A Transducer-Based Model for Representing Functional Constraints on Integer Sequences

Ekaterina Arafailova[1], Nicolas Beldiceanu[1], Mats Carlsson[2], Rémi Douence[3], María Andreína Francisco Rodríguez[5], Helmut Simonis[6]

[1] TASC (LS2N), IMT Atlantique, FR – 44307 Nantes, France
FirstName.LastName@imt-atlantique.fr
[2] SICS, P.O. Box 1263, SE – 164 29 Kista, Sweden
Mats.Carlsson@sics.se
[3] GALINETTE (LS2N), IMT Atlantique, FR – 44307 Nantes, France
Remi.Douence@imt-atlantique.fr
[4] School of Computing, National University of Singapore, 117747, Singapore
andreina@comp.nus.edu.sg
[5] Insight Centre for Data Analytics, University College Cork, Ireland
helmut.simonis@insight-centre.org

Abstract. We extend a transducer-based computational model, originally introduced to express functions over time series concisely, to functions over integer sequences that arise in the context of constraint programming. A transducer now only depends on the regular expression whose maximal occurrences it has to find, and not any more on any quantitative aspect. Also, the parameters that express the quantitative aspect allow one to associate many more functions to the same transducer. The new model preserves the main advantages of the previous model, namely that transducers are still independent of the sequence size and still take time linear in the sequence size for evaluating the corresponding functions. Finally, we show how to generate automatically such a transducer from the regular expression.

1 Introduction

Motivated by representing a large number of sequence constraints, such as [7,13,14], we extend an initial work [5,2] that uses a manually designed transducer [16] on the way to automatically inducing a decomposition of a time-series constraint. Our aim is a concise normalised representation that is expressive enough to capture declaratively many sequence constraints, namely those constraining an aggregation of integer features of all maximal occurrences of a regular expression within an integer sequence, such as the minimal width of all its plateaus.

Our contribution is a regular-expression-based representation for such constraints. From such a representation, we automatically generate a transducer. We not only extend the set of time-series constraints of [5,2], but also cover most sequence constraints of the Global Constraint Catalogue [6], such as [7,13,14].

From a transducer, a register automaton describing the computation of the function can be synthesised [5]. From a register automaton, a decomposition of the represented constraint in terms of basic constraints can be induced [4].

Section 2 provides some background. Section 3 defines our regular-expression-based representation of the considered sequence constraints. Section 4 gives an operational view of such constraints, using for the regular expression a transducer whose output alphabet is a set of instructions denoting the computations for each phase of recognising all maximal occurrences of the regular expression within a sequence. The instructions use registers for recording information about past maximal occurrences of the regular expression, the current possibly unfinished maximal occurrence, and the hypothetical next maximal occurrence. Section 5 shows how to generate automatically such a transducer from a regular expression. Finally, Section 6 summarises our contributions and concludes.

2 Background

We assume the reader is familiar with regular expressions and automata [12]. The empty word is denoted by ε. The concatenation of two words w_1 and w_2 is denoted by $w_1 w_2$. A word w is a *factor* of a word x if there exist two words v and u such that $x = vwu$; if either $v \neq \varepsilon$ or $u \neq \varepsilon$, then w is a *proper factor* of x; if $v = \varepsilon$, then w is a *prefix* of x; if $u = \varepsilon$, then w is a *suffix* of x. 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 a regular expression σ, let $\mathcal{L}_\sigma$ denote its regular language, let $\overrightarrow{\sigma}$ denote the set of all prefixes of all words in $\mathcal{L}_\sigma$, let $\overleftarrow{\sigma}$ denote the set of all suffixes of all words in $\mathcal{L}_\sigma$, and let $\overleftrightarrow{\sigma}$ denote the set of all factors of all words in $\mathcal{L}_\sigma$.

Definition 1 (regular-expression overlap [11]). *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 *and denoted by o_σ. If the word overlap is never defined for any pair of words in $\mathcal{L}_\sigma$, then the σ-overlap is* 0.

For example, for the $\sigma_1 =$ ‘ $>>^*$ ’ regular expression, the value of o_{σ_1} is 0 since the maximum word overlap is never defined for any pair of words in $\mathcal{L}_{\sigma_1}$.

Definition 2 (mismatch overlap [11]). *Given a regular expression σ over an alphabet Σ, a word $w \in \overrightarrow{\sigma} \setminus \mathcal{L}_\sigma$, and a symbol $z \in \Sigma$, if $wz \notin \overrightarrow{\sigma}$, then the length of the longest y, such that y is a suffix of wz and $y \in \overrightarrow{\sigma}$, is called the* mismatch overlap *of w and z. The maximum mismatch overlap of all words in $\overrightarrow{\sigma} \setminus \mathcal{L}_\sigma$ and all symbols in Σ is called the* mismatch overlap *of σ and denoted by μ_σ.*

For example, for the $\sigma_2 =$ ‘ $<^+=<^+=>$ ’ regular expression, the value of μ_{σ_2} is infinite since for any word w in the language of ‘ $<^+=<^+ =$’ and the symbol $z =$ ‘ $<$ ’, the mismatch overlap is one plus the length of the longest suffix of w that is in $\mathcal{L}_{<^+=}$. This value is not bounded.

3 Defining Functions over Integer Sequences

To define a function over integer sequences, we introduce the notion of an abstract pattern, which is a regular expression defined over an abstract alphabet, and we present the notion of a concrete pattern that can be associated with an abstract pattern. We then describe the parameters of a function over integer sequences, one of which isa concrete pattern. Later on, we restrict the values of the parameters describing a function wrt the considered pattern, as to locate unambiguously each pattern occurrence and to avoid overlapping pattern occurrences. Finally, we define the evaluation of a function over integer sequences.

Definition 3 (abstract/concrete alphabets and pattern). *The* abstract alphabet $\mathcal{A}$ *of* k *letters is the set* $\{0, 1, \ldots, k-1\}$. *A* concretisation of $\mathcal{A}$ *is a bijection from* $\mathcal{A}$ *to a set* $\{a_0, a_1, \ldots, a_{k-1}\}$, *called the* concrete alphabet. *An* abstract pattern *over a finite abstract alphabet* $\mathcal{A}$ *is a regular expression over* $\mathcal{A}$. *A* concrete pattern *is obtained from an abstract pattern over an abstract alphabet* $\mathcal{A}$ *by applying a concretisation of* $\mathcal{A}$.

For example, consider the abstract alphabet $\mathcal{A} = \{0, 1\}$ and its concretisation C mapping 0 to ' $\notin$ ' and 1 to ' $\in$ '. The concrete pattern ' $\in\in^*$ ' is obtained by applying C to the abstract pattern '11^*'.

Definition 4 (parametrised function over integer sequences). *A* function over integer sequences $\mathcal{F}$ *is parametrised by* $\langle \psi, \mathrm{sig}_a \rangle$, $\langle f, g, h \rangle$, $\langle before, after \rangle$, $\langle balance \rangle$, *and* $\langle skip \rangle$, *where:*

- ψ *is a concrete pattern over a concrete alphabet* Σ, *and* sig_a *is a total surjective function of arity* $a \in \mathbb{N}^*$ *mapping* $\mathbb{Z}^a$ *to* Σ;
- f, g, *and* h *are respectively one of the features* `one`, `width`, `surface`, `max`, `min`, *one of the primary aggregators* `Sum`, `Max`, `Min`, *and one of the secondary aggregators* `Id`, `Max`, `Min` *defined in Table 1;*
- *before and after are non-negative integers, whose role is to trim the left and right borders of maximal occurrences of the pattern* ψ *in an integer sequence;*
- *balance* $\in \{0, 1\}$ *indicates whether, for computing the feature value, we use only* f *(value* 0*) or both* f *and* $-f$ *(value* 1*);*
- *skip* $\subset \Sigma$ *is the subset of possibly skipped symbols when computing the value of* f *depending on the phase of recognising* ψ *when computing function* $\mathcal{F}$ *on an integer sequence.*

Before defining the result value of a function over integer sequences, we extend the notion of *signature sequence* [5] of an integer sequence to an arbitrary arity, and the notion of *e-occurrence*, which now depends on the new parameter *skip*, which allows us to skip some positions inside a pattern occurrence. The presented notions will be further illustrated in Example 1.

Definition 5 (signature sequence). *Consider an integer sequence* $X = \langle X_1, \ldots, X_n \rangle$ *and a function* $\mathcal{F}$ *over integer sequences whose function* sig_a *is of arity* a. *The* signature sequence *of* X *wrt* sig_a *is the sequence* $\langle S_1, S_2, \ldots, S_{n-a+1} \rangle$, *where every* S_i *(with* $i \in [1, n-a+1]$*) is equal to* $\mathrm{sig}_a(X_i, X_{i+1}, \ldots, X_{i+a-1})$.

f	value	id_f	$\min_f$	$\max_f$
`one`	1	0	n/a	n/a
`width`	$j-i+1$	0	0	$n+1$
`surface`	$\sum_{k=i}^{j} X_k$	0	$-\infty$	$+\infty$
`max`	$\max_{k\in[i,j]} X_k$	$-\infty$	$-\infty$	$+\infty$
`min`	$\min_{k\in[i,j]} X_k$	$+\infty$	$-\infty$	$+\infty$

g	value	id_g^f
`Sum`	$\sum_{k=1}^{m} f_k$	0
`Max`	$\max_{k\in[1,m]} f_k$	$\min_f$
`Min`	$\min_{k\in[1,m]} f_k$	$\max_f$

h	value	$\mathrm{id}_h^{f,g}$
`Id`	id_g^f	id_g^f
`Max`	$\max(\max_{k\in[1,m]} g_k, \mathrm{id}_g^f)$	id_g^f
`Min`	$\min(\min_{k\in[1,m]} g_k, \mathrm{id}_g^f)$	id_g^f

Table 1: Left: features, their values computed from an integer subsequence $\langle X_i, \ldots, X_j\rangle$, their identity, minimum, and maximum values. Centre (resp. Right): primary aggregators (resp. secondary aggregators), their values computed from a sequence $\langle f_1, \ldots, f_m\rangle$ (resp. $\langle g_1, \ldots, g_m\rangle$), and their identities.

Definition 6 (s-occurrence). *Consider a concrete pattern ψ over an alphabet Σ, a sequence $S = \langle S_1, S_2, \ldots, S_m\rangle$ over Σ, and a subsequence $\langle S_i, S_{i+1}, \ldots, S_j\rangle$, with $1 \le i \le j \le m$, forming a maximal word that matches ψ. The* s-occurrence *of S is the index sequence $\langle i, i+1, \ldots, j\rangle$, denoted by $(i..j)$.*

Definition 7 (found index, e-occurrence, i-occurrence). *Consider a function $\mathcal{F}$ over integer sequences whose function sig_a is of arity a, a concrete pattern ψ, two integer constants 'before' and 'after', and an integer sequence $X = \langle X_1, X_2, \ldots, X_n\rangle$ whose signature sequence is $S = \langle S_1, S_2, \ldots, S_{n-a+1}\rangle$ wrt sig_a. For any s-occurrence $(i..j)$ of ψ in S:*

- *the* found index *is the smallest index k in the interval $[i, j]$ such that the word $S_i S_{i+1} \ldots S_k$ belongs to $\mathcal{L}_\psi$;*
- *the* e-occurrence *is the set of indices $\{i + before, \ldots, j + a - 1 - after\}$ such that an index m belongs to the e-occurrence iff the following condition holds: if $m < k$ and $S_m \in skip$, then there exists a signature symbol $S_t \notin skip$ with $m < t < k$;*
- *the* i-occurrence *is the index sequence $\langle i + before, i + before + 1, \ldots, j + a - 1\rangle$, denoted by $[(i + before)..(j + a - 1)]$.*

Definition 8 (well-formed function). *A function $\mathcal{F}$ parametrised by $\langle \psi, \mathrm{sig}_a\rangle$, $\langle f, g, h\rangle$, $\langle before, after\rangle$, $\langle balance\rangle$, and $\langle skip\rangle$ is* well-formed *iff:*

$$balance = 1 \Rightarrow (f = \texttt{width} \vee f = \texttt{surface}) \ \wedge \ after = 0 \tag{1}$$

$$before < \min_{w\in\mathcal{L}_\psi} |w| \ \wedge \ before + after < \min_{w\in\mathcal{L}_\psi} |w| + a - 1 \tag{2}$$

$$before \ge \mathrm{o}_\psi \tag{3}$$

$$\forall p \in \overrightarrow{\psi}, \forall w \in \mathcal{L}_\psi, \exists v_1, v_2 \in \Sigma^* \ (p = v_1 w v_2 \Rightarrow v_1 w \in \mathcal{L}_\psi) \tag{4}$$

$$\exists c \in \mathbb{Z} \ (\mu_\psi \le c) \tag{5}$$

$$\varepsilon \in \mathcal{L}_\psi \ \Rightarrow \ \forall e \in \Sigma \ (e \in \mathcal{L}_\psi) \wedge after = a - 1 \tag{6}$$

Abstract alphabet	Abstract pattern	Arity	Concrete alphabet	Concrete pattern	Concrete function	№
$\langle 0, 1\rangle$	11^*	2	$\langle \leq, > \rangle$	$>>^*$	$\langle$`one`, `Sum`, `Id`, 0, 0, 0, $\emptyset\rangle$	①
		1	$\langle \notin, \in \rangle$	$\in\in^*$	$\langle$`width`, `Max`, `Id`, 0, 0, 0, $\emptyset\rangle$	②
$\langle 0, 1, 2\rangle$	$0(1\|0)^*(2\|1)^*2$	2	$\langle <, =, > \rangle$	$<(=\mid <)^*(>\mid =)^*>$	$\langle$`surface`, `Max`, `Id`, 0, 0, 1, $\{=\}\rangle$	③
		2	$\langle >, =, < \rangle$	$>(=\mid >)^*(<\mid =)^*<$	$\langle$`width`, `Sum`, `Id`, 1, 1, 0, $\emptyset\rangle$	④
$\langle 0, 1\rangle$	$1^*0\|1^*$	2	$\langle =, \neq \rangle$	$=^*\neq \mid =^*$	$\langle$`one`, `Sum`, `Id`, 0, 1, 0, $\emptyset\rangle$	⑤
$\langle 0\rangle$	0	1	$\langle \top \rangle$	$\top$	$\langle$`surface`, `Sum`, `Min`, 0, 0, 0, $\emptyset\rangle$	⑥
$\langle 0, 1\rangle$	1	k	$\langle \notin, \in \rangle$	$\in$	$\langle$`one`, `Sum`, `Id`, 0, 0, 0, $\emptyset\rangle$	⑦

Table 2: Examples of functions, where $\mathcal{F}_①, \mathcal{F}_②, \ldots, \mathcal{F}_⑦$ stand for NB_STRICTLY_DECREASING_SEQUENCE, MAX_WIDTH_GROUP, MAX_SURF_BALANCE_PEAK, SUM_WIDTH_VALLEY, NB_STRETCH, MIN_SUM_SURF_TRUE and NB_IN.

Condition (2) forces every i-occurrence of ψ to be non-empty. Condition (3) imposes disjointness of any two i-occurrences of ψ. By Condition (4), there is discontinuity in the recognition of ψ, allowing us to avoid any regular expression ψ whose language contains words v, w such that v is a proper factor of w, and after consuming a prefix of w whose suffix is v we cannot decide whether v or w is a maximal occurrence of ψ. While extracting an occurrence of a pattern ψ for any possible mismatch, we need to know in advance the length of the suffix keep, which is ensured by Condition (5). Condition (6) is motivated by the fact that, if $\varepsilon \in \mathcal{L}_\psi$, then every sequence contains at least one occurrence of ψ, even if the the sequence is smaller than a. Definition 9 shows how to use Condition (6).

We now define the result value of a function $\mathcal{F}$ over integer sequences.

Definition 9 (function evaluation). *Consider a function $\mathcal{F}$ parameterised by $\langle \psi, \mathrm{sig}_a\rangle$, $\langle f, g, h\rangle$, $\langle before, after\rangle$, $\langle balance\rangle$, $\langle skip\rangle$. For any integer sequence X, the result of $\mathcal{F}$ from X is (R_1, R_2) if $h \neq$ `Id`, R_1 otherwise, where R_1 (resp. R_2) is obtained by applying the aggregator g (resp. h) to the list $\langle f_1, f_2, \ldots, f_t\rangle$ (resp. $\langle g_1, g_2, \ldots, g_t\rangle$), where every g_i is equal to $g(f_1, f_2, \ldots, f_i)$, and every f_i is computed from to the e-occurrence i $\{i_1, \ldots, i_\ell\}$ as follows:*

- *If balance is 0, then f_i is equal to $f(X_{i_1}, X_{i_2}, \ldots, X_{i_\ell})$.*
- *If balance is 1, then f_i is equal to $|f(X_{i_1}, X_{i_2}, \ldots, X_{i_{k-1}}, -X_{i_{k+a-1}}, \ldots, -X_{i_\ell})|$, where i_k is the found index of the s-occurrence i of ψ.*

If the signature of X does not contain any s-occurrences of ψ, then R_1 (resp. R_2) is equal to id_g^f (resp. $\mathrm{id}_h^{f,g}$) according to Table 1. Note that when $\varepsilon \in \mathcal{L}_\psi$, we add a sequence of $a-1$ arbitrary integers at the end of the input sequence.

Example 1. Table 2 provides seven examples of well-formed functions. In examples ① and ② the same abstract alphabet is associated with several concrete alphabets, with even signatures of different arities: in ①, $sig_2(X_i, X_{i+1}) =$ ' $\leq$ ' $\Leftrightarrow$ $X_i \leq X_{i+1} \wedge sig_2(X_i, X_{i+1}) =$ ' $>$ ' $\Leftrightarrow$ $X_i > X_{i+1}$, while in ②,

f	ϕ_f	δ_f^i
`one`	1	1
`width`	$\lambda x, y.x + y$	1
`surface`	$\lambda x, y.x + y$	X_i
`max`	$\lambda x, y. \max(x, y)$	X_i
`min`	$\lambda x, y. \min(x, y)$	X_i

g	ϕ_g
`Max`	$\lambda x, y. \max(x, y)$
`Min`	$\lambda x, y. \min(x, y)$
`Sum`	$\lambda x, y.x + y$

h	ϕ_h
`Max`	$\lambda x, y. \max(x, y)$
`Min`	$\lambda x, y. \min(x, y)$
`Id`	$\lambda x, y.\ y$

Table 3: (Left) Features and their operators ϕ_f and δ_f^i. (Center) (resp. Right) Aggregators (resp. secondary aggregators) and their operators ϕ_g (resp. ϕ_h).

$sig_1(X_i) =$ ' $\notin$ ' $\Leftrightarrow X_i \notin \mathcal{V} \wedge sig_1(X_i) =$ ' $\in$ ' $\Leftrightarrow X_i \in \mathcal{V}$ where $\mathcal{V}$ is a set of integers. $\mathcal{F}_{①}(\langle 1, \boxed{1,0}, 0, \boxed{1,0}, 0, 1\rangle) = 2$ since we have two maximal occurrences of ' $>>^*$ ' (highlighted in grey), and $\mathcal{F}_{②}(\langle 0, \boxed{1}, 0, \boxed{1,1}\rangle)$ with $\mathcal{V} = \{1\}$ is equal to 2 since the maximum number of consecutive ones is 2 (also highlighted). The patterns associated with ③ and ④ correspond to the peak and valley patterns. $\mathcal{F}_{③}(\langle \boxed{0,1,1,1}, 2, \boxed{1,0}, \boxed{0,1}, 2, 2, \boxed{1,1,0}\rangle) = 2$, i.e. the maximum difference $\max(|\boxed{3} - \boxed{1}|, |\boxed{1} - \boxed{2}|)$ of the surface located before/after each peak with 5 (resp. 11) being the found index of the first (resp. second) s-occurrence. $\mathcal{F}_{④}(\langle 0, 1, \boxed{0}, 1, 1, 1, \boxed{0,0,0}, 1\rangle) = 4$, the sum of the widths $\boxed{1} + \boxed{3}$ of the 2 valleys.

$\mathcal{F}_{⑤}(\langle \boxed{0}, \boxed{1,1,1}, \boxed{0}, \boxed{1}, \boxed{0}, \boxed{1}\rangle) = 6$ since we have 6 maximal groups of consecutive identical values. Note that $\mathcal{F}_{⑤}(\langle 0\rangle) = 1$ since, from Condition (6) of Definition 9, when $\varepsilon \in \mathcal{L}_{=^*\neq|=^*}$ and the arity of the signature is 2, we add one integer value at the end of the input sequence. In ⑥, $sig_1(X_i) =$ '$\top$' means that every index i of the signature sequence of any input sequence X is an e-occurrence. $\mathcal{F}_{⑥}(X) = \langle 0, 0\rangle$, where $X_i = 1$ (resp. $X_i = -1$) represents an opening (resp. closing) parenthesis models well formed expressions with parentheses. In ⑦, given low, up in $\mathbb{Z}$, $sig_k(X_i, X_{i+1}, \ldots, X_{i+k-1}) =$ ' $\in$ ' $\Leftrightarrow \sum_{\alpha=i}^{i+k-1} X_\alpha \in [low, up]$. $\mathcal{F}_{⑦}(X)$ returns the number of sliding sequences of k consecutive values of X, whose sum is located in the interval $[low, up]$. △

4 Operational View of Functions Over Integer Sequences

To evaluate a function $\mathcal{F}$ wrt an integer sequence X, i.e. see Definition 9, we need to 1) find all s-occurrences of the pattern ψ of $\mathcal{F}$ in the signature sequence of X, and 2) obtain the corresponding e-occurrences to compute the feature values and aggregate them. The qualitative (resp. quantitative) part 1) (resp. part 2)) is called the *recognition* (resp. *computational*) aspect of $\mathcal{F}$. Note that the recognition aspect of $\mathcal{F}$ is only related to its pattern ψ and its alphabet Σ.

We describe in Section 4.1 a specific transducer, called *seed transducer*, for dealing with the recognition aspect of $\mathcal{F}$. Then we show in Section 4.2 how the computational aspect of $\mathcal{F}$ is handled by a *reduced set of instructions* based on the output alphabet of the seed transducer. This set of instructions is parameterised by all the parameters of $\mathcal{F}$, except the pattern ψ, and it allows us to

synthesise a register automaton with a constant number of registers, which returns the value of $\mathcal{F}$ from X after consuming the signature sequence of X. Hence it takes linear time in the length of X to compute the value of $\mathcal{F}$ from X.

4.1 Handling the Recognition Aspect: Seed Transducer

To find all s-occurrences of a pattern in an integer sequence, in the corresponding signature sequence, we introduce the notion of *seed transducers*: first, we describe a seed transducer of an abstract pattern, and show how to obtain the seed transducer of any concrete pattern from the seed transducer of the corresponding abstract pattern. Second, we give the conditions of *well-formedness* of a transducer wrt any given pattern, as well as wrt a given abstract pattern.

Describing the Seed Transducer of an Abstract Pattern Consider an abstract pattern σ, i.e. a regular expression over an abstract alphabet $\mathcal{A}$. A *seed transducer* of σ is a deterministic transducer where each transition is labelled with (1) a symbol in the input alphabet $\mathcal{A}$, called the *input symbol*, and (2) a word made from symbols in the output alphabet Ω, called the *output word*. Hence, a transducer consumes an input sequence of symbols in $\mathcal{A}$ and produces an output sequence where each element in Ω is called a *phase letter*. Consider different possibilities of the produced output symbols when consuming a symbol S_i of some input signature sequence $\langle S_1, S_2, \ldots, S_{n-a+1} \rangle$.

- [**out**]: corresponds to no occurrence of σ.
- [$\mathbf{maybe}_{\mathrm{r}}^{k}$ with k being an integer constant]: indicates the potential new occurrence of σ that has at least k transitions.
- [$\mathbf{maybe}_{\mathrm{b}}$]: indicates the continuation of a potential new occurrence of σ.
- [$\mathbf{out}_{\mathrm{r}}$]: reflects the fact that the previous potential occurrence of σ is not a true occurrence of σ.
- [**found**]: denotes the discovery of a new occurrence of σ.
- [$\mathbf{maybe}_{\mathrm{a}}$]: indicates the potential extension of the latest occurrence of σ.
- [**in**]: corresponds to the extension of the latest discovered occurrence of σ.
- [**end**]: corresponds to the end of the latest discovered occurrence of σ.

Besides the phase letters **in** and $\mathbf{maybe}_{\mathrm{a}}$ whose meaning was left unchanged compared to [5], we have the following modifications:

- Some transitions that were labelled with **out** are now labelled with $\mathbf{maybe}_{\mathrm{b}}$. For example, in [5], given the pattern ' >><<>> ' this was the case for the two transitions recognising the first two occurrences of ' < '; but to make the transducer independent from *before* they are now labelled with $\mathbf{maybe}_{\mathrm{b}}$.
- The letter $\mathbf{maybe}_{\mathrm{r}}^{k}$ was not in the output alphabet of [5]. It has been added in order to capture patterns that require to restart from a small fixed suffix after a mismatch. It also replaces the first occurrence of $\mathbf{maybe}_{\mathrm{b}}$.
- Furthermore, since in [5], any seed transducer could only produce a single phase letter per transition, we had to introduce the letter $\mathbf{found}_{\mathrm{e}}$, which was

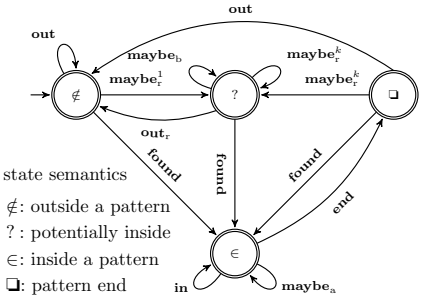
a combination of **found** and **end**. In our new model, this phase letter has disappeared since it is no longer needed. In fact, the same transition may now be labelled with more than one phase letter. For example, given the pattern $\sigma =$ ' >><<>> ', the transition associated with the recognition of σ is labelled by the input symbol ' > ', i.e. the last symbol of σ, and the output word '$\mathbf{found\,end\,maybe}^2_{\mathrm{r}}$': **found** indicates that a new occurrence of σ was found, **end** denotes that this new occurrence ended, and $\mathbf{maybe}^2_{\mathrm{r}}$ indicates that potentially there is a next occurrence of σ whose prefix corresponds to the last two encountered input symbols, i.e. ' >> '.

From the seed transducer of an abstract pattern we obtain the seed transducer of a concrete pattern by the concretisation of the alphabet $\mathcal{A}$.

Well-Formed Transducer We describe the structural properties a seed transducer must have. Condition (1) implies that it is always possible in the future to have an occurrence of pattern σ, Condition (2) defines a partial order between the different phase letters of the same pattern occurrence, Condition (3) forbids the sequence $\mathbf{maybe}^k_{\mathrm{r}}\mathbf{maybe}^{k+1}_{\mathrm{r}}$, which can be replaced by $\mathbf{maybe}^k_{\mathrm{r}}\mathbf{maybe}_{\mathrm{b}}$.

Definition 10 (necessary conditions for a well-formed transducer). *A seed transducer $\mathcal{S}$ is* well-formed *if all the following conditions hold:*

1. *There is a path from each transition to each transition labelled by a* **found**.
2. *The output language is accepted by the automaton at the right.*
3. *For any state q we cannot have simultaneously a transition labelled by* $\mathbf{maybe}^k_{\mathrm{r}}$ *entering q, and a transition labelled by* $\mathbf{maybe}^{k+1}_{\mathrm{r}}$ *exiting q.*



Well-formed output language

Well-Formed Transducer wrt an Abstract Pattern We introduce the notion of a *well-formed transducer wrt an abstract pattern* σ, which guarantees that a transducer recognises all maximal occurrences of an abstract pattern. We first present the notion of t-occurrence as an interval of indices of specific words in the output sequence of the transducer. Finally, we state that, for any path p leading to a state q, the length of the longest suffix in $\overrightarrow{\sigma}$ of the sequence of input symbols of the transitions of p is either 1) a constant and is smaller than $\overline{before}_\sigma + 1$, or 2) is greater than or equal to $\overline{before}_\sigma + 1$, where $\overline{before}_\sigma$ is the largest value of *before* of a well-formed function whose concrete pattern is obtained from the abstract pattern σ. Note that, by Definition 8, such $\overline{before}_\sigma$ always exists and depends only on σ.

Definition 11 (t-occurrence). *Given a seed transducer $\mathcal{S}$ of some abstract pattern over an abstract alphabet $\mathcal{A}$ and an input sequence of symbols of $\mathcal{A}$, the* t-occurrence *of $\mathcal{S}$ for s consists of the indices of the phase letters of a maximal word within the transduction of s that matches the regular expression* '$(\varepsilon|\mathbf{maybe}_{\mathrm{r}}^{k}\mathbf{maybe}_{\mathrm{b}}^{*})\mathbf{found}(\mathbf{maybe}_{\mathrm{a}}^{*}\mathbf{in})^{*}$ '.

Definition 12 ($\mathbf{maybe}_{\mathrm{b}}$-degree of a path). *Consider an abstract pattern σ, and a* path *p, a sequence of consecutive transitions, wrt its transducer $\mathcal{T}_\sigma$.*

- *The* $\mathbf{maybe}_{\mathrm{b}}$-suffix *of p is the maximal suffix of the sequence of output symbols of the transitions of p that matches* '$(\mathbf{maybe}_{\mathrm{r}}^{k}|\varepsilon)\mathbf{maybe}_{\mathrm{b}}^{*}$'.
- *The* $\mathbf{maybe}_{\mathrm{b}}$-degree *of p is* $\min(\overline{before}_{\sigma}+1,\ell)$*, where ℓ is the length of the* $\mathbf{maybe}_{\mathrm{b}}$*-suffix of p plus $k-1$, the degree of* $\mathbf{maybe}_{\mathrm{r}}^{k}$*, if the suffix starts with* $\mathbf{maybe}_{\mathrm{r}}^{k}$.

Definition 13 ($\mathbf{maybe}_{\mathrm{b}}$-degree of a state). *Consider an abstract pattern σ and its transducer $\mathcal{T}_\sigma$. For every state q of $\mathcal{T}_\sigma$, if every path from the initial state of $\mathcal{T}_\sigma$ to the state q has the same* $\mathbf{maybe}_{\mathrm{b}}$*-degree d, then the* $\mathbf{maybe}_{\mathrm{b}}$-degree of *$q$ is equal to d; otherwise, the* $\mathbf{maybe}_{\mathrm{b}}$*-degree of q is undefined.*

Definition 14 (necessary conditions for a well-formed transducer wrt a pattern). *A seed transducer $\mathcal{S}$ is* well-formed wrt an abstract pattern *σ over an alphabet $\mathcal{A}$ if all the following conditions hold:*

1. *It is well-formed in the sense of Definition 10.*
2. *For any state of $\mathcal{S}$, its* $\mathbf{maybe}_{\mathrm{b}}$*-degree is defined.*
3. *For any input sequence S of symbols of $\mathcal{A}$, for any t-occurrence $[[i..j]]$ of $\mathcal{S}$, there exists an s-occurrence $(i-k+1..j)$ of σ in S, where k is the degree of* $\mathbf{maybe}_{\mathrm{r}}^{k}$*, if the t-occurrence $[[i..j]]$ has one, and is 1, otherwise.*

Example 2. Figures 1(A)–1(E) respectively give the seed transducer for the patterns ' $>=^{+}>$ ', ' $>><<>>$ ', ' $=^{*}\neq \mid =^{*}$ ' (the STRETCH pattern in ⑤), ' $\in^{+}$ ' (the GROUP pattern in ②) and ' $<^{+} \mid >^{+}$ '. The minimum and maximum values of *before* and *after* are set up according to Conditions 2 and 3 of Definition 8. In Figure 1(A) the $\mathbf{maybe}_{\mathrm{b}}$-degree of states s, r, t and t' is respectively equal to 0, 1, 2 and 3. Note that states t and t' cannot be merged since, according to Definition 13, the $\mathbf{maybe}_{\mathrm{b}}$-degree of the merged state would be undefined. In Figure 1(B) the $\mathbf{maybe}_{\mathrm{b}}$-degree of states s, r, t, u and v is respectively equal to 0, 1, 2, 3 and 4. In Figures 1(C)–1(E) the $\mathbf{maybe}_{\mathrm{b}}$-degree of all states is 0, since the corresponding transducers do no mention neither $\mathbf{maybe}_{\mathrm{b}}$, nor $\mathbf{maybe}_{\mathrm{r}}^{k}$. △

4.2 Handling the Computational Aspect: Reduced Instruction Set

For a well-formed function $\mathcal{F}$ whose concrete pattern is ψ, and an integer sequence X, if we know where the s-occurrences of σ in X are located, we can compute the value of $\mathcal{F}$. In a single pass, we aim to both 1) detect all s-occurrences of ψ in X, and 2) compute $\mathcal{F}$ from the subsequences of X corresponding to

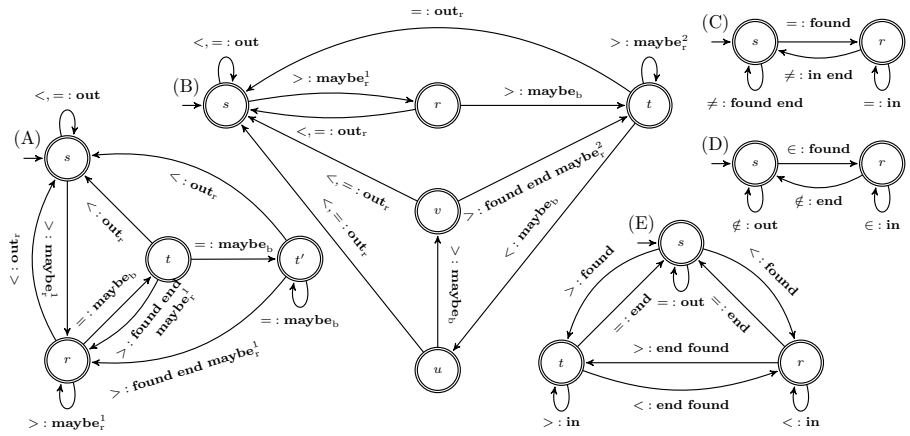


Fig. 1: Seed transducers for (A) ' $>=^+>$ ' with *before* $\in [1,2]$, *after* $\in [0,2]$ and the alphabet $\{<,=,>\}$, (B) ' $>><<>>$ ' with *before* $\in [0,4]$, *after* $\in [0,2]$ and the alphabet $\{<,=,>\}$, (C) ' $=^*\neq \mid =^*$ ' with *before* $= 0$, *after* $= 1$ and the alphabet $\{=,\neq\}$, (D) ' $\in^+$ ' with *before* $= 0 =$ *after* $= 0$ and the alphabet $\{\in,\notin\}$, (E) ' $<^+ \mid >^+$ ' with *before* $= 0$, *after* $\in [0,1]$ and the alphabet $\{<,=,>\}$.

e-occurrences. To do so, we describe a *reduced instruction set* for computing $\mathcal{F}$, which is associated to the phase letters. The reduced instruction set manipulates registers, whose values are updated by performing *micro instructions*. When the seed transducer consumes the next symbol of an signature sequence, a sequence of micro instructions, called a *macro instruction*, is executed.

- In our model, we consider 5 registers described in Part (a).
- The reduced instruction set has 4 micro instructions described in Part (b).
- The macro instructions corresponding to the phase letters of the seed transducer of ψ are described in Part (c).

(a) Registers of the Reduced Instruction Set The evaluation of a well-formed function can be decomposed into at most five levels of computations organised in the following three layers:

- [Past] Level 4 (resp. 3) records the aggregation wrt the aggregator h (resp. g) of the pattern occurrences already completed.
- [Present] Level 2 records the feature value of the current not already completed pattern occurrence.
- [Future] Levels 1 and 0 record the feature value of an hypothetical occurrence of pattern that must be confirmed or invalidated later on, depending of what will be read next. Level 0 is called the *bottom* level.

With each level $\ell \in [0,4]$ we associate a register V_ℓ and a function ϕ_ℓ defined according to Table 3 as follows:

- ϕ_4 is ϕ_h (with $h \in \{\texttt{Max}, \texttt{Min}, \texttt{Id}\}$) and the initial value of V_4 is $\mathrm{id}_h^{f,g}$.
- ϕ_3 is ϕ_g (with $g \in \{\texttt{Max}, \texttt{Min}, \texttt{Sum}\}$) and the initial value of V_3 is id_g^f.
- ϕ_0, ϕ_1 and ϕ_2 correspond all to ϕ_f (with $f \in \{\texttt{max}, \texttt{min}, \texttt{one}, \texttt{surface}, \texttt{width}\}$), and the initial value of V_0, V_1 is id_f, while the initial value of V_2 is id_g^f.

(b) Micro Instructions of the Reduced Instruction Set The next table describes the available micro instructions for modifying register values:

- `compute` the (potential or not) feature value of a pattern occurrence,
- `reset` all registers from the bottom to a given level to their identity values,
- `transmit` the register content of a level to the register of the next level,
- `set` the feature value of the next potential pattern occurrence after a mismatch.

micro instruction	register updates
$\texttt{compute}(\ell, b, v)$	: **if** $b = 0$ **then** $V_\ell \leftarrow \phi_\ell(V_\ell, v)$ **else** $V_\ell \leftarrow \phi_\ell(V_\ell, -v)$
$\texttt{reset}(\ell)$	: **for** $k \in [0, \ell]$ **do** $V_k \leftarrow \mathrm{id}_k$
$\texttt{transmit}(c, b, \ell)$	: **if** $c = 1$ **then** $V_{\ell+1} \leftarrow V_\ell$
	else if $b = 1$ **then** $V_{\ell+1} \leftarrow \phi_{\ell+1}(V_{\ell+1}, \lvert V_\ell \rvert)$
	else $V_{\ell+1} \leftarrow \phi_{\ell+1}(V_{\ell+1}, V_\ell)$
$\texttt{set}(\ell, k)$	: **if** $\mathit{before} + 1 - k > 0$ **then** $V_\ell \leftarrow \mathrm{id}_\ell$
	else if $\mathit{before} + 1 - k = 0$ **then** $V_\ell \leftarrow \delta_f^i$
	else $V_\ell \leftarrow \phi_\ell(\delta_f^{i-k+1+\mathit{before}}, \ldots, \delta_f^i)$

Note that all values $\phi_\ell(\delta_f^{i-k+1+\mathit{before}}, \ldots, \delta_f^i)$, where k are the integer values occurring in the $\mathbf{maybe}_{\mathrm{r}}^k$ phase letter of a seed transducer and $i \in [k - \mathit{before}, n - a + 1]$ is the index of the signature symbol we are processing, used in the 'set' micro instruction, are computed in advance in an initialisation phase in linear time wrt the sequence length so that they are directly available. Also, like in [5], each micro instruction can be turned into a constraint to induce a reformulation of the original constraint. This is not developed here for space reasons.

(c) Macro Instructions of the Reduced Instruction Set Given a well-formed function $\mathcal{F}$ and its concrete pattern ψ, we describe the macro instructions associated with each phase letter of the seed transducer of ψ. A macro instruction may depend on the $\mathbf{maybe}_{\mathrm{b}}$-degree, denoted d in Table 4, of the end state of a transition labelled by the corresponding phase letter. Table 4 defines the macro instructions where the function κ is defined just after. The precondition of a macro instruction must hold in order to execute its corresponding code.

Depending on which of the following conditions holds, $a - 1 - \mathit{after} < 0 \wedge \mathit{balance} = 0$, $a - 1 - \mathit{after} = 0 \wedge \mathit{balance} = 0$, $a - 1 - \mathit{after} > 0 \wedge \mathit{balance} = 0$, $\mathit{balance} = 1$, the function κ used in **found** is respectively defined as id_f, δ_f^i, $\phi_f(\delta_f^i, \ldots, \delta_f^{i+a-1-\mathit{after}})$, δ_f^{i+a-1}, where i is the index of the current signature symbol we are processing.

letter		precondition	macro instruction code
$\mathbf{maybe}_{\mathrm{b}}$	:	$\begin{pmatrix} s \notin \mathit{skip} \wedge \\ d > \mathit{before} \end{pmatrix}$	$\texttt{compute}\left(1, 0, \delta_f^i\right)$, $\texttt{transmit}(0,0,0)$, $\texttt{reset}(0)$
		$\begin{pmatrix} s \in \mathit{skip} \wedge \\ d > \mathit{before} \end{pmatrix}$	$\texttt{compute}\left(0, 0, \delta_f^i\right)$
$\mathbf{maybe}_{\mathrm{r}}^k$	:		$\texttt{reset}(1)$, $\texttt{set}(1,k)$
$\mathbf{out}_{\mathrm{r}}$	:		$\texttt{reset}(1)$
found	:		$\texttt{compute}(1, \mathit{balance}, \kappa)$, $\texttt{transmit}(1,0,1)$, $\texttt{reset}(1)$
$\mathbf{maybe}_{\mathrm{a}}$	:		$\texttt{compute}\left(1, \mathit{balance}, \delta_f^{i+a-1-\mathit{after}}\right)$
in	:		$\texttt{compute}\left(1, \mathit{balance}, \delta_f^{i+a-1-\mathit{after}}\right)$, $\texttt{transmit}(0,0,1)$, $\texttt{reset}(1)$
end	:		$\texttt{transmit}(0, \mathit{balance}, 2)$, $\texttt{transmit}(0,0,3)$, $\texttt{reset}(2)$

Table 4: Macro instructions of the reduced instruction set, where d denotes the $\mathbf{maybe}_{\mathrm{b}}$-degree of the end state of a transition labelled by the corresponding phase letter.

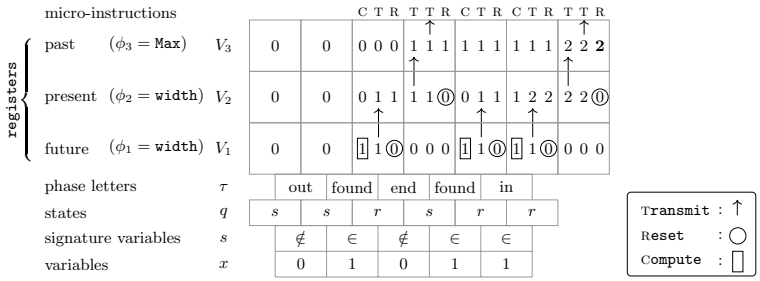


Fig. 2: Trace for the MAX_WIDTH_GROUP constraint, i.e. Example ② of Table 2 on the sequence $\langle 0,1,0,1,1\rangle$: evolution of the register values V_1, V_2, V_3 while executing the micro-instructions `Compute`, `Reset` and `Transmit` leading to the result **2** shown in bold on the right upper corner (since they are not relevant for this example, registers V_0 and V_4 are not shown).

(d) Value Returned by the Function After consuming a signature sequence, the function performs the macro instruction of the **end** phase letter. If h different from `Id`, then the function returns the last values of the registers V_3 and V_4. If h equal to `Id`, then the function only returns the last value of the register V_3.

Figure 2 shows the evaluation of the function in Example ② of Table 2. It provides the phase letters produced by transducer (D) of Figure 1, and the corresponding sequence of micro-instructions updating the registers V_1, V_2 and V_3.

5 Generation of Seed Transducers

We present an algorithm to generate a well-formed seed transducer as specified in Definition 10 from a well-formed function and its regular expression. We present in Section 5.1 the modifications made to the algorithm of [11] for generating

seed transducers so as to generate seed transducers satisfying Definition 10. In Section 5.2 we present a new transducer disambiguation algorithm that takes into account both the new output alphabet of seed transducers and the fact that the transitions in our seed transducers can have more than one output symbol.

5.1 Generating a Seed Transducer from a Regular Expression

Recall that a *deterministic finite automaton* (DFA) is a tuple $\langle Q, \Sigma, \delta, q_0, Q_a \rangle$, where Q is the set of states, Σ is the alphabet, $\delta\colon Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the start state, and $Q_a \subseteq Q$ is the set of accepting states. For any regular expression σ that satisfies the conditions given in Definition 8, given a minimal DFA $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, Q_a \rangle$ accepting $\mathcal{L}_\sigma$, a transducer can be generated by means of the following algorithm:

1. Expand $\mathcal{M}$ using Algorithms 1 and 2 of [11], with the difference that instead of using the exact value of *before* as an input for Algorithm 2, here we use a constant β equal to the minimum value of *before* that satisfies Definition 8.
2. As in [11], divide Q into three disjoint sets: Q_a, the set Q^- of states reachable from q_0 without passing through an accepting state, and the set Q^+ of states that can be reached from an accepting state, excluding the accepting states. After applying Algorithm 1 of [11] we know that $Q^+ = Q \setminus Q_a \setminus Q^-$.
3. Set the output symbol of every transition in $\mathcal{M}$ as in the algorithm of [11] except for the following cases:
   (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 **found end**.
   (b) 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 **in end**.
   (c) For every transition on a path of length β leaving the initial state: we now set the output symbol to $\mathbf{maybe}_r^k$, where $1 \leq k \leq \beta$ is the length of the path. This will not overwrite the symbols already set, because of the expansion performed in Step 1 by Algorithm 2 of [11].
   (d) For all remaining transitions in $\mathcal{M}$ leaving q_0: set the output symbol to $\mathbf{maybe}_r^1$. If there is a loop in q_0, then first expand $\mathcal{M}$ using Algorithm 2 in [11] and the value 1.
4. As in [11], concatenate $\mathcal{M}$ to a transducer for Σ^* where all the output sequences are set to **out**, creating a non-deterministic finite transducer for $\Sigma^*\mathcal{L}_\sigma$.
5. Using our own disambiguation algorithm described in Section 5.2, determinise the transducer for $\Sigma^*\mathcal{L}(\sigma)$.
6. Replace output symbols so that the transducer is well-formed as specified in Definition 10 using the following rules:
   (a) For every transition t from a state q in $Q_a \cup Q^+$ to a state in Q^-: if q has incoming transitions with the output symbols **in** or $\mathbf{maybe}_a$, then:
      - If the output symbol of t is **out**, then replace it with **end**.
      - If the output symbol of t is $\mathbf{maybe}_b$, then replace it with **end** $\mathbf{maybe}_r^1$.
   (b) For every transition from a state in $Q_a \cup Q^+$ with the output symbol **found**: replace the output symbol by **end found**.

(c) For every transition t from a state q in Q^- with the output symbol **out**: if q has incoming transitions with the output symbol $\mathbf{maybe}_{\mathrm{b}}$ or $\mathbf{maybe}_{\mathrm{r}}^{k}$, then replace the output symbol of t by $\mathbf{out}_{\mathrm{r}}$. Note that we do not need to expand the transducer by following Algorithm 3 in [11].
(d) For every transition t from the initial state q_0 with the output symbol $\mathbf{maybe}_{\mathrm{r}}^{1}$, replace the output symbol of t by $\mathbf{maybe}_{\mathrm{b}}$.
(e) For every state q, if q has incoming transitions with the output symbol $\mathbf{maybe}_{\mathrm{r}}^{k}$, the if q has outgoing transitions with the output symbol $\mathbf{maybe}_{\mathrm{r}}^{k+1}$ replace it with the symbol $\mathbf{maybe}_{\mathrm{b}}$.

7. Mark all states as accepting and minimise the transducer [10].
8. For every state q, compute its $\mathbf{maybe}_{\mathrm{b}}$-degree by enumerating paths ending in q. If there are $p > 1$ different $\mathbf{maybe}_{\mathrm{b}}$-degrees of such paths, i.e. the $\mathbf{maybe}_{\mathrm{b}}$-degree of q is undefined, then replace q with p states corresponding to p different $\mathbf{maybe}_{\mathrm{b}}$-degree of paths ending in q and adjust the transitions accordingly. Repeat the whole step for the transducer with the duplicated states until the $\mathbf{maybe}_{\mathrm{b}}$-degree of every state is defined.

Example 3. Consider the regular expression $\sigma =$ ' $>=^{+}>$ '. The minimum-state automaton $\mathcal{M}$ accepting $\mathcal{L}_\sigma$ is in Figure 3(A). The expansion algorithms in Step 1 do not modify $\mathcal{M}$. The set of states $Q = \{s, r, t, u\}$ of $\mathcal{M}$ is divided into the subsets $Q_{\mathrm{a}} = \{u\}$, $Q^- = \{s, r, t\}$, and $Q^+ = \emptyset$. We then proceed to set the output symbols of the transitions in order to build the transducer for σ. After Step 3a, we have the transducer in Figure 3(B). After Step 3c, we have the transducer in Figure 3(C). After finishing Step 3, we have the transducer in Figure 3(D). After concatenating the transducer for Σ^* in Step 4 we have the transducer in Figure 3(E). After determinising the transducer in Step 5, we have the transducer in Figure 3(F). After Step 7, once all the states are made accepting and the obtained transducer is minimised, we have the minimal transducer in Figure 3(G). Finally, after Step 8 we have the transducer in Figure 1(A). △

5.2 Disambiguation Algorithm

Our disambiguation algorithm is an extension of the one in [11], which is in turn based on the powerset construction [15] used for the determinisation of automata. In the transducer case, [11] redefines the transition function of the powerset construction so that for any transition between two states of the deterministic transducer, only the maximum output symbol among all the possible ones is kept, that is, $\delta'(q, a) = \langle \{u \mid \langle u, * \rangle \in T\}, (\max(b) \mid \langle *, b \rangle \in T) \rangle$, where $T = \{\delta(r, a) \mid r \in q\}$. Here transitions can have more than one output symbol, and so we define the transition function as in [11], except that we choose the output sequence that is lexicographically larger. Moreover, when comparing the sequences **found end** and $\mathbf{maybe}_{\mathrm{r}}^{k}$, we merge them into **found end** $\mathbf{maybe}_{\mathrm{r}}^{k}$.

6 Conclusion, Related Work, and Future Work

Our contributions over related work can be summarised as follows:

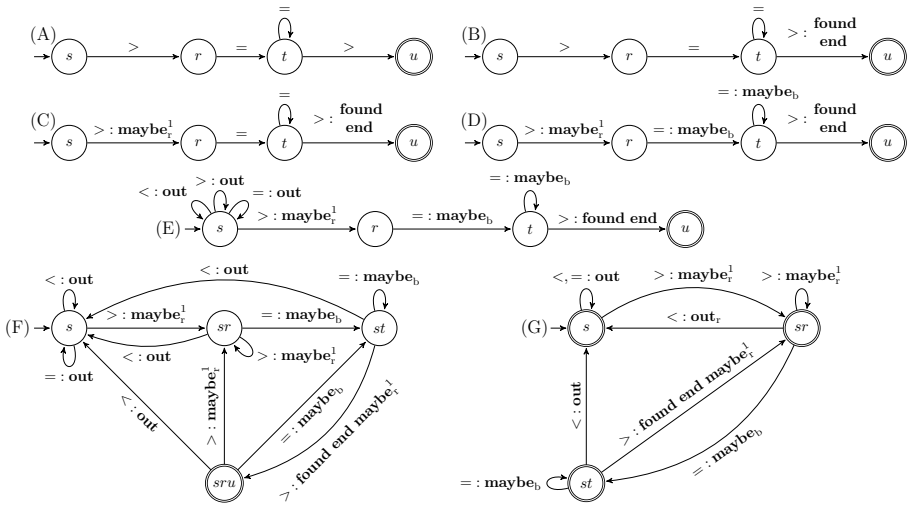


Fig. 3: (A) The minimal automaton recognising $\mathcal{L}(\text{`}{>}{=}^{+}{>}\text{'})$. Intermediate transducer for ' $>=^{+}>$ ' after applying (B) Step 3a, (C) Step 3c, (D) Step 3, (E) Step 4, (F) Step 5, and (G) Step 7 of the algorithm in Section 5.1

1. We have extended the *qualitative* aspect of the transducer-based computational model of [5]. The input alphabet of transducers is not fixed to $\{<, =, >\}$, that is the binary topological comparison operators that are useful for time-series constraints, but can be any set of operators, including unary ones (such as $\{\in, \notin\}$ with fixed sets, used in [1]) and k-ary ones (as frequently used in the Global Constraint Catalogue [6]). The output alphabet of transducers is augmented by $\mathbf{maybe}_{\mathrm{r}}$ and simplified, since transducers can output more than one letter for each input letter.
2. We have parametrised the *quantitative* aspect of the computation:
   - The model of [5] had parameters for trimming the borders of a maximal occurrence of a regular expression, with the major drawback that transducers were dependent on these parameters devoted to the quantitative aspect of the computation. In the new model, transducers are independent of such trimming parameters.
   - Within a maximal occurrence of a regular expression, based on the current recognition phase, a function f or its opposite $-f$ may now be used for computing the contribution of an input letter to the feature value.

   While regular expressions and transducers are already used in the context of frequent sequence mining [3], they are focussed on the qualitative aspect, i.e. they do not compute a value for each pattern occurrence.
3. We upgraded the automatic transducer generator of [11].
4. The small number of phase letters and the very small set of micro instructions allow a compact implementation of checkers and reformulation.

While learning from a large collection of examples can be done with neural networks without assuming any bias, learning from very few examples still requires a proper bias. Hence, future work may exploit the canonical form introduced here to learn constraint models having functional constraints on integer sequences both from very few samples [8] and with few queries [9].

References

1. Alur, R., Fisman, D., Raghothaman, M.: Regular programming for quantitative properties of data streams. In: Thiemann, P. (ed.) ESOP 2016. LNCS, vol. 9632, pp. 15–40. Springer (2016)
2. Arafailova, E., Beldiceanu, N., Douence, R., Carlsson, M., Flener, P., Francisco Rodríguez, M.A., Pearson, J., Simonis, H.: Global Constraint Catalog, Volume II, Time-Series Constraints. CoRR **abs/1609.08925** (2016), http://arxiv.org/abs/1609.08925
3. Beedkar, K., Gemulla, R.: Desq: Frequent sequence mining with subsequence constraints. In: 2016 IEEE 16th International Conference on Data Mining (ICDM). pp. 793–798 (Dec 2016). https://doi.org/10.1109/ICDM.2016.0092
4. Beldiceanu, N., Carlsson, M., Debruyne, R., Petit, T.: Reformulation of global constraints based on constraints checkers. Constraints **10**(4), 339–362 (2005)
5. Beldiceanu, N., Carlsson, M., Douence, R., Simonis, H.: Using finite transducers for describing and synthesising structural time-series constraints. Constraints **21**(1), 22–40 (January 2016)
6. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog, 2nd Edition (revision a). Tech. Rep. T2012:03, Swedish Institute of Computer Science (February 2012), http://soda.swedish-ict.se/view/sicsreport/
7. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathematical and Computer Modelling **20**(12), 97–123 (1994)
8. Beldiceanu, N., Simonis, H.: Modelseeker: Extracting global constraint models from positive examples. In: Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach, pp. 77–95 (2016)
9. Bessière, C., Daoudi, A., Hebrard, E., Katsirelos, G., Lazaar, N., Mechqrane, Y., Narodytska, N., Quimper, C., Walsh, T.: New approaches to constraint acquisition. In: Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach, pp. 51–76 (2016)
10. Choffrut, C.: Minimizing subsequential transducers: A survey. Theoretical Computer Science **292**(1), 131–143 (2003)
11. Francisco Rodríguez, M.A., Flener, P., Pearson, J.: Automatic generation of descriptions of time-series constraints. In: Virvou, M. (ed.) ICTAI 2017. IEEE Computer Society (2017)
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 3rd edn. (2007)
13. Pachet, F., Roy, P.: Automatic generation of music programs. In: Jaffar, J. (ed.) CP 1999, LNCS, vol. 1713, pp. 331–345. Springer (1999)
14. Pesant, G.: A filtering algorithm for the STRETCH constraint. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 183–195. Springer (2001)
15. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development **3**(2), 114–125 (1959)
16. Sakarovitch, J.: Elements of Language Theory. Cambridge University Press (2009)