our automata.

**Definition 3** [13,20] A *weak k-pebble automaton* (in short, $k$-PA) over $\mathbb{U}$ is a system $\mathcal{A} = \langle Q, q_0, F, \mu \rangle$ whose components are defined as follows.

- $Q$ is a finite set of *states*; $q_0 \in Q$ is the *initial state*; and $F \subseteq Q$ is the set of *final states.*
- $\mu \subseteq \mathcal{C} \times \mathcal{D}$ is a *finite* set of transitions, where $\mathcal{C}$ is the set of elements of the form $(i, \sigma, q)$ or $(i, V, q)$, where
  - $1 \leq i \leq k$;
  - $\sigma \in \mathbb{U} \cup \{\triangleleft, \triangleright\}$;
  - $V \subseteq \{1, \ldots, i-1\}$;
  - $q \in Q$;

  and $\mathcal{D}$ is the set of elements of the form $(q, \texttt{act})$, where $q \in Q$ and `act` is either `stay`, `right`, `place-pebble` or `lift-pebble`.

Elements of $\mu$ will be written as $\alpha \to \beta$, where $\alpha \in \mathcal{C}$ and $\beta \in \mathcal{D}$.

We assume that the input to $\mathcal{A}$ is always of the form $\triangleleft w \triangleright$, where $w \in \mathbb{U}^*$.

For a word $w = d_1 \cdots d_n \in \mathbb{U}^*$, with each $d_j \in \mathbb{U}$, a *configuration* of $\mathcal{A}$ on $w$ is a triple $[i, q, \theta]$, where $i \in \{1, \ldots, k\}$, $q \in Q$, and $\theta : \{1, \ldots, i\} \to \{0, 1, \ldots, n, n+1\}$. The number $i$ denotes the active pebble and $q$ the current state. The function $\theta$ defines the position of the pebbles and is called the *pebble assignment.* When $\theta$ assigns 0 or $n+1$ to a pebble, it means that the pebble is currently placed on the start symbol $\triangleleft$ or on the end symbol $\triangleright$, respectively. We assume that the head pebble never moves beyond the end symbol $\triangleright$. That is, it never moves right once it reads $\triangleright$.

The *initial* configuration is $\gamma_0 = [1, q_0, \theta_0]$, where $\theta_0(1) = 0$ is the *initial* pebble assignment. A configuration $[i, q, \theta]$ with $q \in F$ is called an *accepting* configuration.

A transition $(i, \sigma, p) \to \beta$ applies to a configuration $[j, q, \theta]$ (on $w$), if $i = j$, $p = q$ and $d_{\theta(i)} = \sigma$. A transition $(i, V, p) \to \beta$ applies to a configuration $[j, q, \theta]$, if $i = j$, $p = q$, $V = \{l < i : d_{\theta(l)} = d_{\theta(i)}\}$ and there is no transition of the form $(i, \sigma, p)$ that applies to it. Note that in a configuration $[i, q, \theta]$, pebble $i$ is in control, serving as the head pebble.

Next, we define the transition relation $\vdash_{\mathcal{A},w}$ on configurations as follows: $[i, q, \theta] \vdash_{\mathcal{A},w} [i', q', \theta']$, if there is a transition $\alpha \to (p, \texttt{act}) \in \mu$ that applies to $[i, q, \theta]$ on $w$ such that $q' = p$, $\theta'(j) = \theta(j)$, for all $j \leq i$, and

- if $\texttt{act} = \texttt{right}$, then $\theta(i) < n+1$, $i' = i$ and $\theta'(i) = \theta(i) + 1$;
- if $\texttt{act} = \texttt{lift-pebble}$, then $i > 1$ and $i' = i - 1$;
- if $\texttt{act} = \texttt{place-pebble}$, then $i < k$, $i' = i + 1$, and $\theta'(i+1) = \theta'(i) = \theta(i)$.

As usual, we denote the reflexive transitive closure of $\vdash_{\mathcal{A},w}$ by $\vdash^*_{\mathcal{A},w}$. When the automaton $\mathcal{A}$ is clear from the context, we will omit the subscript $\mathcal{A}$.

We finally say that $w$ is accepted by $\mathcal{A}$, if there is an accepting configuration $[i, q, \theta]$ such that $[1, q_0, \theta_0] \vdash^*_{\mathcal{A},w} [i, q, \theta]$. The language $L(\mathcal{A})$ consists of all the words accepted by $\mathcal{A}$.

*Remark 4* There are a few points worth stating here.

- In this paper we only consider non-deterministic weak PA, which is sufficient for our purpose. Moreover, it has already been shown [20,21] that for every integer $k \geq 1$, deterministic, non-deterministic and alternating weak $k$-PA are equivalent in expressive power.
- We note that there are two version of PA: strong PA and weak PA [13,21]. In strong PA, when a new pebble is placed, it is placed at the beginning of the input word, whereas in weak PA, at the same position of the head pebble. They are not equivalent in expressive power. As the name indicates, strong PA is more expressive than weak PA. Moreover, in the original definition of PA, the automaton can detect whether some pebbles occupy the same position. In weak PA, such capability does not add any expressive power, hence, omitted in Definition 3 above.
- Finally, in [20,21], the pebbles are numbered starting from $k$ and going down to 1. That is, it starts with pebble $k$ and when pebble $j$ is the head pebble, it places pebble $j-1$. The order is reversed in this paper, where we start from pebble 1 and go up to pebble $k$. The purpose is only to present a neater proof, where we can match each wildcard $?_i$ with pebble $i$.

### 3.3 Proof of Lemma 1.

We begin by noting the following lemma which can be proven by induction.

**Lemma 2** *Let $\Sigma \subseteq \Sigma'$, let $\varphi$ be a $\mathcal{P}^{f_k}$-formula over $\Sigma$, and let $s \in \Sigma^*$. Then $s \in M_\Sigma(\varphi)$ if and only if $s \in M_{\Sigma'}(\varphi)$.*

*Proof* For any basic pattern $\beta$ and any assignment $\xi : \{?_1, \dots, ?_k\} \to \Sigma$, we can prove by induction on $\beta$ that $s \in L^\xi_\Sigma(\beta)$ iff $s \in L^\xi_{\Sigma'}(\beta)$. Note also that if $s \in L^\xi_{\Sigma'}(\beta)$ for some $\xi : \{?_1, \dots, ?_k\} \to \Sigma'$, then the range of $\xi$ must lie in $\Sigma$, because $s \in \Sigma^*$. It follows that $s \in L_\Sigma(\beta)$ iff $s \in L_{\Sigma'}(\beta)$. Proceeding similarly by induction on $\varphi$, we can now show the statement of the lemma. □

In the following, let $\varphi$ be a formula in $\mathcal{P}^{f_k}$. By the above lemma, and since weak $k$-PA languages are closed under the boolean operators [21], it is sufficient to prove Lemma 1 where $\varphi$ is a single pattern of the form: $\hat{}\,\beta\,\$$, $\hat{}\,\beta$, $\beta\,\$$ and $\beta$, where $\beta$ is a basic pattern.

Let $\Sigma$ be the set of symbols that appear in $\varphi$. We will first consider the case when $\varphi$ is of the form:

$$\hat{}\ c_1 \circledast_1 c_2 \circledast_2 \cdots c_m \circledast_m c_{m+1}\,\$$$

where each $c_i$ is either a symbol $a \in \Sigma$ or $?_j \setminus Z$ and each $\circledast_i$ is either **.** or **,**. By renaming the wildcards, if necessary, we may assume that the first appearance of $?_i$ is to the left of the first appearance of $?_{i+1}$.

The automaton $\mathcal{A}_\varphi$ is non-deterministic and intuitively, it is defined as follows. The set of states is $\{q_0, q_1, \ldots, q_{m+2}, q_{m+3}, p\}$, where $q_0$ is the initial state and $q_{m+3}$ is the only final state. The state $p$ is the "sink" state, from which the automaton can never get out from. The purpose of each state $q_i$ is to verify whether the pattern starting from $c_i$, i.e., $c_i \circledast_i \cdots \circledast_m c_{m+1}\,\$$, is satisfied.

- If $c_i$ is a symbol from $\Sigma$, then it can only enter state $q_{i+1}$ by reading $c_i$.
- If $c_i$ is $\texttt{?}_j \setminus Z$ and $\texttt{?}_j$ has not appeared before, then it places a new pebble here, moves right and enters state $q_{i+1}$. Moreover, if it reads any of the symbol from $Z$, it enters the "sink" state $p$.
  Due to the assumption that the first appearance of $\texttt{?}_j$ is before the first appearance of $\texttt{?}_{j+1}$, the head pebble can only be pebble $j$. After it places new pebble, the head pebble becomes pebble $j+1$. Furthermore, this also implies that the interpretation of each $\texttt{?}_j$ is simulated by the symbol in the position where pebble $j$ is placed.
- If $c_i$ is $\texttt{?}_j \setminus Z$ and $\texttt{?}_j$ has appeared before, then it verifies that the head pebble reads the same symbol as pebble $j$, moves right and enters state $q_{i+1}$. If it reads any of the symbol from $Z$, it enters the "sink" state $p$.
- If $\circledast_i$ is the comma operator $\textbf{,}$, the head can move right for an arbitrary number of times.

We now present the formal definition of the transitions in $\mathcal{A}_\varphi$. In the following for each $i \in \{1, \ldots, m+1\}$, we define the index $l_i$:

$$l_i := \max(\{j \mid \texttt{?}_j \text{ appears in } c_1, \ldots, c_{i-1}\} \cup \{0\}).$$

That is, $l_i$ is the maximal index of the wildcards $\texttt{?}_j$'s that has already appeared in $c_1, \ldots, c_{i-1}$. If none of the wildcard appear in $c_1, \ldots, c_{i-1}$, then $l_i = 0$.

First, it contains the following two transitions:

$$(1, \triangleleft, q_0) \to (q_1, \texttt{right}) \text{ and } (l_{m+2}+1, \triangleright, q_{m+2}) \to (q_{m+3}, \texttt{right})$$

For each $i \in \{1, \ldots, m+1\}$, it contains the following transitions.

- If $c_i \in \Sigma$, then $\mu$ contains the transition $(l_i + 1, c_i, q_i) \to (q_{i+1}, \texttt{right})$.
- If $c_i \in \texttt{?}_j \setminus Z$ and $j \le l_i$ (i.e., $\texttt{?}_j$ has already appeared before), then $\mu$ contains the transitions:
  - $(l_i + 1, V, q_i) \to (q_{i+1}, \texttt{right})$, for every $V \ni j$.
  - $(l_i + 1, \sigma, q_i) \to (p, \texttt{right})$, for every $\sigma \in Z$.
- If $c_i \in \texttt{?}_j \setminus Z$ and $j > l_i$ (i.e., $\texttt{?}_j$ has not already appeared before), then $\mu$ contains the transitions:
  - $(l_i + 1, V, q_i) \to (q_{i+1}, \texttt{place-pebble})$, for every $V$.
  - $(l_i + 1, \sigma, q_i) \to (p, \texttt{right})$, for every $\sigma \in Z$.
- If $\circledast_i$ is the comma operator $\textbf{,}$, then $\mu$ contains the transitions:
  - $(l_i + 1, V, q_{i+1}) \to (q_{i+1}, \texttt{right})$, for every $V$.
  - $(l_i + 1, \sigma, q_{i+1}) \to (q_{i+1}, \texttt{right})$, for every $\sigma \in \Sigma$.

For the cases when the pattern $\varphi$ is without the start and end markers `^` or `$`, the automaton $\mathcal{A}_\varphi$ can be defined similarly. Without `^`, the head can move right for an arbitrary number of times before entering $q_1$, and without `$`, the head can move right for an arbitrary number of times before entering $q_{m+3}$.

### 3.4 Proof of Theorem 4

Suppose there is a formula $\varphi \in \mathcal{P}^{f_k}$ such that for every finite alphabet $\Sigma$ appropriate for both $\varphi$ and $\psi_{k+1}$, $M_\Sigma(\varphi) = M_\Sigma(\psi_{k+1})$. This implies $\mathcal{U}(\varphi) = \mathcal{U}(\psi_{k+1})$. By Lemma 1, the language $\mathcal{U}(\psi_{k+1})$ is accepted by a weak $(k+1)$-PA, which contradicts a known result [21, Lemma 4.3].

## 4 Comma-regular expressions

The theme of this paper is to compare substring constraints to subsequence constraints. Classical regular expressions are based on the concatenation operator and thus slanted more towards substring constraints (although subsequence constraints are certainly expressible). It is therefore natural to ask what happens with regular expressions when we replace the concatenation operator by the comma operator that we used in Section 2.1 to define the semantics of subsequence patterns: $S \mathbin{,} T = S \cdot \Sigma^* \cdot T$.

Thus, define the *comma-regular expressions*, denoted by $\mathrm{RE}^{,}$, just like the regular expressions as recalled in Section 2.3, except that we replace $e_1 \cdot e_2$ by $e_1 \mathbin{,} e_2$. Moreover, we leave out Kleene star.

Leaving out Kleene star is explained as follows. In the "world of comma", we would want to modify the semantics of Kleene star so as to be based not on concatenation but on the comma operator. Classical Kleene star is the closure of a set under concatenation. Accordingly, for any set $S$ of sequences, let us now redefine $S^+$ to be the smallest superset of $S$ that is closed under the comma operator in the sense that if $s \in S^+$ and $t \in S^+$ then $\{s\} \mathbin{,} \{t\} \subset S^+$. However, it turns out that this is already definable:

**Proposition 3** *The modified $S^+$ equals $S \cup (S \mathbin{,} S)$.*

*Proof* Clearly, $(S \cup (S \mathbin{,} S)) \subseteq S^+$. Conversely, also $S^+ \subseteq (S \cup (S \mathbin{,} S))$. Indeed, we verify that $S \cup (S \mathbin{,} S)$ is closed under the comma operator. Let $w \in \{u\} \mathbin{,} \{v\}$, where $u, v \in S \cup (S \mathbin{,} S)$. This means that $w$ has a prefix and a suffix from $S$. Thus, $w \in (S \mathbin{,} S)$ as desired. □

So, we continue without Kleene star. In particular, $\mathrm{RE}^{,}$ falls within the star-free regular languages.

Let us first look at the fragment of $\mathrm{RE}^{,}$ without $\varepsilon$, and without the intersection and the complementation operators. We denote this fragment by $\mathrm{disjRE}^{,}$. Indeed, intersection and complementation are neither present in the classical regular expressions, although there, over any fixed alphabet, they are definable using the other operators.[7]

We next show that intersection is actually definable in disjRE$^{\boldsymbol{,}}$. By the positive fragment of a logic $\mathcal{P}^f$, denoted by pos$\mathcal{P}^f$, we mean all $f$-formulas that do not use negation (only conjunction and disjunction). In the disjunctive fragment, denoted by disj$\mathcal{P}^f$, we only use disjunction. We have the following characterization:

**Theorem 5** *The following four logics have equal expressiveness uniformly:*

1. *pos*$\mathcal{P}^{\{\boldsymbol{,},\hat{}\$,?\}}$*;*
2. *disj*RE$^{\boldsymbol{,}}$ *extended with intersection;*
3. *disj*RE$^{\boldsymbol{,}}$*;*
4. *disj*$\mathcal{P}^{\{\boldsymbol{,},\hat{}\$,?\}}$.

*Proof* We will show $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 1$, where the implication $i \Rightarrow j$ means that the logic in item $j$ is uniformly more expressive than the logic in item $i$.

For the implication $1 \Rightarrow 2$, basic patterns in pos$\mathcal{P}^{\{\boldsymbol{,},\hat{}\$,?\}}$ can be literally viewed as expressions in disjRE$^{\boldsymbol{,}}$. Also, disjunction and conjunction can be translated to union and intersection. So we only need to show how a pattern $\alpha$ in pos$\mathcal{P}^{\{\boldsymbol{,},\hat{}\$,?\}}$ is uniformly expressible by an expression $e_\alpha$ in disjRE$^{\boldsymbol{,}}$. This is shown in the following table. Here, $\beta$ stands for a basic pattern. As always, ? abbreviates $? \setminus \emptyset$.

| $\alpha$ | $e_\alpha$ |
|---|---|
| $\beta$ | $\beta \cup (? \boldsymbol{,} \beta) \cup (\beta \boldsymbol{,} ?) \cup (? \boldsymbol{,} \beta \boldsymbol{,} ?)$ |
| $\hat{}\beta$ | $\beta \cup (\beta \boldsymbol{,} ?)$ |
| $\beta\,\$$ | $\beta \cup (? \boldsymbol{,} \beta)$ |
| $\hat{}\beta\,\$$ | $\beta$ |

For the implication $2 \Rightarrow 3$, let $e$ be an expression from disjRE$^{\boldsymbol{,}}$. We claim that $e$ can be rewritten into a union of expressions of the form: $(c_1 \boldsymbol{,} c_2 \boldsymbol{,} \cdots \boldsymbol{,} c_n)$, where each $c_i$ is either a letter or a wildcard.

We begin by normalizing the expression $e$ so that each wildcard is of the form $? \setminus \Sigma$ with $\Sigma$ the alphabet of all letters actually used in $e$. Such normalisation is always possible using union, similar to Example 5.

The proof of the claim is by induction on $e$. The base case is either:

- $e$ is a single expression of the form $(a_1 \boldsymbol{,} a_2 \boldsymbol{,} \cdots \boldsymbol{,} a_k)$, where each $a_i$ is either a letter or a wildcard, or
- $e$ is $e_1 \cap e_2$, where $e_1$ and $e_2$ are of the form: $(a_1 \boldsymbol{,} a_2 \boldsymbol{,} \cdots \boldsymbol{,} a_k)$ and $(b_1 \boldsymbol{,} b_2 \boldsymbol{,} \cdots \boldsymbol{,} b_l)$, respectively, and each $a_i$ and $b_j$ are either letters or wildcards.

The first case is of course trivial. We prove the second case. If $a_1 \neq b_1$ or $a_k \neq b_l$, then $e$ can be rewritten as $\emptyset$. So, suppose that $a_1 = b_1$ and $a_k = b_l$. We define the expression $e'$ which is the union of all the expressions $(c_1 \boldsymbol{,} c_2 \boldsymbol{,} \cdots \boldsymbol{,} c_m)$, where $m \leq kl$ and there is a mapping $\xi_1 : \{1, \ldots, k\} \to \{1, \ldots, m\}$ and $\xi_2 : \{1, \ldots, l\} \to \{1, \ldots, m\}$ such that the following holds.

[7] Uniformly over all alphabets, or over an infinite alphabet, this is another matter [10].

- Both $\xi_1$ and $\xi_2$ are strictly increasing, i.e., whenever $i < i'$ and $j < j'$, $\xi_1(i) < \xi_1(i')$ and $\xi_2(j) < \xi_2(j')$.
- $\xi_1(1) = \xi_2(1) = 1$.
- $\xi_1(k) = \xi_2(l) = m$.
- For each $i \in \{1, \dots, k\}$, $a_i = c_{\xi_1(i)}$.
- For each $i \in \{1, \dots, l\}$, $b_i = c_{\xi_2(i)}$.
- For each $j \in \{1, \dots, m\}$, there is $i$ such that $j = \xi_1(i)$ or $j = \xi_2(i)$.

Since $e'$ is the union of all expressions $(c_1 \mathbin{\textbf{,}} \cdots \mathbin{\textbf{,}} c_m)$ with the properties above, it is rather obvious that $e'$ captures the intersection $e_1 \cap e_2$.

The induction step is rather straightforward, since both comma and intersection distribute over union.

For the implication 3 ⇒ 4, since comma distributes over union, we can rewrite a disjRE$^{\textbf{,}}$ expression as a union of expressions of the form $(a_1 \mathbin{\textbf{,}} \cdots \mathbin{\textbf{,}} a_n)$, where each $a_i$ is either a letter or a wildcard. To obtain the desired formula it now suffices to replace union by disjunction, and to add ^ and $, respectively, at the beginning and end of each expression $(a_1 \mathbin{\textbf{,}} \cdots \mathbin{\textbf{,}} a_n)$.

The implication 4 ⇒ 1 is trivial. □

We now know that disjRE$^{\textbf{,}}$ is a fairly weak logic as it is equivalent to pos$\mathcal{P}^{\{\textbf{,},\hat{}\$\}}$ over any fixed alphabet. What if we move to RE$^{\textbf{,}}$, i.e., add complementation? Surprisingly we make a big jump in expressiveness, and get all star-free regular languages.

**Theorem 6** RE$^{\textbf{,}}$ *is equally expressive as* RE$_0$ *over every fixed alphabet.*

*Proof* We rely on the characterisation of RE$_0$ as the smallest family of languages containing all finite languages, closed under the boolean operations, and closed under the operations $L \to La\Sigma^*$ and $L \to \Sigma^* aL$, as presented in [15, Theorem 7.11].

To illustrate how any fixed sequence can be described, consider the sequence $abc$. This sequence can be defined as $a \mathbin{\textbf{,}} b \mathbin{\textbf{,}} c \cap (?, ?, ?, ?)^{\mathrm{c}}$.

To show closure under $L \to La\Sigma^*$ it suffices to show closure under $L \to La$ as $La\Sigma^* = La \mathbin{\textbf{,}} \varepsilon$. But this is readily verified by induction: $(e_1 \cup e_2) \mathbin{\bullet} a = (e_1 \mathbin{\bullet} a) \cup (e_2 \mathbin{\bullet} a)$; $(e_1 \mathbin{\textbf{,}} e_2) \mathbin{\bullet} a = e_1 \mathbin{\textbf{,}} (e_2 \mathbin{\bullet} a)$; and $e^{\mathrm{c}} \mathbin{\bullet} a = (\varepsilon \mathbin{\textbf{,}} a) \cap (e \mathbin{\bullet} a)^{\mathrm{c}}$. Closure under $L \to \Sigma^* aL$ is symmetric. □

## 5 Comparison with first-order logic

In this section we will present a detailed comparison between the pattern logic and first-order logic. A non-empty word $w = d_1 \cdots d_n \in \Sigma^*$ of length $n$ can be viewed as a mathematical structure with domain $[n] = \{1, \dots, n\}$ and the following relations.

- The order relation $<$ to be interpreted in the standard way.
- An equivalence relation $\sim$, where $i \sim j$ if and only if $d_i = d_j$.
- Each letter $a \in \Sigma$ defines a unary relation where $a(i)$ holds if and only if $d_i = a$.

We will consider the first-order logic (FO) for finite non-empty strings, i.e., the class of first-order sentences with atomic predicates: $x < y$, $x \sim y$ and $a(x)$, for each letter $a \in \mathbb{U}$, where $x$ and $y$ are first-order variables. We say that a word $w$ match a sentence $\varphi \in$ FO, if $\varphi$ holds in $w$ viewed as a mathematical structure. For more details, see, e.g., [23,13].

In this section we are going to compare the expressiveness of the pattern logics with FO. Note that Definition 2 can be easily adapted to include FO, but with the empty string $\varepsilon$ excluded from $\mathrm{Mod}_{\Sigma}(\varphi)$. The notion of appropriate alphabet for an FO sentence $\varphi$ can be defined similarly: An alphabet $\Sigma$ is appropriate for $\varphi$, if it contains all the letters in $\varphi$.

Theorem 7 below states over a fixed alphabet, FO is more expressive than $\mathcal{P}^{\{\bullet, \mathbf{,}, ?, \hat{}\$\}}$, unless the alphabet contains only one letter.

**Theorem 7**

- *Over an alphabet $\Sigma$ that contains only one letter, $\mathcal{P}^{\{\bullet, \mathbf{,}, ?, \hat{}\$\}}$ and FO are equally expressive over $\Sigma$.*
- *Over an alphabet $\Sigma$ that contains at least two letters, FO is strictly more expressive than $\mathcal{P}^{\{\bullet, \mathbf{,}, ?, \hat{}\$\}}$.*

*Proof* It is a well known fact that FO and $\mathrm{RE}_0$ are equally expressive over any alphabet [12]. It has been shown that the dot-depth hierarchy is infinite for alphabets containing at least two letters, but collapses to level 1, i.e., $\mathrm{RE}_0^1$, when the alphabet contains only one letter [2]. Since we have shown in Proposition 1 that $\mathcal{P}^{\{\bullet, \mathbf{,}, ?, \hat{}\$\}}$ and $\mathrm{RE}_0^1$ are equally expressive uniformly, our theorem follows immediately. $\square$

Similarly, we can show that FO is strictly more expressive uniformly than $\mathcal{P}^{f_k}$, for any integer $k \geq 1$, as stated below.

**Theorem 8**

1. *For every integer $k \geq 1$, FO is uniformly more expressive than $\mathcal{P}^{f_k}$.*
2. *There is an FO sentence $\psi$ that is not uniformly expressible by any $\mathcal{P}^{f_k}$ formula, for any integer $k \geq 1$.*

*Hence, FO is strictly more expressive than $\mathcal{P}^{f_k}$, for every integer $k \geq 1$.*

*Proof* For the first part, let $\varphi$ be $\mathcal{P}^{f_k}$ formula. It suffices to show when $\varphi$ is a pattern. We will first consider the case when $\varphi$ is of the form:

$$\hat{}\ c_1 \circledast_1 c_2 \circledast_2 \cdots c_m \circledast_m c_{m+1}\ \$$$

where each $c_i$ is either a letter $a$ or $?_j \setminus Z$ and each $\circledast_i$ is either $\bullet$ or $\mathbf{,}$.

The FO sentence that expresses $\varphi$ uniformly states as follows. There is $z_1, \ldots, z_{m+1}$ such that the following holds.

1. $z_1$ is the minimum, i.e., for all $x$, either $x = z_1$ or $x > z_1$.
2. $z_{m+1}$ is the maximum, i.e., for all $x$, either $x = z_{m+1}$ or $x < z_{m+1}$.
3. For each $i \in \{1, \ldots, m\}$,

  - if $\circledast_i = \mathbf{,}$, then $z_i < z_{i+1}$;
  - if $\circledast_i = \bullet$, then $z_i + 1 < z_{i+1}$ and for all $y \neq z_i, z_{i+1}$, either $y < z_i$ or $y > z_{i+1}$.
4. For each $i \in \{1, \ldots, m+1\}$,
  - if $c_i = a$, then the label in position $z_i$ is $a$;
  - if $c_i = \texttt{?}_j \setminus Z$, then the label in position $z_i$ is not from $Z$.
5. For each $i \neq i' \in \{1, \ldots, m+1\}$, if $c_i$ is $\texttt{?}_j \setminus Z$ and $c_{i'}$ is $\texttt{?}_j \setminus Z'$, for some $Z, Z'$, then $z_i \sim z_{i'}$.

Condition (1) is dropped, if the pattern $\varphi$ does not start with ^. Likewise, condition (2) is dropped, if $\varphi$ does not end with $.

Next, we show the second part. We use the same technique as in Section 3.1 by reducing it to a known result for pebble automata. Let # be a letter, and consider the language $L_{\supseteq}$ that consists of all words of the form: $u\#v$, where # does not appear in either $u$ or $v$ and every letter that appears in $v$ also appears in $u$.

The language $L_{\supseteq}$ can be expressed by an FO sentence $\psi_{\supseteq}$ that states the following. There exists a position $x$ such that the following holds.

- The letter in position $x$ is # and it does not appear elsewhere.
- For every position $y > x$, there is $z < x$ such that $x \sim y$.

Now, if $\psi_{\supseteq}$ is expressible uniformly by a $\mathcal{P}^{f_k}$ formula $\varphi$, by Lemma 1, $\mathcal{U}(\varphi) = L_{\supseteq}$ is accepted by a weak $(k+1)$-PA. This contradicts the fact that $L_{\supseteq}$ is not accepted by any weak pebble automata [14,8]. □

## 6 Conclusions

Starting from the basic dichotomy between substring and subsequence constraints, our goal has been to understand the expressiveness of very simple, user-friendly sequence pattern logics.

Moreover, we have extended our pattern logic with multiple wildcards and have given a connection between our logic and weak pebble automata, a model of computation over infinite alphabets. This connection allows us to establish a strict hierarchy of expressiveness based on the number of wildcards.

**Acknowledgements** We would like to thank the anonymous referees for their careful and helpful comments in improving our paper. We also thank Frank Neven for suggesting the connection to locally testable languages, and Jean-Eric Pin for his encouragement and help in proving Theorem 6.

## References

1. J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logic und Grundlagen der Mathematik*, 6:66–92, 1960.
2. J.A. Brzozowski and R. Knast. The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences*, 16, 1978.

3. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
4. R.S. Cohen and J.A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.
5. G. Dong and J. Pei. *Sequence Data Mining*. Springer, 2007.
6. J.M. Patel (editor). Special issue on querying biological sequences. *IEEE Data Engineering Bulletin*, 27(3), 2004.
7. Ch. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 419–429, 1994.
8. D. Genkin, M. Kaminski, and L. Peterfreund. Closure Under Reversal of Languages over Infinite Alphabets. In F. Fomin and V. Podolskii, editor, *Computer Science Symposium in Russia, Proceedings* (CSR), volume 10846 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2018.
9. H.V. Jagadish et al. Making database systems usable. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 13–24, 2007.
10. M. Kaminski and T. Tan. Regular expressions for languages over infinite alphabets. *Fundamenta Informaticae*, 69:301–318, 2006.
11. A. Loeffen. Text databases: A survey of text models and systems. *SIGMOD Record*, 23(1):97–106, 1994.
12. R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.
13. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.
14. L. Peterfreund. Closure under reversal of languages over infinite alphabets: a case study. Master thesis, Department of Computer Science, Technion – Israel Institute of Technology (2015).
15. J.E. Pin. Syntactic semigroups. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 10. Springer, 1997.
16. J.E. Pin. The Dot-Depth Hierarchy, 45 Years Later. *The Role of Theory in Computer Science* 2017: 177–202.
17. T. Place, L. van Rooijen, M. Zeitoun: Separating Regular Languages by Locally Testable and Locally Threshold Testable Languages. *Logical Methods in Computer Science* 10(3) (2014).
18. L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In Z. Ésik, editor, *Computer Science Logic, Proceedings* (CSL), volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006.
19. I. Simon. Piecewise testable events In H. Barkhage, editor, *Automata Theory and Formal Languages, Proceedings*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 1975.
20. T. Tan. On pebble automata for data languages with decidable emptiness problem. *Journal of Computer and System Sciences*, 76(8):778–791, 2010.
21. T. Tan. Graph reachability and pebble automata over infinite alphabets. *ACM Transactions on Computational Logic*, 14(3):article 19, 2013.
22. W. Thomas. A concatenation game and the dot-depth hierarchy. In *Computation Theory and Logic*, volume 270 of *Lecture Notes in Computer Science*, pages 415–426. Springer-Verlag, 1987.
23. W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7. Springer, 1997.
24. J.T.L. Wang, B.A. Shapiro, and D. Shasha, editors. *Pattern Discovery in Biomolecular Data*. Oxford University Press, 1999.