Automatic Generation of Descriptions of Time-Series Constraints

María Andreína Francisco Rodríguez, Pierre Flener, and Justin Pearson
Uppsala University, Department of Information Technology, SE – 751 05 Uppsala, Sweden
{Maria.Andreina.Francisco, Pierre.Flener, Justin.Pearson}@it.uu.se

***Abstract*—Integer time series are often subject to constraints on the aggregation of the features of all occurrences of some pattern within the series. For example, the number of inflexions may be constrained, or the sum of the peak maxima, or the minimum of the valley widths. Many time-series constraints can be described by transducers. The output alphabet of such a transducer consists of symbols that denote the phases of identifying the maximal occurrences of a pattern. It was recently shown how to synthesise automatically a constraint propagator and a constraint checker from such a transducer, which however has to be designed manually from a pattern. Here we define a large class of patterns, present an algorithm for automatically generating a low-level transducer from such a high-level pattern, and prove it correct. This class covers all 20 patterns of the Time-Series Constraint Catalogue, which can now be automatically extended at will.**

***Index Terms*—time-series constraints, automata, transducers.**

I. Introduction

A *time series* is here a sequence of integers, corresponding to measurements taken over a time interval. Time series are common in many application areas, such as the output of electric power stations over multiple days [1], the manpower required in a call centre [2], or the daily capacity of a hospital clinic over a period of years. Time series are often constrained by physical or organisational limits.

In [3] it was shown that many useful constraints $\gamma(\langle X_1, \ldots, X_n\rangle, N)$ on an unknown time series $X = \langle X_1, \ldots, X_n\rangle$ of given length n can be specified by a triple $\langle \pi, f, g\rangle$, where π is called a *pattern* and in this introductory section is a regular expression over the alphabet {'<', '=', '>'} (we assume the reader is familiar with regular expressions and automata [4]), while $f \in \{\texttt{max}, \texttt{min}, \texttt{one}, \texttt{surface}, \texttt{width}\}$ is called a *feature*, and $g \in \{\texttt{Max}, \texttt{Min}, \texttt{Sum}\}$ is called an *aggregator*; integer variable N is constrained to be the aggregation, computed using g, of the list of values of feature f for all maximal words matching π in X. We name a time-series constraint predicate specified by $\langle \pi, f, g\rangle$ as g_f_π.

Let the sequence $S = \langle S_1, \ldots, S_{n-1}\rangle$, called the *signature* and containing *signature variables*, be linked to a time series $X = \langle X_1, \ldots, X_n\rangle$ via the *signature constraints* $(X_i < X_{i+1} \Leftrightarrow S_i = \text{'<'}) \wedge (X_i = X_{i+1} \Leftrightarrow S_i = \text{'='}) \wedge (X_i > X_{i+1} \Leftrightarrow S_i = \text{'>'})$ for all $i \in [1, n-1]$.

Example 1: The time series $X = \langle 4, 4, 0, 0, 2, 4, 4, 7, 4, 1, 1, 5, 5, 5, 5, 5, 5, 3\rangle$ has the signature $S =$ '=>=<<=<>>=<=====>'. Consider the regular expression $\texttt{Peak} =$ '<(<|=)*(>|=)*>': a *peak* within a time series corresponds to a maximal word matching Peak in the signature. The max feature value of a peak is its highest value. The time series X contains two peaks, namely $\langle 0, 2, 4, 4, 7, 4, 1\rangle$ and $\langle 1, 5, 5, 5, 5, 5, 5, 3\rangle$, visible in Figure 1, of highest values 7 and 5 respectively. Hence the lowest peak, obtained by using the aggregator Min, has as highest value $N = 5$. The underlying constraint is MIN_MAX_PEAK(X, N). □

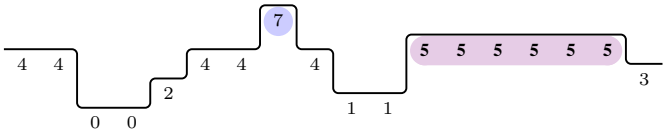

Fig. 1: Visual representation of MIN_MAX_PEAK$(X, 5)$, with $X = \langle 4, 4, 0, 0, 2, 4, 4, 7, 4, 1, 1, 5, 5, 5, 5, 5, 5, 3\rangle$.

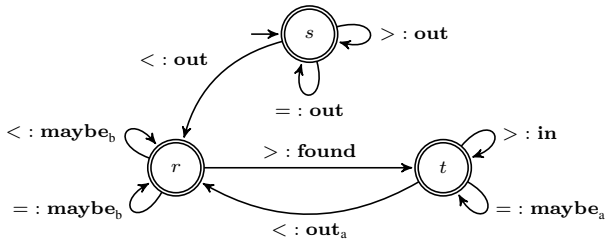

Fig. 2: Transducer for Peak = '<(<|=)*(>|=)*>'.

In [3] a set of 20 useful patterns was manually converted into transducers whose input alphabet is {'<', '=', '>'} and whose output alphabet consists of symbols that denote the phases of finding the maximal occurrences of a given pattern. For example, the transducer for the pattern corresponding to the regular expression Peak from Example 1 is in Figure 2; its notation will be explained in Section III. It was also shown in [3] how to synthesise automatically a constraint propagator and a constraint checker from such a transducer.

In this paper we show how to synthesise directly from a high-level pattern instead of a low-level transducer. The **contributions** of this paper are:

- We characterise in Section II the class of patterns that can be handled by the synthesis in [3], which makes it possible to decide when the synthesiser is applicable.
- We give in Section IV an algorithm for generating a transducer from such a pattern, and prove its correctness.

In Section V, we conclude and discuss other related work.

II. Patterns

Here patterns describe topological aspects of time series, as adjacent values of a time series are compared within the signature constraints. We refer to the domain $\Sigma = \{\text{'<'}, \text{'='}, \text{'>'}\}$ of the signature variables as the *topological alphabet*. Note that all results are independent of the chosen alphabet.

Definition 1 (regular-expression occurrence): Given a signature S and a regular expression σ over Σ, a *σ-occurrence* in S is a maximal subsignature of S that matches σ. □

Example 2: Consider again the time series $X = \langle 4, 4, 0, 0, 2, 4, 4, 7, 4, 1, 1, 5, 5, 5, 5, 5, 3\rangle$, its signature $S =$ '=>=<<=<>>=<=====>', and the regular expression `Peak` = '<(<|=)*(>|=)*>' from Example 1. The peak $\langle 0, 2, 4, 4, 7, 4, 1\rangle$ corresponds to the subsignature '<<=<>>', which is a `Peak`-occurrence because it is a maximal word in S matching `Peak`. The subsequence $\langle 4, 7, 4\rangle$ corresponds to the subsignature '<>', which is not a `Peak`-occurrence because it is not a *maximal* word in S matching `Peak`. □

We first define the general notion of pattern and then define the specific class of patterns whose transducers can be input to the synthesiser in [3]. Our formalisation is simpler than the one in [3], and none of our stated conditions are identified in [3].

A. Definition of a Pattern

For many time-series constraints, one is not interested in whole occurrences of a regular expression within the signature of a time series, but instead one wants or has to ignore prefixes of the occurrences, so that the remaining suffixes are used when computing the feature values. Before giving a motivating example, we define the resulting notion of pattern as a generalisation of a regular expression. We denote by $\mathcal{L}(\sigma)$ the regular language defined by a regular expression σ.

Definition 2 (pattern, ignore count, and pattern occurrence): A *pattern* is a pair $\pi = \langle \sigma, \beta\rangle$, where σ is a regular expression over Σ only matched by non-empty words and β is called the *ignore count*, which must be smaller than the length of a shortest word in $\mathcal{L}(\sigma)$. Given a signature S, a *π-occurrence* in S is a σ-occurrence in S except for its first β symbols. □

Example 3: Consider again the regular expression `Peak` = '<(<|=)*(>|=)*>'. The unique shortest word in $\mathcal{L}$(`Peak`) is '<>', of length 2. Therefore the ignore count β must satisfy $0 \leq \beta < 2$, because otherwise a shortest `Peak`-occurrence would become empty after ignoring the first 2 symbols. Under two ways of plotting a time series, as in Figure 3, the same subsignature '<<>>' can correspond to a peak of width 5 (left), achieved with $\beta = 0$, or a peak of width 4 (right), achieved with $\beta = 1$. □

Picking a suitable ignore count for a given regular expression is left to the modeller of the constraint problem at hand.

B. Recognisable Patterns

We now state two previously unidentified lower bounds on the ignore count of a pattern, as only transducers for such patterns can be input to the synthesiser in [3].

First, consider again the signature $S =$ '=>=<<=<>>=<=====>', but now the regular expression

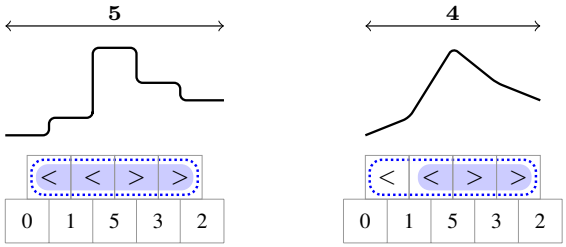

Fig. 3: Alternative plots and definitions of peaks: $\beta = 0$ (left) and $\beta = 1$ (right).

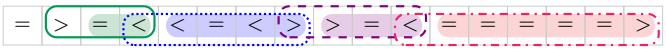

Fig. 4: Overlapping `Inflexion`-occurrences (contours) and non-overlapping $\langle$`Inflexion`$, 1\rangle$-occurrences (shaded areas).

`Inflexion` = '<(<|=)*> | >(>|=)*<'. The four `Inflexion`-occurrences in S — namely '>=<', '<<=<>', '>>=<', and '<=====>' — are in Figure 4, surrounded by solid, dotted, dashed, and dash-dotted lines, respectively. Note that for any signature with more than one `Inflexion`-occurrence, the occurrences are always next to each other and overlap in one symbol. For the correctness of the synthesised propagators and checkers, only transducers for patterns that have non-overlapping pattern occurrences can be input to the synthesiser in [3]. By ignoring the $\beta = 1$ first symbols of each `Inflexion`-occurrence, the resulting $\langle$`Inflexion`$, 1\rangle$-occurrences, indicated by the four shaded areas in Figure 4, are forced not to overlap. We now define the resulting first lower bound on the ignore count of a pattern.

Definition 3 (regular-expression overlap): Given a regular expression σ and three words w, x, y such that $xw \in \mathcal{L}(\sigma)$, $wy \in \mathcal{L}(\sigma)$, and $xwy \notin \mathcal{L}(\sigma)$, the length of w is called the *word overlap* of xw and wy. The maximum word overlap between all pairs of such words xw and wy in $\mathcal{L}(\sigma)$ is called the *σ-overlap*. If the word overlap is never defined for any pair of words in $\mathcal{L}(\sigma)$, then the σ-overlap is 0. □

For example, the `Inflexion`-overlap is 1, as just seen. Therefore, a lower bound on the ignore count β of an $\langle$`Inflexion`$, \beta\rangle$-pattern is 1, not 0.

Second, consider the regular expression '<<=$^+$>>'. The word $w =$ '<<' can be extended to a word in $\mathcal{L}$('<<=$^+$>>'), for instance the word '<<=>>'. But the longer word $wz =$ '<<<', with $z =$ '<', cannot be extended to a word in $\mathcal{L}$('<<=$^+$>>'). Note that there is a suffix of $wz =$ '<<<' of length 2, namely '<<', that can be extended to a word in $\mathcal{L}$('<<=$^+$>>'). For the correctness of the synthesised propagators and checkers, at least the first 2 symbols of any occurrence of '<<=$^+$>>' must be ignored by a transducer that can be input to the synthesiser in [3]. We now define the resulting second lower bound on the ignore count of a pattern.

Definition 4 (prefix language): The *prefix language* $\mathcal{L}_\text{p}(\sigma)$ of a regular expression σ has every word w_1 such that $w_1 w_2 \in \mathcal{L}(\sigma)$ for some word w_2. The *proper-prefix language* $\mathcal{L}_\text{pp}(\sigma)$ of a regular expression σ has every word w_1 such that

$w_1 w_2 \in \mathcal{L}(\sigma)$ for some non-empty word w_2. □

Definition 5 (mismatch overlap): Given a regular expression σ over Σ, a word $w \in \mathcal{L}_{\text{pp}}(\sigma)$, and a symbol $z \in \Sigma$, if $wz \notin \mathcal{L}_{\text{p}}(\sigma)$, then the length of the longest suffix in $\mathcal{L}_{\text{p}}(\sigma)$ of the word wz is called the *mismatch overlap* of w and z. The maximum mismatch overlap of all words in $\mathcal{L}_{\text{pp}}(\sigma)$ and all symbols in Σ is called the *mismatch overlap* of σ. □

Example 4: Consider again the regular expression '<<=$^+$>>'. We have $\mathcal{L}_{\text{p}}$('<<=$^+$>>') = $\mathcal{L}$('<|<<=*|<<=$^+$>|<<=$^+$>>') and $\mathcal{L}_{\text{pp}}$('<<=$^+$>>') = $\mathcal{L}$('<|<<=*|<<=$^+$>). The word w = '<<' is in $\mathcal{L}_{\text{pp}}$('<<=$^+$>>'), but the word wz = '<<<', with symbol z = '<', is not in $\mathcal{L}_{\text{p}}$('<<=$^+$>>'). The longest suffix of wz = '<<<' that is in $\mathcal{L}_{\text{p}}$('<<=$^+$>>') is '<<', of length 2. Hence the mismatch overlap of wz = '<<<' is 2. If we repeat this for all words in $\mathcal{L}_{\text{pp}}$('<<=$^+$>>') and all symbols z in Σ, then we have that the mismatch overlap of '<<=$^+$>>' is 2. □

We now define the class of patterns whose transducers can be input to the synthesiser in [3].

Definition 6 (Recognisable pattern): A pattern $\pi = \langle \sigma, \beta \rangle$ is *recognisable* if the ignore count β is at least the σ-overlap and at least the mismatch overlap of σ. □

Example 5: Consider again the regular expression `Inflexion` = '<(<|=)*> | >(>|=)*<'. The shortest words in $\mathcal{L}$(`Inflexion`) are of length 2, namely '<>' and '><'. The `Inflexion`-overlap is 1. The mismatch overlap of `Inflexion` is 0. Therefore, the pattern $\langle$`Inflexion`$, \beta\rangle$ is recognisable if and only if $\max(1, 0) \leq \beta < 2$, that is $\beta = 1$. □

III. Background: Transducers

Recall that a *deterministic finite transducer* [5] is a tuple $\langle Q, \Gamma, \Gamma', \delta, q_0, Q_{\text{a}} \rangle$, where Q is the set of *states*, Γ is the *input alphabet*, Γ' is the *output alphabet*, $\delta \colon Q \times \Gamma \to Q \times \Gamma'$ is the *transition function*, which must be total, $q_0 \in Q$ is the *initial state*, and $Q_{\text{a}} \subseteq Q$ is the set of *accepting states*. When $\delta(q, a) = \langle q', a' \rangle$, there is a transition from state q to state q' upon reading the input symbol a and producing the output symbol a': we write this as $q \xrightarrow{a \,:\, a'} q'$. A *deterministic finite automaton* (DFA) is a transducer without an output alphabet. In a graphical representation of a transducer or automaton, the initial state has an arrow coming from nowhere. A transition is depicted by an arrow between two states and is annotated by a consumed input symbol and, in the case of a transducer, followed by a colon and a sequence of produced output symbols. Accepting states are denoted by a double circle.

In [3] there are 20 patterns, each represented by what is called a seed transducer. A *seed transducer* is a deterministic finite transducer with only accepting states, whose input alphabet is Σ and whose output alphabet, called the *semantic alphabet*, consists of symbols that denote the phases of finding the occurrences of a given pattern. A seed transducer consumes a signature and produces a sequence of output symbols, whose purpose is to guide the synthesis of checkers and propagators. Before giving an example, we introduce the symbols of the semantic alphabet and their meaning:

- $\mathbf{found}$: the symbol consumed is in a new pattern occurrence that may have started before and may be extended.
- $\mathbf{found}_{\text{end}}$: the symbol consumed is the last symbol in a new pattern occurrence that may have started before.
- $\mathbf{maybe}_{\text{before}}$: the symbol consumed may belong to a pattern occurrence, but this must be confirmed by producing a $\mathbf{found}$ or $\mathbf{found}_{\text{end}}$.
- $\mathbf{out}_{\text{reset}}$: the symbol consumed is outside any pattern occurrence and all the $\mathbf{maybe}_{\text{before}}$ produced just before are outside any pattern occurrence.
- $\mathbf{in}$: the symbol consumed is inside a pattern occurrence for which a $\mathbf{found}$ was already produced and all symbols between the one producing such a $\mathbf{found}$ and the one being read belong to the pattern occurrence.
- $\mathbf{maybe}_{\text{after}}$: the symbol consumed may belong to a pattern occurrence for which a $\mathbf{found}$ was already produced, but this must be confirmed by producing $\mathbf{in}$ while consuming the rest of the signature.
- $\mathbf{out}_{\text{after}}$: a pattern occurrence ended at the last $\mathbf{found}$ or $\mathbf{in}$ symbol produced.
- $\mathbf{out}$: the symbol consumed is not in a pattern occurrence.

For space reasons, the subscripts after, before, end, and reset are sometimes abbreviated by their first letters.

Example 6: Consider the transducer in Figure 2 for the pattern $\langle$`Peak`$, 1\rangle$ and the signature S = '=>=<<=<>>=<=====>'. The transitions are as follows:

$$s \xrightarrow{=\,:\,\mathbf{out}} s \xrightarrow{>\,:\,\mathbf{out}} s \xrightarrow{=\,:\,\mathbf{out}} s \xrightarrow{<\,:\,\mathbf{out}} r \xrightarrow{<\,:\,\mathbf{maybe}_{\text{b}}} r$$
$$\xrightarrow{=\,:\,\mathbf{maybe}_{\text{b}}} r \xrightarrow{<\,:\,\mathbf{maybe}_{\text{b}}} r \xrightarrow{>\,:\,\mathbf{found}} t \xrightarrow{>\,:\,\mathbf{in}} t$$
$$\xrightarrow{=\,:\,\mathbf{maybe}_{\text{a}}} t \xrightarrow{<\,:\,\mathbf{out}_{\text{a}}} r \xrightarrow{=\,:\,\mathbf{maybe}_{\text{b}}} r \xrightarrow{=\,:\,\mathbf{maybe}_{\text{b}}} r$$
$$\xrightarrow{=\,:\,\mathbf{maybe}_{\text{b}}} r \xrightarrow{=\,:\,\mathbf{maybe}_{\text{b}}} r \xrightarrow{=\,:\,\mathbf{maybe}_{\text{b}}} r \xrightarrow{>\,:\,\mathbf{found}} t$$

The two $\mathbf{found}$ correspond to two $\langle$`Peak`$, 1\rangle$-occurrences: the first one corresponds to the word from the first $\mathbf{maybe}_{\text{b}}$ to the first $\mathbf{in}$ (i.e., the word $\mathbf{maybe}_{\text{b}}^3$ $\mathbf{found}$ $\mathbf{in}$); the second one corresponds to the word from just after the last $\mathbf{out}_{\text{a}}$ to the last $\mathbf{found}$ (i.e., the word $\mathbf{maybe}_{\text{b}}^5$ $\mathbf{found}$). □

We refer to seed transducers simply as transducers. In [3], conditions were given so as to define a correct transducer:

Definition 7 (transducer wellformedness): A transducer is *well-formed* with respect to a pattern π if the following conditions hold: 1) its output language is a subset of the language accepted by the automaton in Figure 5, 2) all π-occurrences in a signature produce maximal words matching the regular expression $\mathbf{maybe}_{\text{b}}^*(\mathbf{found}_{\text{e}} \mid \mathbf{found}(\mathbf{maybe}_{\text{a}}^*\mathbf{in})^*)$. □

Example 7: The transducer in Figure 2 for the pattern $\langle$`Peak`$, 1\rangle$ is well-formed since its output language is a subset of the language accepted by the automaton in Figure 5, and all $\langle$`Peak`$, 1\rangle$-occurrences produce maximal words matching the regular expression $\mathbf{maybe}_{\text{b}}^*\mathbf{found}(\mathbf{maybe}_{\text{a}}^*\mathbf{in})^*$. □

IV. Automatically Generating a Transducer

To automatically generate the minimum-state transducer from a given recognisable pattern $\pi = \langle \sigma, \beta \rangle$, we first generate a DFA accepting the language $\mathcal{L}(\sigma)$, which can be obtained using standard techniques [4]. We then describe in Section IV-A how to transform this DFA, if need be, into an equivalent non-minimal one that satisfies some additional

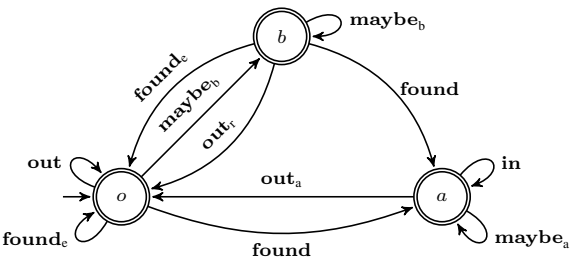

Fig. 5: Automaton accepting the output language of a well-formed transducer (taken with permission from [3]).

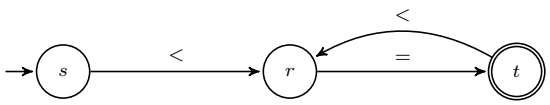

Fig. 6: The minimum-state automaton accepting $\mathcal{L}(\text{`(<=)}^{+}\text{'})$.

necessary conditions so that we can correctly transform it into a transducer for π. Finally, we give an algorithm in Section IV-B for transforming such a DFA into a transducer that produces the phases of detecting all π-occurrences within a signature.

A. Expanding the Automaton

We describe two potential problems for our transducer generator when starting from an automaton $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, Q_\text{a} \rangle$ accepting the language $\mathcal{L}(\sigma)$ and we give algorithms to solve these problems by duplicating states and adding the corresponding transitions to $\mathcal{A}$. Note that the minimum-state automaton accepting $\mathcal{L}(\sigma)$ is enough in most cases.

1) Disjoint sets of states: Consider a recognisable pattern $\pi = \langle \sigma, \beta \rangle$. From [3] we know that the transduction of a single word in $\mathcal{L}(\sigma)$ must match the regular expression $\mathbf{out}^\beta \mathbf{maybe}_\text{b}^*(\mathbf{found}_\text{e} \mid \mathbf{found}(\mathbf{maybe}_\text{a}^*\mathbf{in})^*)$. Recall that a π-occurrence in a signature S is a σ-occurrence in S except for its first β symbols, and that the transduction of a π-occurrence must match the regular expression $\mathbf{maybe}_\text{b}^*(\mathbf{found}_\text{e} \mid \mathbf{found}(\mathbf{maybe}_\text{a}^*\mathbf{in})^*)$.

In order to properly transduce a word in $\mathcal{L}(\sigma)$, we need to know at every state if an accepting state has already been reached and can be reached again, even if we are not at an accepting state at the moment. This is the case of the words that transduce into $\mathbf{out}^\beta \mathbf{maybe}_\text{b}^* \mathbf{found}(\mathbf{maybe}_\text{a}^*\mathbf{in})^*$.

Example 8: Consider the regular expression `IncreasingStairs` = $\text{`(<=)}^{+}\text{'}$. The minimal automaton accepting $\mathcal{L}(\text{`(<=)}^{+}\text{'})$ is in Figure 6. Note that, after consuming a word w in $\mathcal{L}(\text{`(<=)}^{+}\text{'})$, we reach the accepting state $t$. Moreover, note that after extending the word $w$ to the word $w_1 = w\text{`<'}$ we reach the same state as after consuming the word $w_2 =$ `<`, namely the state r. From a language membership point of view, w_1 and w_2 are equivalent. Nevertheless, the last symbol of w_1 must be transduced into $\mathbf{maybe}_\text{a}$, while the last (and here only) symbol of w_2 must be transduced into $\mathbf{out}$ if $\beta = 1$ or into $\mathbf{maybe}_\text{b}$ if $\beta = 0$.

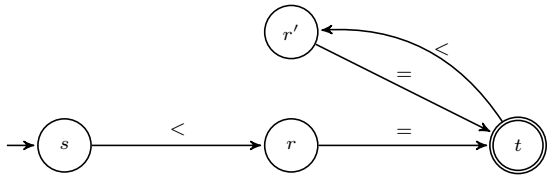

Fig. 7: Expanded automaton accepting $\mathcal{L}(\text{`(<=)}^{+}\text{'})$.

Therefore, w_1 and w_2 are not equivalent from a transduction point of view. Hence we must know if a word in $\mathcal{L}(\text{`(<=)}^{+}\text{'})$ has been consumed and is currently being extended. □

To solve this issue, after reaching an accepting state, there should be no transitions leading back to states on a path from the initial state to an accepting state: it should be made possible to partition the set of states Q of an automaton into the set Q^- of states reachable from the initial state strictly before reaching an accepting state, and the set Q_a^+ of states that can be reached from the accepting states, including the set Q_a of accepting states. That is, we require $Q^- \cap Q_\text{a}^+ = \emptyset$. Neither Q^- nor Q_a^+ are empty because $q_0 \in Q^-$, $\emptyset \subset Q_\text{a} \subseteq Q_\text{a}^+$, and by Definition 2 $q_0 \notin Q_\text{a}$ because the empty word is not in $\mathcal{L}(\sigma)$.

Example 9: Consider again the regular expression `IncreasingStairs` = $\text{`(<=)}^{+}\text{'}$ of Example 8. It is not possible to partition the states of the minimum-state automaton $\mathcal{A}$ in Figure 6 into disjoint sets Q^- and Q_a^+ because state r is reachable both from the initial state s and the accepting state t. To be able to partition the states, it suffices to expand $\mathcal{A}$ by creating a new state r', redirecting the transition from t to r so that it goes from t to r', and copying the only outgoing transition of r, that is adding a transition from r' to t consuming the symbol '='. The resulting automaton is in Figure 7. Now $Q^- = \{s, r\}$ and $Q_\text{a}^+ = \{t, r'\}$. □

It is possible to calculate both Q^- and Q_a^+ using depth-first search, so as to check whether they are disjoint. Algorithm 1 expands, if need be, an automaton $\mathcal{A}$ following the intuition in Example 9. The semantics of Q_open is the set of states that belong to Q_a^+ but could have transitions to states in Q^-. All states created by Algorithm 1 are non-accepting by definition of Q^- and Q_a^+. Note that if Q^- and Q_a^+ are disjoint, the Algorithm 1 actually does not modify the automaton.

Theorem 1: The expansion of an automaton $\mathcal{A}$ achieved with Algorithm 1 preserves the language of $\mathcal{A}$.

Proof: Without loss of generality, consider a minimal automaton $\mathcal{A}$ and its corresponding expanded automaton $\mathcal{A}'$, such that a state q has been duplicated creating the state q'. We show that q and q' are not differentiable, that is, any automaton minimisation algorithm would merge them. Consider the partition of the states of $\mathcal{A}'$ such that q and q' are in one set and all other states are in one-element sets. For every transition t leaving q, such that t is not a self-loop, by construction q' has a corresponding transition consuming the same symbol and reaching the same state. For every self-loop on q, there is a corresponding self-loop consuming the same symbol on q'. In consequence, states q and q' are not differentiable and any automaton minimisation algorithm would merge them.

Algorithm 1: Ensure disjoint sets of states

Data: A DFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, Q_\text{a} \rangle$
use depth-first search to calculate Q^-
initialise the sets $Q_\text{open} \coloneqq Q_\text{a}$ and $Q_\text{a}^+ \coloneqq \emptyset$
while Q_open is not empty **do**
  move a state q from Q_open to Q_a^+
  foreach transition $t = q \xrightarrow{a} v$ in δ **do**
    if $v \in Q^-$ **then**
      create a state v' and add it to Q_open
      replace t with $q \xrightarrow{a} v'$
      foreach transition $v \xrightarrow{b} x$ in δ **do**
        if it is not a self-loop **then**
          create a transition $v' \xrightarrow{b} x$
        else create a self-loop $v' \xrightarrow{b} v'$
    else
      if $v \notin Q_\text{a}^+$ **then** add v to Q_open

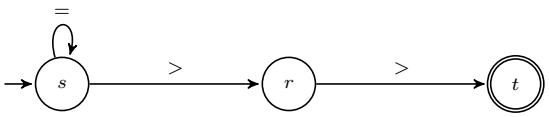

Fig. 8: The minimum-state automaton accepting $\mathcal{L}(\text{`=}^*\text{>>'})$.

Therefore, when minimising $\mathcal{A}'$ we obtain $\mathcal{A}$. Hence, the language remains unchanged. □

2) Enough ignore count transitions: Consider the recognisable pattern $\pi = \langle \sigma, \beta \rangle$ where the regular expression σ is '=*>>'. The unique shortest word in the language $\mathcal{L}(\sigma)$ is of length 2, namely '>>'. The mismatch overlap of $\mathcal{L}(\sigma)$ is 1. Hence, by Definition 6, we must have $\beta = 1$. The minimum-state automaton accepting $\mathcal{L}(\sigma)$ is in Figure 8. Note that it is not possible to set the output symbols of the transitions in a way that the $\beta = 1$ first symbols are transduced into **out**. For example, if we set the output symbol of the self-loop to **out**, then all the '=' will be transduced into **out**.

Given a recognisable pattern $\langle \sigma, \beta \rangle$, it is always possible to expand the automaton accepting σ so that all transitions in all paths from the initial state of length at least the ignore count β can be used only once because, by Definition 2, β must be strictly smaller than the length of the shortest words in $\mathcal{L}(\sigma)$, that is, β must be smaller than the length of the shortest path from the initial state to an accepting state.

Example 10: Consider the recognisable pattern $\langle \text{`=}^*\text{>>'}, 1 \rangle$. The minimum-state automaton of Figure 8 accepting the language $\mathcal{L}(\text{`=}^*\text{>>'})$ can be expanded by creating a new state s' and unfolding the loop in state s so it is possible to always know which one was the first symbol consumed. The resulting automaton is in Figure 9. Note that both transitions leaving the initial state s can be used at most once and only one of them can be used on any path to the accepting state t. □

To expand an automaton $\mathcal{A}$ following the intuition in Example 10, we use Algorithm 2. Note that all newly created

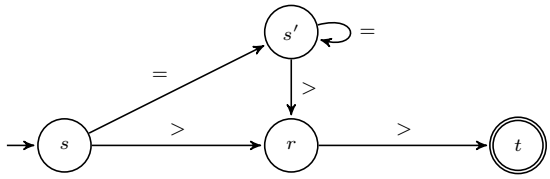

Fig. 9: Expanded automaton accepting $\mathcal{L}(\text{`=}^*\text{>>'})$.

Algorithm 2: Ensure enough ignore count transitions

Data: A DFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, Q_\text{a} \rangle$ and ignore count β
initialise the sets $Q_\text{open} \coloneqq \{q_0\}$ and $Q_\beta^- \coloneqq \emptyset$
for $i \coloneqq \beta$ **downto** 1 **do** // invariant: $Q_\text{open} \cap Q_\text{a} = \emptyset$
  move a state q from Q_open to Q_β^-
  foreach transition $t = q \xrightarrow{a} v$ in δ **do**
    if $v \in Q_\beta^-$ **then** // expand node
      create a state v' and add it to Q_open
      replace t with $v \xrightarrow{a} v'$
      foreach transition $v \xrightarrow{b} x$ in δ **do**
        if $v \neq x$ **then** create a transition $v' \xrightarrow{b} x$
        else create a self-loop $v' \xrightarrow{b} v'$
    else if $v \notin Q_\text{open}$ **then** add v to Q_open
while Q_open is not empty **do**
  move a state q from Q_open to Q_closed
  foreach transition $q \xrightarrow{a} v$ in δ **do**
    if $v \in Q_\beta^-$ **then** ...// copy of expand node above
    else if $v \notin Q_\text{closed}$ **then** add v to Q_open

states are by definition non-accepting because the ignore count is strictly smaller than the length of a minimal-length word in the language accepted by $\mathcal{A}$. The semantics of Q_β^- is the set of states in all paths from the initial state q_0 of length the ignore count β such that we know there are no cycles between them. The semantics of Q_open is the set of states in Q^- such that we do not know if there are any cycles back to states in Q_β^-. The semantics of Q_closed is the set of states in Q^- such that we already know that they do not have outgoing transitions leading back to states in Q_β^-. At the end of the algorithm we know that $Q_\beta^- \cup Q_\text{closed} = Q^-$. Note that Algorithm 2 only modifies the automaton when needed.

Theorem 2: The expansion of an automaton $\mathcal{A}$ achieved with Algorithm 2 preserves the language of $\mathcal{A}$.

Proof: This proof is similar to that of Theorem 1 and is omitted for space reasons. □

B. Adding the Output Symbols

The concatenation $L_1 L_2$ of two languages L_1 and L_2 is the language of all words $w_1 w_2$ where $w_1 \in L_1$ and $w_2 \in L_2$.

For any recognisable pattern $\pi = \langle \sigma, \beta \rangle$, a transducer can be generated by means of the following algorithm:

1) Build a deterministic finite automaton $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, Q_\text{a} \rangle$ that accepts the language $\mathcal{L}(\sigma)$.

2) Expand $\mathcal{A}$ using Algorithm 1 and Algorithm 2.
3) Divide the states of $\mathcal{A}$ into three disjoint sets:
   - The set Q_{a} of accepting states.
   - The set Q^- of states reachable from the initial state q_0 without passing through an accepting state. Recall that the empty word cannot be in $\mathcal{L}(\sigma)$, hence $q_0 \in Q^-$.
   - The set Q^+ of all states that can be reached from an accepting state, excluding the accepting states. After applying Algorithm 1 we know that $Q^+ = Q \setminus Q_{\text{a}} \setminus Q^-$. Note that it is possible for Q^+ to be empty.
4) Set the output symbol of every transition in $\mathcal{A}$ as follows:
   a) For each accepting state q: if q has no outgoing transitions, then set the output symbol of all transitions from states in Q^- to q to $\mathbf{found}_{\text{end}}$.
   b) For each accepting state q: if q has outgoing transitions, then set the output symbol of all transitions from states in Q^- to q to $\mathbf{found}$.
   c) Set the output symbol of all transitions from states in $Q_{\text{a}} \cup Q^+$ to states in Q_{a} to $\mathbf{in}$.
   d) Set the output symbol of all transitions from states in $Q_{\text{a}} \cup Q^+$ to states in Q^+ to $\mathbf{maybe}_{\text{after}}$.
   e) For every transition on a path of length β leaving the initial state: set the output symbol to $\mathbf{out}$. This will not overwrite the symbols set in Steps 4a–4d, because of the expansion performed in Step 2 by Algorithm 2.
   f) For all remaining transitions in $\mathcal{A}$: set the output symbol to $\mathbf{maybe}_{\text{before}}$.
5) Using standard transducer concatenation techniques [6], concatenate $\mathcal{A}$ to a transducer for Σ^* where all the output labels are set to $\mathbf{out}$. This creates a non-deterministic finite transducer for $\Sigma^*\mathcal{L}(\sigma)$. Given a language $\mathcal{L}$, constructing an automaton recognising $\Sigma^*\mathcal{L}$ is a common string searching technique [7].
6) Determinise the transducer for $\Sigma^*\mathcal{L}(\sigma)$ using the disambiguation algorithm in Section IV-C.
7) Replace output symbols so that the transducer is well-formed using the following rules:
   a) For every transition t from a state q in $Q_{\text{a}} \cup Q^+$ to a state in Q^-: if q has incoming transitions with the output symbols $\mathbf{in}$ or $\mathbf{maybe}_{\text{after}}$, then:
      - If the output symbol of t is $\mathbf{out}$, then replace it with $\mathbf{out}_{\text{after}}$.
      - If the output symbol of t is $\mathbf{maybe}_{\text{before}}$, then replace it with $\mathbf{out}_{\text{after}}$ $\mathbf{maybe}_{\text{before}}$.
   b) For every transition from a state in $Q_{\text{a}} \cup Q^+$ with the output symbol $\mathbf{found}$: replace the output symbol by $\mathbf{out}_{\text{after}}$ $\mathbf{found}$.
   c) For every transition t from a state q in Q^- with the output symbol $\mathbf{out}$: if q has incoming transitions with the output symbol $\mathbf{maybe}_{\text{before}}$, then replace the output symbol of t by $\mathbf{out}_{\text{reset}}$. If t is a self-loop, then expand the transducer by following Algorithm 3.
8) Mark all states of the transducer as accepting.
9) Minimise the transducer using standard transducer minimisation techniques [8].

Algorithm 3: Ensure $\mathbf{out}_{\text{r}}$ is not in a self-loop

Data: A transducer $\mathcal{T}$ and a transition $t = q \xrightarrow{a\,:\,\mathbf{out}_{\text{r}}} q$
replace t with $q \xrightarrow{a\,:\,\mathbf{out}_{\text{r}}} q'$ for a new state q'
create a self-loop $q' \xrightarrow{a\,:\,\mathbf{out}} q'$
foreach transition $q \xrightarrow{b\,:c} v$ in $\mathcal{T}$ **do**
  create a transition $q' \xrightarrow{b\,:c} v$

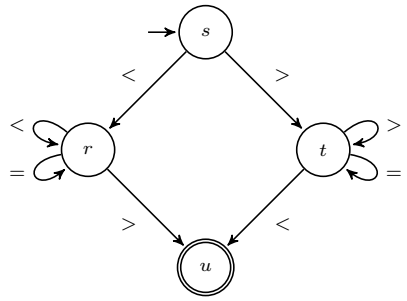

Fig. 10: The minimal automaton recognising $\mathcal{L}(\texttt{Inflexion})$.

Example 11: Consider again the recognisable pattern $\pi = \langle \texttt{Inflexion}, 1 \rangle$. The minimum-state automaton $\mathcal{A}$ accepting the language $\mathcal{L}(\texttt{Inflexion})$ is in Figure 10. The expansion algorithms in Step 2 do not modify $\mathcal{A}$. The set of states $Q = \{s, r, t, u\}$ of $\mathcal{A}$ is divided into the subsets $Q_{\text{a}} = \{u\}$, $Q^- = \{s, r, t\}$, and $Q^+ = \emptyset$. We then proceed to set the output symbols of the transitions in order to build the transducer for π. After Step 4b, we have the transducer in Figure 11. After Step 4f, we have the transducer in Figure 12. After concatenating the transducer for Σ^* in Step 5 we have the transducer in Figure 13. After determinising the transducer in Step 6, we have the transducer in Figure 14. Applying Step 7 does not change the transducer in Figure 14. Once all the states are made accepting and the obtained transducer is minimised, we have the minimal transducer in Figure 15, which is equal to transducer for π in [9]. □

C. Disambiguation Algorithm

Typically, concatenating two transducers results in a transducer that is ambiguous [6]: it is both non-deterministic, and non-functional, so it can produce multiple output sequences for the same input sequence.

Example 12: Consider the pattern $\langle \texttt{Inflexion}, 1 \rangle$ and the signature $S =$ '=<<=<>' of length 6. The transducer in Figure 13, obtained after Step 5 of the algorithm in Section IV-B, can transduce S into four different sequences: $\mathbf{out}^5\mathbf{found}_{\text{end}}$, $\mathbf{out}^4\mathbf{maybe}_{\text{before}}\mathbf{found}_{\text{end}}$, $\mathbf{out}^3\mathbf{maybe}^2_{\text{before}}\mathbf{found}_{\text{end}}$, and $\mathbf{out}^2\mathbf{maybe}^3_{\text{before}}\mathbf{found}_{\text{end}}$. Note that the lengths of all possible transductions of S are the length of S, namely 6. □

Disambiguating a non-functional transducer in a way that preserves the best output symbols is not a trivial task [10], and this cannot be handled directly by known algorithms like transducer determinisation [11] and functional trans-

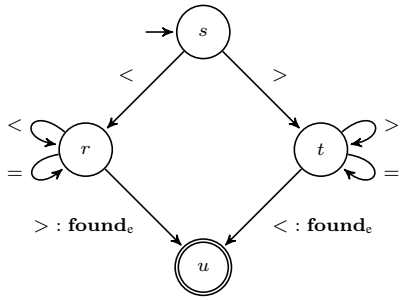

Fig. 11: Intermediate transducer for $\langle \texttt{Inflexion}, 1\rangle$ after applying Step 4b of the algorithm in Section IV-B.

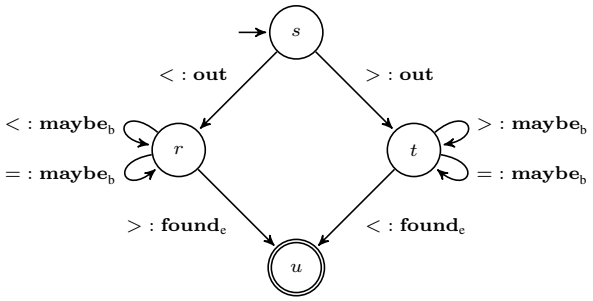

Fig. 12: Intermediate transducer for $\langle \texttt{Inflexion}, 1\rangle$ after applying Step 4f of the algorithm in Section IV-B.

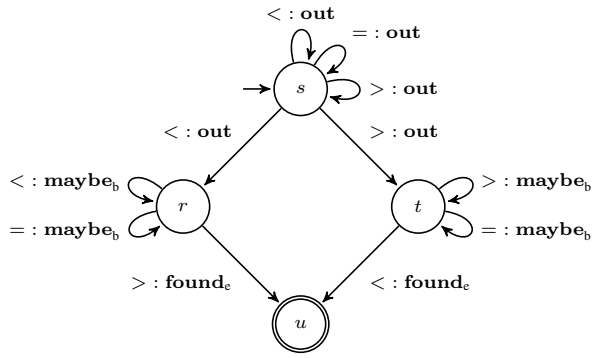

Fig. 13: Nondeterministic transducer for $\langle \texttt{Inflexion}, 1\rangle$ after applying Step 5 of the algorithm in Section IV-B.

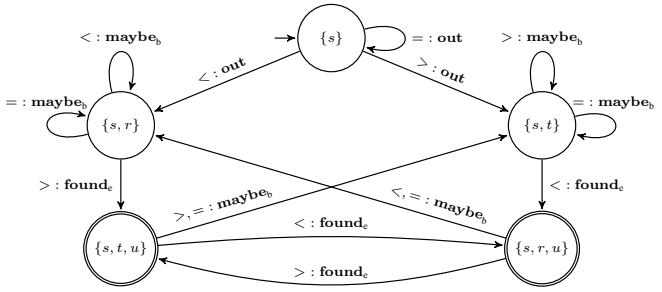

Fig. 14: A non-minimal transducer for $\langle \texttt{Inflexion}, 1\rangle$.

ducer disambiguation [12]. In our particular case, we want to transform the transducer obtained after applying Step 5 into a deterministic transducer that transduces all pattern occurrences in a signature. The best output symbols are then those guaranteeing that pattern occurrences are transduced into maximal words matching the regular expression $\mathbf{maybe}^*_{\text{before}}(\mathbf{found}_{\text{end}} \mid \mathbf{found}(\mathbf{maybe}^*_{\text{after}}\mathbf{in})^*)$.

Example 13: Consider again the recognisable pattern $\langle \texttt{Inflexion}, 1\rangle$ and the signature $S =$ '=<<=<>' from Example 12. We want to transform the non-deterministic transducer in Figure 13 into a deterministic transducer that transduces S into the sequence $\mathbf{out}^2\mathbf{maybe}^3_{\text{before}}\mathbf{found}_{\text{end}}$. □

The symbols of the semantic alphabet have the total order $\mathbf{out} \prec \mathbf{maybe}_{\text{b}} \prec \mathbf{found} \prec \mathbf{found}_{\text{e}} \prec \mathbf{maybe}_{\text{a}} \prec \mathbf{in}$.

Consider a recognisable pattern $\pi = \langle \sigma, \beta\rangle$ and a signature S having a suffix matching σ. In order to guarantee the whole π-occurrence corresponding to the longest suffix of S matching σ is transduced into $\mathbf{maybe}^*_{\text{before}}(\mathbf{found}_{\text{end}} \mid \mathbf{found}(\mathbf{maybe}^*_{\text{after}}\mathbf{in})^*)$, and not just any suffix matching σ after skipping the first β symbols, the best output sequence for S corresponds to choosing the transition with the largest output symbol. The intuition is that, the larger the symbol in the order, the farther we are inside an occurrence.

Example 14: Consider again Example 13. Out of the four possible transductions of the signature $S =$ '=<<=<>' made by the transducer in Figure 13, the transduction $\mathbf{out}^2\mathbf{maybe}^3_{\text{before}}\mathbf{found}_{\text{end}}$ corresponds to always choosing the largest output symbol among all the possible ones. □

Our disambiguation algorithm is similar to the powerset construction [13] used for the determinisation of automata. In the powerset construction, the states of the DFA are sets of states of the non-deterministic automaton (NFA). Given an NFA $\mathcal{A} = \langle Q, \Gamma, \delta, q_0, Q_{\text{a}}\rangle$, the equivalent DFA has states corresponding to subsets of Q. The initial state of the DFA is $\{q_0\}$, the (one-element) set of initial states. The transition function of the DFA maps a state q, which is a subset of Q, and an input symbol $a \in \Gamma$ to the set $\delta'(q, a) = \{\delta(r, a) \mid r \in q\}$, the set of all states that can be reached by a transition from a state in q consuming a. A state q of the DFA is an accepting state if and only if at least one member of q is an accepting state of the NFA. In the transducer case, we redefine the transition function so that for any transition between two states of the deterministic transducer, we keep only the maximum output symbol among all the possible ones, that is, $\delta'(q, a) = \langle\{u \mid \langle u, *\rangle \in T\},\ (\max(b) \mid \langle *, b\rangle \in T)\rangle$, where $T = \{\delta(r, a) \mid r \in q\}$.

Example 15: Consider again the non-deterministic transducer $\mathcal{T}$ in Figure 13. From state s with input symbol '<' it is possible to reach both s, with output symbol $\mathbf{out}$, and r, also with output symbol $\mathbf{out}$. So, there exists a transition in the deterministic transducer $\mathcal{T}'$ from state $\{s\}$ to state $\{s, r\}$ with input symbol '<' and output symbol $\mathbf{out}$. From states s and r in $\mathcal{T}$ with input symbol '<', it is possible to reach again both s and r, but there is a self-loop on r with input symbol '<' and output symbol $\mathbf{maybe}_{\text{before}}$, as well as a self-loop on s with input symbol '<' and output symbol $\mathbf{out}$. In this case,

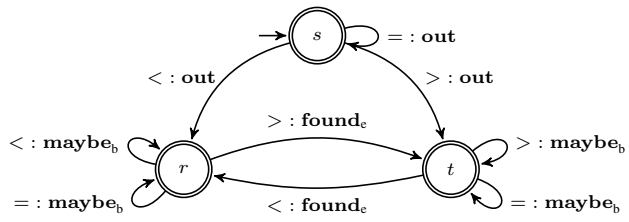

Fig. 15: Minimal transducer for $\langle \texttt{Inflexion}, 1 \rangle$.

we keep the maximum symbol between $\mathbf{out}$ and $\mathbf{maybe}_{\text{before}}$, that is we keep the $\mathbf{maybe}_{\text{before}}$ and discard the $\mathbf{out}$. So, there exists a self-loop in $\mathcal{T}'$ on state $\{s, r\}$ with input symbol '<' and output symbol $\mathbf{maybe}_{\text{before}}$. Repeating this process for all input symbols in Γ and all sets of states of $\mathcal{T}$ results in the deterministic transducer $\mathcal{T}'$ in Figure 14. □

We now show that the resulting transducer does not only transduce the last occurrence of a pattern, but all of them.

Theorem 3: Given a pattern $\pi = \langle \sigma, \beta \rangle$, a transducer $\mathcal{T}_1$ accepting Σ^* and producing words matching $\mathbf{out}^*$, and a transducer $\mathcal{T}_2$ accepting $\mathcal{L}(\sigma)$ and producing words matching $\gamma = \mathbf{out}^{\beta}\mathbf{maybe}^*_{\text{before}}(\mathbf{found}_{\text{end}} \mid \mathbf{found}(\mathbf{maybe}^*_{\text{after}}\mathbf{in})^*)$, the transducer obtained when applying our disambiguation algorithm to the transducer $\mathcal{T}_1\mathcal{T}_2$ transduces all π-occurrences.

Proof: By the powerset construction, we know that the resulting transducer $\mathcal{T}$ after applying our disambiguation algorithm is deterministic. Consider a signature S_1 that is a word of size m matching σ. S_1 is transduced into T_1. We know there is at least one non-deterministic path in $\mathcal{T}_1\mathcal{T}_2$ such that the π-occurrence in S_1 is transduced into a sequence T_1 matching the regular expression $\mathbf{maybe}^*_{\text{b}}(\mathbf{found}_{\text{e}} \mid \mathbf{found}(\mathbf{maybe}^*_{\text{a}}\mathbf{in})^*)$. Such a path corresponds to only using the deterministic transitions in $\mathcal{T}_2$. Choosing any other accepting path, if possible, would lead to a transduction with a longer prefix of $\mathbf{out}$ symbols. Therefore, when given a choice between different possible output symbols, choosing the largest symbol in the order guarantees that a longer suffix matches γ. Consider a signature S_2 that is also a word matching σ. S_2 is transduced into T_2. Assume that the signature S_1S_2 contains two disjoint maximal words matching σ. Given that $\mathcal{T}$ is deterministic, we know that the prefixes of the output sequences do not change. So, when transducing S_1S_2, the prefix S_1 is transduced into T_1. Given that $\mathcal{T}$, by construction, transduces the longest possible suffix of S_1S_2 matching σ into a word matching γ, $\mathcal{T}$ transduces S_1S_2 into T_1T_2. The reasoning for other signatures where there are symbols that do not belong to any π-occurrence is similar and has been omitted for space reasons. Hence, $\mathcal{T}$ transduces all π-occurrences. □

V. Conclusion

This work contributes, in the context of time-series constraints, to the systematic reconstruction of the Global Constraint Catalogue that we have previously advocated [14]. Our setting covers a large class of useful constraints, whether they are listed in the Time-Series Constraint Catalogue [9] or not. Our approach differs from existing ones, which design dedicated propagators [15], [16] and reformulations [17], [18] for specific constraints.

We have formalised the class of time-series patterns whose transducers can be handled by the propagator and checker synthesis in [3] and described a fully automated parametric tool that generates, in an off-line process, a transducer from such a pattern. Our tool was implemented in SICStus Prolog 4.2 [19] and is available at http://www.it.uu.se/research/group/astra/software/transducer-generator.zip.

Our tool generates the handcrafted transducers of [3]. By proving that our tool only generates well-formed transducers, we also proved that the transducers in [3] are well-formed.

In the **future** we will extend the semantic alphabet, and in turn our algorithm, to cover a broader range of patterns.

Acknowledgements: We thank E. Arafailova, N. Beldiceanu, and the anonymous referees for their helpful comments. The authors are supported by grants 2012-4908 and 2015-0491 of the Swedish Research Council (VR).

References

[1] N. Beldiceanu, G. Ifrim, A. Lenoir, and H. Simonis, "Describing and generating solutions for the EDF unit commitment problem with the ModelSeeker," in *CP 2013*, ser. LNCS, vol. 8124. Springer, 2013.

[2] E. Arafailova, N. Beldiceanu, R. Douence, P. Flener, M. A. Francisco Rodríguez, J. Pearson, and H. Simonis, "Time-series constraints: Improvements and application in CP and MIP contexts," in *CP-AI-OR 2016*, ser. LNCS, vol. 9676. Springer, 2016, pp. 18–34.

[3] N. Beldiceanu, M. Carlsson, R. Douence, and H. Simonis, "Using finite transducers for describing and synthesising structural time-series constraints," *Constraints*, vol. 21, no. 1, pp. 22–40, January 2016.

[4] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Addison-Wesley, 2007.

[5] J. Sakarovitch, *Elements of Language Theory*. Cambridge University Press, 2009.

[6] M. Mohri, "Weighted automata algorithms," in *Handbook of Weighted Automata*. Springer, 2009, pp. 213–254.

[7] ——, "String-matching with automata," *Nordic Journal of Computing*, vol. 4, no. 2, pp. 217–231, 1997.

[8] C. Choffrut, "Minimizing subsequential transducers: a survey," *Theoretical Computer Science*, vol. 292, no. 1, pp. 131–143, 2003.

[9] E. Arafailova, N. Beldiceanu, R. Douence, M. Carlsson, P. Flener, M. A. Francisco Rodríguez, J. Pearson, and H. Simonis, "Global Constraint Catalog, Volume II, Time-Series Constraints," *arXiv:1609.08925*, 2016.

[10] G. Iglesias, A. de Gispert, and B. Byrne, "Transducer disambiguation with sparse topological features," in *EMNLP 2015*, 2015, pp. 2275–2280.

[11] M. Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 2, pp. 269–311, 1997.

[12] ——, "A disambiguation algorithm for finite automata and functional transducers," in *CIAA 2012*, ser. LNCS, vol. 7381. Springer, 2012.

[13] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM J. Res. Dev.*, vol. 3, no. 2, pp. 114–125, 1959.

[14] N. Beldiceanu, P. Flener, J.-N. Monette, J. Pearson, and H. Simonis, "Toward sustainable development in constraint programming," *Constraints*, vol. 19, no. 2, pp. 139–149, 2014.

[15] J.-C. Régin, "A filtering algorithm for constraints of difference in CSPs," in *AAAI 1994*. AAAI Press, 1994, pp. 362–367.

[16] N. Beldiceanu and M. Carlsson, "Sweep as a generic pruning technique applied to the non-overlapping rectangles constraints," in *CP 2001*, ser. LNCS, vol. 2239. Springer, 2001, pp. 377–391.

[17] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, C.-G. Quimper, and T. Walsh, "Reformulating global constraints: The *slide* and *regular* constraints," in *SARA 2007*, ser. LNAI, vol. 4612. Springer, 2007.

[18] C. Bessière, G. Katsirelos, N. Narodytska, C.-G. Quimper, and T. Walsh, "Decomposition of the NValue constraint," in *CP 2010*, ser. LNCS, D. Cohen, Ed., vol. 6308. Springer, 2010, pp. 114–128.

[19] M. Carlsson, G. Ottosson, and B. Carlson, "An open-ended finite domain constraint solver," in *PLILP 1997*, ser. LNCS, vol. 1292. Springer, 1997.