Noname manuscript No.
(will be inserted by the editor)

Invariants for Time-Series Constraints

Ekaterina Arafailova · Nicolas Beldiceanu · Helmut Simonis

the date of receipt and acceptance should be inserted later

Abstract Many constraints restricting the result of some computations over an integer sequence can be compactly represented by counter automata. We improve the propagation of the conjunction of such constraints on the same sequence by synthesising a database of linear and non-linear invariants using their counter-automaton representation. The obtained invariants are formulae parameterised by the sequence length and proven to be true for any long enough sequence. To assess the quality of such linear invariants, we developed a method to verify whether a generated linear invariant is a facet of the convex hull of the feasible points. This method, as well as the proof of non-linear invariants, are based on the systematic generation of constant-size deterministic finite automata that accept all integer sequences whose result verifies some simple condition. We apply such methodology to a set of 44 time-series constraints and obtain 1400 linear invariants from which 70% are facet defining, and 600 non-linear invariants, which were tested on short-term electricity production problems.

1 Introduction

Many combinatorial problems seek for producing a combinatorial object, e.g. a sequence, a permutation, a tree, having simultaneously several characteristics. In this context identifying invariants that link different characteristics of an object is crucial since these characteristics often cannot vary independently. For instance, in a sequence, the maximum absolute difference between the number of peaks and the number of valleys is bounded by 1.

Although adding invariants is a crucial point in solving combinatorial problems, there is currently no database of invariants, even in the more restricted cases of CP and MIP. In CP invariants are usually added by hand after studying the problem. In MIP, invariants are either added by hand or generated

This is an extended version of the CP 2017 article [4]. Ekaterina Arafailova is supported by the EU H2020 programme under grant 640954 for project GRACeFUL. Nicolas Beldiceanu is partially supported by the GRACeFUL project and by the Gaspard Monge Program for Optimisation and Operations Research (PGMO). Helmut Simonis is supported by Science Foundation Ireland (SFI) under grant SFI/10/IN.1/I3032; the Insight Centre for Data Analytics is supported by SFI under grant SFI/12/RC/2289.

Ekaterina Arafailova and Nicolas Beldiceanu
TASC (LS2N) IMT Atlantique, FR – 44307 Nantes, France
E-mail: {Ekaterina.Arafailova,Nicolas.Beldiceanu}@imt-atlantique.fr

Helmut Simonis
Insight Centre for Data Analytics, University College Cork, Ireland
E-mail: Helmut.Simonis@insight-centre.org

by the system depending on the instance of the problem being solved. Our hypothesis is that it is very useful to produce databases of invariants for families of constraints for the following reasons:

- Having a base of invariants can potentially benefit several technologies, for instance linear invariants benefit both to CP and MIP.
- Having a base of invariants in a computer format allows their diffusion, as well as a critical review, allowing them to be extended and improved in the long term.
- Even if it is costly, the generation of invariants is done once and for all during pre-treatment. Once generated, the extraction of invariants to solve a particular problem is immediate.

This article is a first step in this direction for a conjunction of constraints on time series. This work is motivated by our interest in the generation of electricity production plans based on models learned from past production data [10]. In this context, a large number of models must be generated according to the characteristics of the production units, their operational mode, and the time of year. It is impractical to try and improve all of these models by manually adding invariants as required. Instead we focus on creating the invariants once and for all, and adding them automatically to each of the generated models. Experiments confirm that adding invariants to models significantly speeds up the search for solutions.

We present a framework for synthesising necessary conditions for a conjunction of two sequence constraints that are each represented by a counter automaton [14], and are imposed on the same integer sequence of length n. Our necessary conditions are in the form of linear inequalities, implications whose right-hand side is a linear inequality, and disjunctions of inequalities. In addition, they are parameterised by n and instance-independent, i.e. they are true for any integer sequence of length n greater than some small constant.

In order to synthesise linear inequalities and implications with linear inequalities we draw full benefit from counter automata representing the constraints since they do not encode explicitly all potential values of counters as states, and allow a constant-size representation of many counting constraints imposed on a sequence of integer variables. Moreover their compositional nature permits representing a conjunction of two sequence constraints as the intersection of the corresponding counter automata [33,32], i.e. the intersection of the languages accepted by all counter automata, without representing explicitly the Cartesian product of all counter values. As a consequence, the size of such an intersection counter automaton is often quite compact, even if maintaining domain consistency for such constraints is in general NP-hard [13]; for instance, the intersection of the 22 counter automata for all NB_σ time-series constraints described in [3] has only 16 states.

To formally analyse the quality of the generated invariants we developed a method allowing us to verify whether a linear invariant is a facet of the convex hull or not. The method identifies two distinct points located on the line corresponding to the linear invariant, and shows that these points are always feasible provided the precondition associated with the invariant holds.

For synthesising disjunctions of inequalities, we use a slightly different approach, comprising three steps: data generation, mining of invariants, and proof of invariants. The proof part is based on the idea that, in order to prove that there is no sequence satisfying a conjunction of conditions, we can represent a set of sequences satisfying each condition by a constant-size automaton without counters. Then, a sequence satisfying all the conditions must be accepted by the intersection of such automata. If the intersection is empty, then such a sequence does not exist.
The contributions of this article are:

- First, Section 4 provides the basis of a simple, systematic method to precompute linear inequalities and conditional linear inequalities for a conjunction of two AUTOMATON constraints on the same sequence. We call such inequalities and implications *linear invariants* and *conditional linear invariants*, respectively. Each linear invariant and each conditional linear invariant involves the result variables of the different AUTOMATON constraints in a considered conjunction representing the fact that the result variables cannot vary independently. Such invariants may be parametrised by a function of the sequence length and are independent of the domains of the sequence variables. Finally, we describe a systematic method for verifying whether a linear invariant is a facet of the convex hull or not.

– Second, Section 5 shows how to obtain disjunctions of inequalities, possibly parameterised by the sequence length. We call such disjunctions *non-linear invariants*.
– Third, to mechanise all proofs required in Section 4 for proving that a linear invariant is facet defining, and in Section 5 for proving non-linear invariants, Section 6 defines a special kind of constant-size automaton without counters, named *conditional automata* that recognises all (and only all) sequences satisfying some condition, e.g. all sequences maximising the number of peaks. It shows how to construct such conditional automata in a systematic way.
– Fourth, within the context of time-series constraints, Section 7 shows the impact of the database of 2000 synthesised invariants on the propagation of time-series constraints on short-term electricity production problems.

Note that all obtained parameterised invariants are logical formulae involving linear inequalities, whose variables correspond to time-series characteristics that are *always true*. Hence they are computed once and for all in a preprocessing phase, put into a *database of parameterised invariants*, and consulted every time when required: there is no need to rerun our methods for synthesising invariants for every instance. While our methods for generating parameterised invariants are exponential, it turns out that (1) the generation for all the $\frac{35\times(35-1)}{2}$ pairs of time series constraints we consider in this article could be performed (once) overnight, (2) generating invariants for a conjunction of only two constraints had already a significant impact on our benchmark involving conjunction of 35 time-series constraints.

Adding redundant constraints to a constraint model has been recognised from the very beginning of Constraint Programming as a major source of improvement [23]. Attempts to generate such implied constraints in a systematic way were limited (1) by the difficulty to manually prove a large number of conjectures [29,11], (2) by the limitations of automatic proof systems [27,19], or (3) to special cases for very few constraints like ALLDIFFERENT, CARDINALITY, ELEMENT [31,1,30]. Within the context of counter automata, linear invariants relating consecutive counter values of the same constraint were obtained [24] using Farkas's lemma [16] in a resource-intensive procedure.

2 Background

This section presents the necessary background and notation on regular expressions, counter automata, and time-series constraints. Two complementary facets of time-series constraints will be presented: first, their declarative definition, second the transducers used to synthesise an implementation of time-series constraints. These transducers will be used in Section 6 to generate a constant-size automaton associated with an upper bound minus a constant shift of a time-series constraint.

2.1 Background on Regular Expressions and Counter Automata

For a regular expression σ, its language [22] is denoted by $\mathcal{L}_\sigma$. The *size* [5] of a regular expression σ, denoted by ω_σ, is the number of letters in the shortest word of $\mathcal{L}_\sigma$.

A *counter automaton* [8] $\mathcal{M}$ with $p > 0$ counters is a tuple $\langle Q, \Sigma, \delta, q_0, I, A, \alpha\rangle$, where Q is the set of *states*, Σ is the *input alphabet*, $\delta\colon (Q \times \mathbb{Z}^p) \times \Sigma \to Q \times \mathbb{Z}^p$ is the *transition function*, $q_0 \in Q$ is the *initial state*, I is a sequence of length p of the initial values of the p counters, $A \subseteq Q$ is the *set of accepting states*, and $\alpha\colon \mathbb{Z}^p \to \mathbb{Z}^k$ is a function, called *acceptance function*, which maps the counters of an accepting state into k integers. If, by consuming the symbols of a word w in Σ^*, the automaton $\mathcal{M}$ triggers a sequence of transitions from q_0, its initial state, to some accepting state where $\langle d_1, d_2, \ldots, d_p\rangle$ are the values of the counters at this stage, then $\mathcal{M}$ returns $\alpha(d_1, d_2, \ldots, d_p)$, otherwise it *fails*. In this article, the input alphabet of the counter automata is $\{\text{'<'}, \text{'='}, \text{'>'}\}$.

Within all figures, the acceptance function is depicted by a box connected by dotted lines to each state. If a counter is left unchanged while triggering a given transition, then we do not mention this counter update on the corresponding transition.

The *intersection* of two counter automata $\mathcal{M}_1$ and $\mathcal{M}_2$ is a counter automaton, denoted by $\mathcal{I}$, such that the following conditions holds:

1. The language of $\mathcal{I}$ is the intersection of the languages of $\mathcal{M}_1$ and $\mathcal{M}_2$.
2. The number of counters of $\mathcal{I}$ is equal to $p_1 + p_2$, where p_1 (resp. p_2) is the number of counters of $\mathcal{M}_1$ (resp. $\mathcal{M}_2$).
3. When consuming any input signature S, for every counter $C_{i,j}$ of $\mathcal{I}$, at every transition its value is equal to the value of the counter j of $\mathcal{M}_i$ when consuming S.
4. For every input signature S, the counter automaton $\mathcal{I}$ returns a tuple $\langle R_{1,1}, R_{1,2}, \ldots, R_{k_1+k_2}\rangle$, where $R_1, R_2, \ldots, R_{k_1}$ (resp. $R_{k_1+1}, R_{k_1+2}, \ldots, R_{k_1+k2}$) are the values returned by $\mathcal{M}_1$ (resp. $\mathcal{M}_2$).

2.2 Defining Time-Series Constraints

Given an integer sequence $X = \langle X_1, X_2, \ldots, X_n\rangle$, a *time-series constraint* $g\_f\_\sigma(X, R)$, introduced in [9], restricts a time-series characteristics R to be the result of some computations over an integer sequence $X = \langle X_1, X_2, \ldots, X_n\rangle$, where:

- σ is a regular expression [22] over the alphabet $\Sigma = \{\text{'<'}, \text{'='}, \text{'>'}\}$ with which we associate two integer constants b_σ and a_σ whole role is explained below; the sequence $S = \langle S_1, S_2, \ldots, S_{n-1}\rangle$, called the *signature* and containing *signature symbols*, is linked to the sequence X via the *signature conditions* $(X_i < X_{i+1} \Leftrightarrow S_i = \text{'<'}) \wedge (X_i = X_{i+1} \Leftrightarrow S_i = \text{'='}) \wedge (X_i > X_{i+1} \Leftrightarrow S_i = \text{'>'})$ for all $i \in [1, n-1]$ [8,36]. When $\langle S_i, S_{i+1}, \ldots, S_j\rangle$ (with $1 \leq i \leq j \leq n$) is a maximal word matching σ, the sequence $\langle X_{i+b_\sigma}, X_{i+b_\sigma+1}, \ldots, X_{j+1-a_\sigma}\rangle$ is called a *σ-pattern*;
- f is a function over sequences, called *feature*, and is used for computing a value for each σ-pattern; the role of the two constants b_σ and a_σ is to trim the left and right borders of an occurrence of the regular expression σ when computing the feature values;
- g is a function over sequences, called *aggregator*, and is used for aggregating the feature values of the different σ-patterns.

The result value R of a time-series constraints is restricted to be the result of aggregation, computed using g, of the list of values of feature f for all σ-patterns in X. In this article, we consider the following class of time-series constraints.

Definition 1 (value-independent time-series constraints) A time-series constraints $g\_f\_\sigma(X, R)$ is *value independent* if any two integer sequences with the same signature yield the same value of R.

We denote by $\mathbb{S}$ the class of all value independent time-series constraints. In the rest of the article, we only consider time-series constraints in $\mathbb{S}$, namely the SUM_ONE_$\sigma(X, R)$ and the SUM_WIDTH_$\sigma(X, R)$ families:

- For SUM_ONE_σ, the feature `one` denotes the constant function 1, and the aggregator `sum` is a sum. Consequently R is the number of σ-patterns of X. In the following we use NB_σ as a shorthand for SUM_ONE_σ.
- For SUM_WIDTH_σ, the feature `width` denotes the number of elements in a σ-pattern. Then R is the sum of the number of elements of all σ-patterns of X.

If there is no σ-pattern in X, then R is the default value of g, which is 0 in the case of the `sum` aggregator. The *length* of an integer sequence is the number of its elements. In the following, we assume non-empty integer sequences.

Example 1 Consider the PEAK = '$<(<\mid=)^*(>\mid=)^*>$' and the VALLEY = '$>(>\mid=)^*(<\mid=)^*<$' regular expressions with the values b_{PEAK}, a_{PEAK}, b_{VALLEY} and a_{VALLEY} all being 1. The signature of $X = \langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$ is $S = \langle <, <, =, >, =, <, >\rangle$. There is one maximal occurrence of the VALLEY regular expression in S, namely '>=<'. There are two maximal occurrences of the PEAK regular expression

in S, namely '<<=>' and '<>'. Hence, NB_PEAK$(X, 2)$ holds. The PEAK-pattern $\langle 1, 2, 2\rangle$ (resp. $\langle 4\rangle$) corresponds to the first (resp. second) maximal occurrence of PEAK in S. The width of the first and the second PEAK-patterns of X, is, respectively, 3 and 1. The sum of the widths of all PEAK-patterns of X is $3 + 1 = 4$. Hence, SUM_WIDTH_PEAK$(X, 4)$ holds. △

2.3 Operational View of Time-Series Constraints

Both, to identify all σ-patterns of an integer sequence X and to synthesise a counter automaton computing the result R of a time-series constraint $g\_f\_\sigma(X, R)$, the notion of *seed transducer* was introduced in [9]. It was shown in [25] how to generate such seed transducer from a regular expression. For the purpose of this article, we consider a simplified version of seed transducers of [9,25] that we now present.

A *seed transducer* of σ is a deterministic transducer where each transition is labelled with two letters: a letter in the input alphabet $\Sigma = \{\text{'<'}, \text{'='}, \text{'>'}\}$, called the *input symbols*, and a letter in the output alphabet $\Omega = \{\texttt{found}, \texttt{not\_found}\}$, called the *output symbols*. Hence, a transducer consumes the signature S of an integer sequence X and produces an output sequence T where each element is in Ω. Every element of Ω is called a *phase letter* and corresponds to a recognition phase of a new occurrence of σ in S. Consider different possibilities of the produced symbol T_i when consuming a symbol S_i of S:

- T_i is `found`. A transition labelled by this output symbol corresponds to the discovery of a new occurrence of σ in S.
- T_i is `not_found`. Such transitions do not correspond to the discovery of a new occurrence of σ in S, but rather to some intermediate phases that do not need to be detailed for the purpose of this article.

A transition labelled with `found` is called a `found`-*transition*. A `found`-*path* is any sequence of consecutive transitions of the transducer containing at least one `found`-transition.

Example 2 Consider the PEAK regular expression introduced in Example 1, and its seed transducer given in Part (A) of Figure 9:

- the transition from r to t is a single `found`-transition,
- the sequence of transitions from s to r, from r to t and from t to r is a `found`-path.

While consuming the signature $S = \langle <, <, =, >, =, <, >\rangle$ of the integer sequence $\langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$, the seed transducer produces the output sequence $\langle$`not_found`, `not_found`, `not_found`, `found`, `not_found`, `found`$\rangle$. As shown in Example 1, S contains two maximal occurrences of PEAK, complying with the two `found` letters in t. △

3 Types of Synthesised Invariants

Consider a conjunction of two time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$ imposed on the same sequence of integer variables $X = \langle X_1, X_2, \ldots, X_n\rangle$. In this section, we present a classification of different types of invariants that involves R_1, R_2 and n.

Farkas Linear Invariants for a Single Constraint The method for generating linear invariants based on the Farkas's lemma was described in [24], and is used for generating linear invariants linking the counters of a counter automaton representing a single constraint γ_i with i in $\{1, 2\}$. Such linear invariants involve k consecutive counter values associated with consecutive prefixes of X. For instance, for the counter automaton of the NB_PEAK constraint given in Part (A) of Figure 1, the method described in [24] generates the linear invariants $P_i - P_{i-1} \geq 0$ and $-P_i - P_{i-2} + 1 \geq 0$, where P_{i-k} corresponds to the number of peaks (an increase followed by a decrease) in the sequence $X_1, X_2, \ldots, X_{i-k+1}$ ($k \in [0, 2]$).

Although, this method is fairly general, the generation of invariants can be time consuming and the set of generated invariants is too large. This requires an extra step for selecting the tightest generated invariants.

Linear Invariants for a Conjunction of Constraints A contribution of this article is a systematic method for generating linear invariants of the form $a \cdot R_1 + b \cdot R_2 + c \cdot n + d \leq 0$ ($a, b \in \mathbb{Z}^*$ and $c, d \in \mathbb{Z}$) linking the result variables R_1 and R_2 of two time-series constraints. This method applies for any conjunction of constraints, where each constraint can be represented by a counter automaton, satisfying a certain property, named the incremental-automaton property, which will be introduced in Property 1 of Section 4. The class of automata satisfying the incremental-automaton property is smaller compared to the ones satisfying the conditions of the method of [24]. However, it still covers 35 constraints of the volume II of the Global Constraint Catalogue [3]: this covers two classes of constraints, namely constraints counting the number of occurrences of a pattern, and constraints returning the number of positions belonging to a pattern occurrence. We further show in a systematic way that many of the generated invariants are facets of the convex hull of feasible combinations of R_1 and R_2.

Conditional Linear Invariants for a Conjunction of Constraints We also generate conditional linear invariants of one of the following forms $R_1 > 0 \Rightarrow \ell in$, $R_2 > 0 \Rightarrow \ell in$, $R_1 > 0 \wedge R_2 > 0 \Rightarrow \ell in$, $n > e \Rightarrow \ell in$ with ℓin corresponding to $a \cdot R_1 + b \cdot R_2 + c \cdot n + d \leq 0$ and $a, b \in \mathbb{Z}^*$, $c, d \in \mathbb{Z}$ and $e \in \mathbb{N}$. Such invariants are useful when, for example, *(i)* a linear invariant holds only for non-default values of R_1, i.e. when there is no occurrence of the regular expression associated with γ_1 in the signature of X, *(ii)* a linear invariant is a facet of the convex hull and holds only for long enough sequences. The method for generating such invariants is based on the method for synthesising linear invariants, and the same conditions on counter automata apply.

Non-Linear Invariants The non-linear invariants we synthetise are of the form $P_1 \vee P_2 \vee \cdots \vee P_k$, where every P_k is a negation of an *atomic relation*. We define in Section 5 a set of 8 atomic relations, some of which are $R_i = c$, $R_i = \mathrm{up}_{R_i}(n) - c$, where c is a natural number, and $\mathrm{up}_{R_i}(n)$ is the maximum value of R_i among all time series of length n [5]. Such invariants are required when the envelope of the set of feasible combinations of R_1 and R_2 is non-convex and therefore linear invariants are not enough for fully describing it.

4 Synthesising Linear Invariants

Consider two counter automata $\mathcal{M}_1, \mathcal{M}_2$ over the same alphabet Σ. Let r_i denote the number of counters of $\mathcal{M}_i$, and let R_i designate its returned value. In this section we show how to systematically generate linear invariants of the form

$$e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i \geq 0 \quad \text{with} \quad e, e_0, e_1, e_2 \in \mathbb{Z}, \tag{1}$$

which hold after the signature of the same input sequence $\langle X_1, X_2, \ldots, X_n \rangle$ is completely consumed by the two counter automata $\mathcal{M}_1$, $\mathcal{M}_2$. We call such linear invariant *general* since it holds regardless of any conditions on the result variables R_1, R_2. Stronger, but less general, invariants may be obtained when the initial values of the counters cannot be assigned to the result variables.

Our method for generating invariants is applicable to a restricted class of counter automata that we now introduce.

Property 1 (incremental-automaton property) A counter automaton $\mathcal{M}$ with r counters has the *incremental-automaton* property if the following four conditions are all satisfied:

1. For every counter A_j of $\mathcal{M}$, its initial value α_j^0 is a natural number.
2. For every counter A_j of $\mathcal{M}$ and for every transition t of $\mathcal{M}$, the update of A_j upon triggering transition t is of the form $A_j \leftarrow \alpha_{j,0}^t + \sum_{i=1}^{r} \alpha_{j,i}^t \cdot A_i$, with $\alpha_{j,0}^t \in \mathbb{N}$ and $\alpha_{j,1}^t, \alpha_{j,2}^t, \ldots, \alpha_{j,r}^t \in \{0, 1\}$.

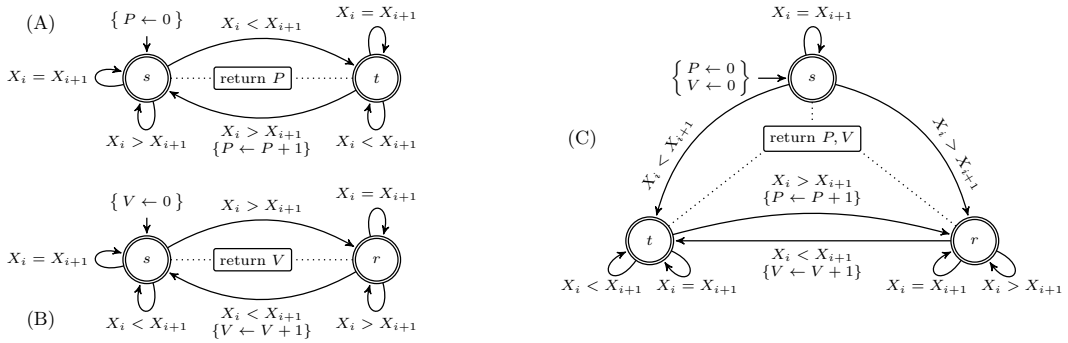


Fig. 1: (A) Counter automaton for NB_PEAK; (B) Counter automaton for NB_VALLEY; (C) Intersection of (A) and (B).

3. The counter A_r is called the *main counter* and verifies all the following three conditions:
   (a) the value returned by $\mathcal{M}$ is the last value of its main counter A_r,
   (b) for every transition t of $\mathcal{M}$, $\alpha^t_{r,r} = 1$,
   (c) for a non-empty subset T of transitions of $\mathcal{M}$, $\sum_{i=1}^{r-1} \alpha^t_{r,i} > 0,\ \forall t \in T$.
4. For all other counters A_j with $j < r$, on every transition t of $\mathcal{M}$, we have $\sum_{i=1, i\neq j}^{r} \alpha^t_{j,i} = 0$ and, if $\alpha^t_{r,j} > 0$, then $\alpha^t_{j,j}$ is 0.

The intuition behind the incremental-automaton property is that there is one counter that we name the *main counter*, whose last value is the final value, returned by the counter automaton, (see 3a). At some transitions, the update of the main counter is a linear combination of the other counters, while on the other transitions its value either does not change or is incremented by a non-negative constant, (see 3b and 3c). All other counters may only be incremented by a non-negative constant or assigned to some non-negative integer value, and they *may* contribute to the final value, (see 4). These counters are called *potential counters*. Both counter automata in Parts (A) and (B) of Figure 1 have the incremental-automaton property, and their single counters are the main counters. Volumes I and II of the global constraint catalogue contain more than 50 such counter automata. In particular, in Volume II, the counter automata for all the constraints of the NB_σ and the SUM_WIDTH_σ families have the incremental-automaton property. In the rest of this article we assume that all counter automata $\mathcal{M}_1$, $\mathcal{M}_2$ have the incremental-automaton property.

Our approach for systematically generating linear invariants of type $e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i \geq 0$ considers each combination of signs of the coefficients e_i (with $i \in [0, 2]$). It consists of two steps:

1. Select the coefficients e_0, e_1, e_2, called the *relative coefficients* of the linear invariant, so that there exists a constant C such that $e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i \geq C$ (see Section 4.1 and Section 4.2).
2. Compute C and set the coefficient e, called the *constant term* of the linear invariant, to $-C$ (see Section 4.3).

The previous steps are performed as follows:

1. First, we assume a sign for each coefficient e_i (with $i \in [0, 2]$), which tells whether we have to consider or not the contribution of the potential counters; note that each combination of signs of the coefficients e_i (with $i \in [0, 2]$) will lead to a different linear invariant. Then, from the intersection $\mathcal{I}$ of $\mathcal{M}_1$, $\mathcal{M}_2$, we construct a digraph called the *invariant digraph*, where each transition t of $\mathcal{I}$ is replaced

by an arc whose weight represents the lower bound of the variation of the term $e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$ while triggering t.

2. Second, we find the coefficients e_i (with $i \in [0, 2]$) so that the invariant digraph does not contain any negative cycles. When the invariant digraph has no negative cycles, the value of $e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$ is bounded from below for any integer sequence.
3. Third, to obtain C we compute the shortest path in the invariant digraph from the node of the invariant digraph corresponding to the initial state of $\mathcal{I}$, to all nodes corresponding to accepting states of $\mathcal{I}$.

4.1 Constructing the Invariant Digraph for a Conjunction of AUTOMATON Constraints wrt a Linear Function

First, Definition 2 introduces the notion of *invariant digraph* $G_{\mathcal{I}}^{v}$ of the counter automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$ wrt a linear function v involving the values returned by these counter automata. Second, Definition 3 introduces the notion of *weight of an accepting sequence* X wrt $\mathcal{I}$ in $G_{\mathcal{I}}^{v}$, which makes the link between a path in $G_{\mathcal{I}}^{v}$ and the vector of values returned by $\mathcal{I}$ after consuming the signature of X. Finally, Theorem 1 shows that the weight of X in $G_{\mathcal{I}}^{v}$ is a lower bound on the linear function v.

Definition 2 (invariant digraph) Consider an accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt the counter automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$, and a linear function $v = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$, where (R_1, R_2) is the vector of values returned by $\mathcal{I}$ after consuming the signature of X. The *invariant digraph* of $\mathcal{I}$ wrt v, denoted by $G_{\mathcal{I}}^{v}$, is a weighted digraph defined in the following way:

- The set of nodes of $G_{\mathcal{I}}^{v}$ is the set of states of $\mathcal{I}$.
- The set of arcs of $G_{\mathcal{I}}^{v}$ is the set of transitions of $\mathcal{I}$, where for every transition t, the corresponding symbol of the alphabet is replaced by an integer weight, which is $e_0 + \sum_{i=1}^{2} e_i \cdot \beta_i^t$, where β_i^t is defined as follows:

$$\beta_i^t = \begin{cases} \alpha_{i,r_i,0}^t & \text{if } e_i \geq 0, \quad (2) \\ \sum_{j=1}^{r_i} \alpha_{i,j,0}^t & \text{if } e_i < 0, \quad (3) \end{cases}$$

  where r_i denotes the number of counters of $\mathcal{M}_i$, and $\alpha_{i,p,0}^t$ (with $p \in [1, r_i]$) is the constant in the update of the counter of $\mathcal{I}$ corresponding to the counter p of $\mathcal{M}_i$.

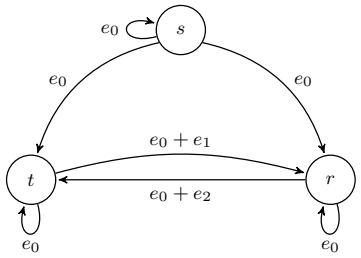
Definition 3 (walk and weight of an accepting sequence) Consider an accepting sequence X of length n wrt the counter automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$, and a linear function $v = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$, where (R_1, R_2) is the vector of values returned by $\mathcal{I}$ after consuming the signature of X.

- The *walk of X in $G_{\mathcal{I}}^{v}$* is a path in $G_{\mathcal{I}}^{v}$ whose sequence of arcs is the sequence of the corresponding transitions of $\mathcal{I}$ triggered upon consuming the signature of X.
- The *weight of X in $G_{\mathcal{I}}^{v}$* is the weight of its path in $G_{\mathcal{I}}^{v}$ plus a constant value, which is a lower bound on v corresponding to the initial values of the counters and is called the *initialisation weight* in $G_{\mathcal{I}}^{v}$. It equals $e + e_0 \cdot (p-1) + \sum_{i=1}^{2} e_i \cdot \beta_i^0$, where p is the arity of the signature, and where β_i^0 is defined as follows:

$$\beta_i^0 = \begin{cases} \alpha_{i,r_i}^0 & \text{if } e_i \geq 0, \quad (4) \\ \sum_{j=1}^{r_i} \alpha_{i,j}^0 & \text{if } e_i < 0, \quad (5) \end{cases}$$

where r_i denotes the number of counters of $\mathcal{M}_i$, and $\alpha_{i,p}^0$ (with $p \in [1, r_i]$) is the initial value of the counter of $\mathcal{I}$ corresponding to the counter p of $\mathcal{M}_i$.

Example 3 Consider the NB_PEAK(X, P) and the NB_VALLEY(X, V) constraints introduced in Example 1 on the same sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$. Figure 1 gives the automata for NB_PEAK, NB_VALLEY, and their intersection $\mathcal{I}$. We aim to find inequalities of the form $e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V \geq 0$ that hold for every integer sequence X. After consuming the signature of X, $\mathcal{I}$ returns a pair of values (P, V), which are the number of peaks (resp. valleys) in X. The invariant digraph of $\mathcal{I}$ wrt $v = e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V$ is given in the figure on the right. As neither of the two automata has any potential counters, the weights of the arcs of $G_{\mathcal{I}}^v$ do not depend on the signs of e_1 and e_2. Hence, for every integer sequence X, its weight in $G_{\mathcal{I}}^v$ equals $e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V$. △



Theorem 1 (lower bound on the weight of an accepting sequence) *Consider an accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt the counter automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$, and a linear function $v = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$, where (R_1, R_2) is the vector of values returned by $\mathcal{I}$. Then, the weight of X in $G_{\mathcal{I}}^v$ is less than or equal to $e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$.*

Proof Since, when doing the intersection of counter automata we do not merge counters, the counters of $\mathcal{I}$ that come from different counter automata do not interact, i.e. their updates are independent, hence their returned values are also independent. By definition of the invariant digraph, the weight of any of its arc is $e_0 + \sum_{i=1}^{2} e_i \cdot \beta_i^t$, where β_i^t depends on the sign of e_i, and where t is the corresponding transition in $\mathcal{I}$. Then, the weight of X in $G_{\mathcal{I}}^v$ is the constant $e + e_0 \cdot (p-1) + \sum_{i=1}^{2} e_i \cdot \beta_i^0$ (see Definition 3) plus the weight of the walk of X, which is in total $e + e_0 \cdot (p-1) + \sum_{i=1}^{2} e_i \cdot \beta_i^0 + e_0 \cdot (n-p+1) + \sum_{j=1}^{n-p+1} \sum_{i=1}^{2} e_i \cdot \beta_i^{t_j} = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot \left(\beta_i^0 + \sum_{j=1}^{n-p+1} \beta_i^{t_j} \right)$, where p is the arity of the considered signature, and $t_1, t_2, \ldots t_{n-p+1}$ is the sequence of transitions of $\mathcal{I}$ triggered upon consuming the signature of X. We now show that the value $e_i \cdot \left(\beta_i^0 + \sum_{j=1}^{n-p+1} \beta_i^{t_j} \right)$ is not greater than $e_i \cdot R_i$. This will imply that the weight of the walk of X in $G_{\mathcal{I}}^v$ is less than or equal to $v = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$.

Consider the $v_i = e_i \cdot R_i$ linear function. We show that the weight of X in $G_{\mathcal{I}}^{v_i}$, which equals $e_i \cdot \left(\beta_i^0 + \sum_{j=1}^{n-p+1} \beta_i^{t_j} \right)$, is less than or equal to $e_i \cdot R_i$. Depending on the sign of e_i we consider two cases.

Case 1: $\mathbf{e_i \geq 0}$. In this case, the weight of every arc of $G_{\mathcal{I}}^{v_i}$ is e_i multiplied by $\alpha_{r_i,0}^t$, where t is the corresponding transition in $\mathcal{I}$, and r_i is the main counter of $\mathcal{M}_i$ (see Case 2 of Definition 2). If, on transition t, some potential counters of $\mathcal{M}_i$ are incremented by a positive constant, the real contribution of the counter updates on this transition to R_i is at least $\alpha_{r_i,0}^t$ since $e_i \geq 0$. The same reasoning applies to the contribution of the initial values of the potential counters to the final value R_i. Since this contribution is non-negative, it is ignored, and $\beta_i^0 = \alpha_j^0$ (see Case 2 of Definition 3). Hence $e_i \cdot \left(\beta_i^0 + \sum_{j=1}^{n-p+1} \beta_i^{t_j}\right) = e_i \cdot \left(\alpha_{r_i}^0 + \sum_{j=1}^{n-p+1} \alpha_{r_i,0}^t\right) \leq e_i \cdot R_i$.

NEW VERSION:

In this case, the weight of every arc of $G_{\mathcal{I}}^{v_i}$ is e_i multiplied by $\alpha_{i,r_i,0}^t$, where t is the corresponding transition in $\mathcal{I}$, and r_i is the main counter of $\mathcal{M}_i$ (see Case 2 of Definition 2). If, on transition t, some potential counters of $\mathcal{M}_i$ are incremented by a positive constant, the real contribution of the counter updates on this transition to R_i is at least $\alpha_{i,r_i,0}^t$ since $e_i \geq 0$. The same reasoning applies to the contribution of the initial values of the potential counters to the final value R_i. Since this contribution is non-negative, it is ignored, and $\beta_i^0 = \alpha_{i,r_i}^0$ (see Case 2 of Definition 3). Hence $e_i \cdot \left(\beta_i^0 + \sum_{j=1}^{n-p+1} \beta_i^{t_j}\right) = e_i \cdot \left(\alpha_{i,r_i}^0 + \sum_{j=1}^{n-p+1} \alpha_{i,r_i,0}^{t_j}\right) \leq e_i \cdot R_i$.

Case 2: $\mathbf{e_i < 0}$. In this case, the weight of every arc of $G_{\mathcal{I}}^{v_i}$ is e_i multiplied by the sum of the non-negative constants, which come from the updates of *every* counter of $\mathcal{M}_i$ (see Case 5 of Definition 2). The contribution of the potential counters is always taken into account, and since $e_i < 0$, it is always negative. The same reasoning applies to the contribution of the initial values of the potential counters to the returned value R_i. To obtain a lower bound on v, observe that the initial values of the potential counters are non-negative and that $e_i < 0$; therefore we assume that the initial values of the potential counters always contribute to R_i (see Case 3 of Definition 3). Hence $e_i \cdot (\beta_i^0 + \sum_{j=1}^{n-p+1} \beta_i^{t_j}) \leq e_i \cdot R_i$. □

Note that, if all the considered counter automata $\mathcal{M}_1$, $\mathcal{M}_2$ do not have potential counters, then for every accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$ and for any linear function $v = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$, the weight of X in $G_{\mathcal{I}}^v$ is *equal* to v. If there is at least one potential counter for at least one counter automaton $\mathcal{M}_i$, then there may exist an accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$ whose weight in $G_{\mathcal{I}}^v$ is strictly less than v.

4.2 Finding the Relative Coefficients of the Linear Invariant

We now focus on finding the relative coefficients e_0, e_1, e_2 of the linear invariant $v = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i \geq 0$ such that, after consuming the signature of any accepting sequence by the counter automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$, the value of v is non-negative.

For any accepting sequence X wrt $\mathcal{I}$, by Theorem 1, we have that the weight w of X in $G_{\mathcal{I}}^v$ is less than or equal to v. Recall that w consists of a constant part, and of a part that depends on X, which involves the coefficients e_0, e_1, e_2; thus, these coefficients must be chosen in a way that there exists a constant C such that $w \geq C$, and C does not depend on X. This is only possible when $G_{\mathcal{I}}^v$ does not contain *any* negative cycles. Let $\mathcal{C}$ denote the set of all simple circuits of $G_{\mathcal{I}}^v$, and let w_e denote the weight of an arc e of $G_{\mathcal{I}}^v$. In order to prevent negative cycles in $G_{\mathcal{I}}^v$, we solve the following minimisation problem, parameterised by (s_0, s_1, s_2), the signs of e_0, e_1, e_2, under the convention that the sign -1

represents a non-positive number, and the sign $+1$ corresponds to a non-negative number:

$$\text{minimise} \quad \sum_{c \in \mathcal{C}} W_c + \sum_{i=1}^{2} |e_i| \tag{6}$$

$$\text{subject to} \; W_c = \sum_{e \in c} w_e \qquad \forall c \in \mathcal{C} \tag{7}$$

$$W_c \geq 0 \qquad \forall c \in \mathcal{C} \tag{8}$$

$$s_i \cdot e_i \geq 0 \qquad \forall i \in [0, 2] \tag{9}$$

$$e_i \neq 0 \qquad \forall i \in [1, 2] \tag{10}$$

In order to obtain the coefficients e_0, e_1, e_2 so that $G^v_{\mathcal{I}}$ does not contain any negative cycles, it is enough to find a solution to the satisfaction problem (7)-(10). Minimisation is required to obtain linear invariants that eliminate as many infeasible values of (R_1, R_2) as possible. Within the objective function (6), the term $\sum_{c \in \mathcal{C}} W_c$ is for minimising the weight of every simple circuit, while the term $\sum_{i=1}^{2} |e_i|$ is for obtaining the coefficients with the smallest absolute value. We empirically established that using the considered objective function for time-series constraints produces linear invariants of good quality. However, for a different class of constraints a different objective function may be more appropriate. By changing the sign vector (s_0, s_1, s_2) we obtain different linear invariants.

Example 4 (finding the relative coefficients) Consider NB_PEAK(X, P) and NB_VALLEY(X, V) with X being a time series of length n. The invariant digraph of the intersection of the counter automata for the NB_PEAK and NB_VALLEY constraints wrt $v = e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V$ was given in Example 3. This digraph has four simple circuits, namely $s - s$, $t - t$, $r - r$, and $r - t - r$, which are labelled by 1, 2, 3 and 4, respectively. Then, the minimisation problem for finding the relative coefficients of the linear invariant $v \geq 0$, parameterised by (s_0, s_1, s_2), the signs of e_0, e_1 and e_2, is the following:

$$\begin{aligned} \text{minimise} \quad & \sum_{j=1}^{4} W_j + \sum_{i=0}^{2} |e_i| \\ \text{subject to} \quad & W_j = e_0, && \forall j \in [1, 3] \\ & W_4 = e_0 + e_1 + e_2 \\ & W_j \geq 0 && \forall j \in [1, 4] \\ & s_i \cdot e_i \geq 0 && \forall i \in [0, 2] \\ & e_i \neq 0 && \forall i \in [1, 2] \end{aligned} \tag{11}$$

Note that the value of e_0 must be non-negative otherwise (11) cannot be satisfied for $j \in \{1, 2, 3\}$. Hence we consider only the combinations of signs of the form $(\text{`+'}, s_1, s_2)$ with s_1 and s_2 being either '$-$' or '$+$'. The following table gives the optimal solution of the minimisation problem for the considered combinations of signs:

(s_0, s_1, s_2)	$(+, -, -)$	$(+, -, +)$	$(+, +, -)$	$(+, +, +)$
(e_0, e_1, e_2)	$(1, -1, -1)$	$(0, -1, 1)$	$(0, 1, -1)$	$(0, 1, 1)$

△

4.3 Finding the Constant Term of the Linear Invariant

Finally, we focus on finding the constant term e of the linear invariant $v = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i \geq 0$, when the coefficients e_0, e_1, e_2 are known, and when the digraph of the counter automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$

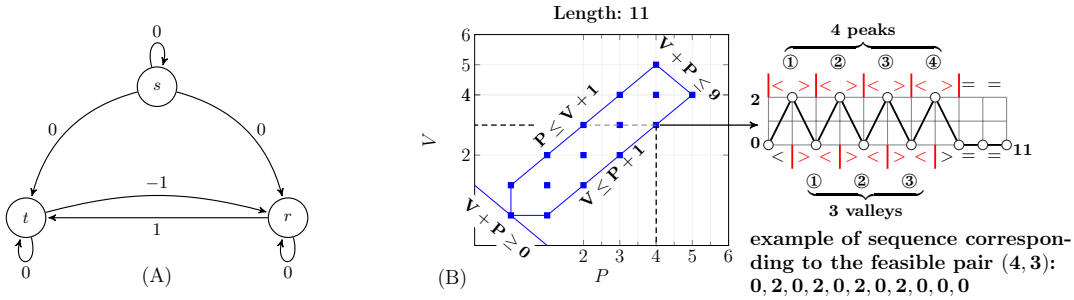


Fig. 2: (A) The invariant digraph of the counter automata for the NB_PEAK and the NB_VALLEY time-series constraints; (B) The set of feasible values of the result variables P and V of the NB_PEAK and the NB_VALLEY time-series constraints, respectively, for sequences of length 11.

wrt v does not contain any negative cycles. By Theorem 1, the weight of any accepting sequence X wrt $\mathcal{I}$ in $G^v_{\mathcal{I}}$ is less than or equal to v, then if the weight of X is non-negative, it implies that v is also non-negative. Since the invariant digraph $G^v_{\mathcal{I}}$ does not contain any negative cycles, then the weight of X cannot be smaller than some constant C. Hence it suffices to find this constant and set the constant term e to $-C$. The value of C is computed as the constant $e_0 \cdot (p-1) - \sum_{i=1}^{2} \beta_i^0$ (see Definition 3) plus the shortest path length from the node of $G^v_{\mathcal{I}}$ corresponding to the initial state of $\mathcal{I}$ to all the nodes of $G^v_{\mathcal{I}}$ corresponding to the accepting states of $\mathcal{I}$.

Example 5 (obtaining invariants) Consider NB_PEAK(X, P) and NB_VALLEY(X, V) with X being a time series of length n such that $n \geq 2$. In Example 4, we found four vectors for the relative coefficients e_0, e_1, e_2 of the linear invariant $e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V \geq 0$. For every found vector for the relative coefficients (e_0, e_1, e_2), we obtain a weighted digraph, whose weights now are integer numbers. For example, for the vector $(e_0, e_1, e_2) = (0, -1, 1)$, the obtained digraph is given in Part (A) of Figure 2. We compute the length of the shortest path from the node s, which corresponds to the initial state of the counter automaton in Part (C) of Figure 1 to every node corresponding to an accepting state of the counter automaton in Part (C) of Figure 1. The length of the shortest path from s to s is 0, from s to t is 0, and from s to r is -1. The minimum of these values is -1, hence the constant term e equals $-(0 + (-1)) = 1$. The obtained linear invariant is $P \leq V + 1$.

In a similar way, we find the constant terms for the other found vectors of the relative coefficients (e_0, e_1, e_2), and obtain three other linear invariants: $V \leq P + 1$, $V + P \leq n - 2$, $V + P \geq 0$.

Part (B) of Figure 2 shows the polytope of feasible points (P, V) when n is 11. Observe that three of the four linear invariants found are facets of the convex hull of this polytope. △

4.4 Improving the Generated Linear Invariants

When at least one of the counter automata $\mathcal{M}_1$, $\mathcal{M}_2$ has at least one potential counter, then there may exist an accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$ such that the weight of X in the invariant digraph $G^v_{\mathcal{I}}$ is strictly less than $v = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$. This may lead to weaker invariants and Example 6 illustrates such a situation.

Example 6 (weak invariant) Given the proper plateau regular expression '>=$^+$<', consider a conjunction of NB_PROPER_PLATEAU(X, R_1) and SUM_WIDTH _PROPER_PLATEAU(X, R_2) imposed on the same time series X of length n, and a linear function $v = e + e_0 \cdot n + e_1 \cdot R_1 + e_2 \cdot R_2$. The intersection of the counter automata for these two constraints is given in Part (A) of Figure 3. By inspection we can

derive the invariant $R_2 \geq 2 \cdot R_1$, which cannot be generated by the method described in Sections 4.1, 4.2 and 4.3, because of the following reason: when $e_0 = 0$, $e_1 = -2$, and $e_2 = 1$, the weights of the arcs from a to b and from b to c are both e_0, and the weight of the arcs from c to a is $e_0 + e_1 + e_2$, and thus the weight of the cycle $a - b - c - a$ is $3 \cdot e_0 + e_1 + e_2 = -1$.

Just before triggering the transition from c to a, the value of the counter D_2 is at least 1 since the counter automaton had triggered the transition from b to c before, which incremented D_2. Let us modify the intersection $\mathcal{I}$ so that the counter D_2 is not updated on the transition from b to c, and the counter R_2 is updated as $R_2 + D_2 + 2$ on the transition from c to a. The modified counter automaton $\mathcal{I}^*$ recognises the same set of signatures as $\mathcal{I}$, and after consuming any accepting sequence wrt $\mathcal{I}$, the counter automaton $\mathcal{I}^*$ returns the same tuple of final values as $\mathcal{I}$. In addition, the weight of the cycle $a-b-c-a$ in $\mathcal{I}^*$ is equal to $3 \cdot e_0 + e_1 + 2 \cdot e_2$, which is 0 when $e_0 = 0$, $e_1 = -2$, and $e_2 = 1$. Hence, the invariant $R_2 \geq 2 \cdot R_1$ can be generated after some modifications of the intersection $\mathcal{I}$. △

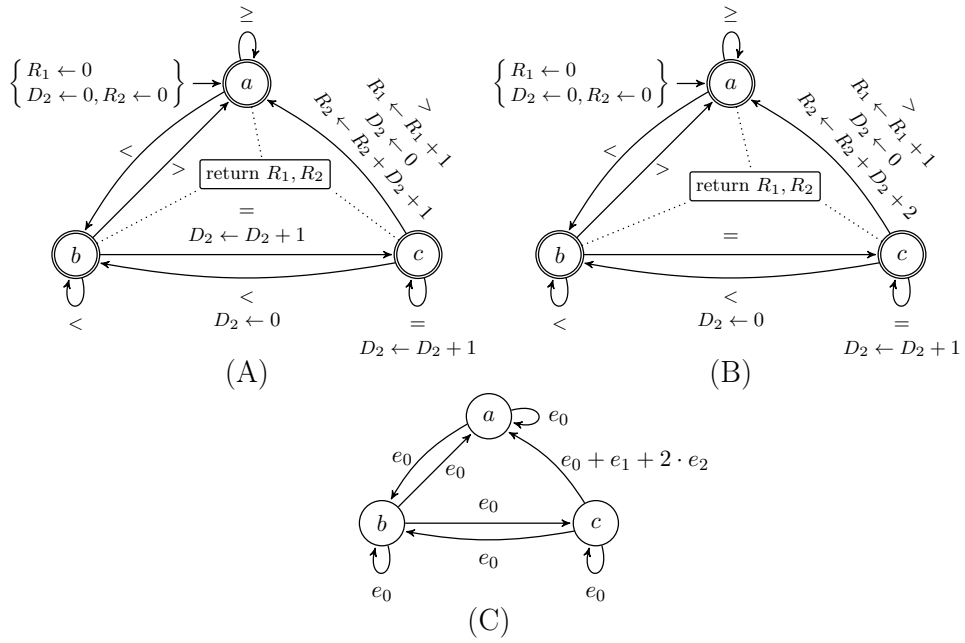


Fig. 3: (A) Intersection of counter automata for NB_PROPER_PLATEAU and SUM_WIDTH_PROPER_PLATEAU, for which the method described in Sections 4.1, 4.2 and 4.3 does not generate facet-defining invariants; (B) Delayed intersection obtained from the intersection in (A); (C) Invariant digraph obtained from the delayed intersection in (B).

To handle the issue presented in Example 6 we introduce a *preprocessing technique* of the intersection of counter automata. The technique relies on the notion of *delay* of a potential counter A at a state q of the intersection $\mathcal{I}$, which is a lower bound on the value of A when a sequence of triggered transitions of the counter automaton ends up in state q. Intuitively, we can change the updates of some counters in a way that for any accepting sequence wrt $\mathcal{I}$, the returned tuple of values does not change, but the arcs of the invariant digraph obtained from the modified intersection $\mathcal{I}^*$ will have larger weights. The modified intersection that we obtain satisfies the three following conditions:

1. The set of accepting sequences wrt $\mathcal{I}$ coincides with the set of accepting sequences wrt $\mathcal{I}^*$.

2. For every accepting sequence X wrt $\mathcal{I}$, the counter automata $\mathcal{I}$ and $\mathcal{I}^*$ return the same tuple of values.
3. For any accepting sequence X, the weight of X in $G^v_{\mathcal{I}^*}$ is greater than or equal to the weight of X in $G^v_{\mathcal{I}}$, where v is $e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$.

By Condition 3, since for every X, the weight of X in $G^v_{\mathcal{I}^*}$ is greater than or equal to the weight of X in $G^v_{\mathcal{I}}$, the weight of every simple circuit in X may also increase, which may lead to stronger invariants. To obtain such counter automaton $\mathcal{I}^*$, we first introduce in Definition 4 the notion of *list of delays* of a state q of the intersection $\mathcal{I}$, denoted by d_q. An element i of d_q is an array whose values correspond to the potential counters of $\mathcal{M}_i$. The value j of this array represents a lower bound on the value of the counter of $\mathcal{I}$ corresponding the potential counter j of $\mathcal{M}_i$ when the counter automaton $\mathcal{I}$ arrives to the state q. Further, based on this notion, in Definition 5, we introduce the notion of *delayed intersection*. Finally, in Theorem 2 we show that the delayed intersection satisfies Conditions 1, 2, and 3.

Definition 4 (list of delays of a state) Consider a counter automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$. The *list of delays* d_q of a state q is a list of arrays, where the size of the i-th array in d_q is the number of potential counters in the counter automaton $\mathcal{M}_i$. Let j be the index of a counter of $\mathcal{M}_i$, let $\mathcal{T}_q$ denote the set of transitions entering q, and $\mathcal{T'}_q$ denote a subset of transitions of $\mathcal{T}_q$ starting from a state different from q, then the value $d_q[i][j]$ is defined as

$$d_q[i][j] = \begin{cases} 0 & \exists t \in \mathcal{T}_q,\ \alpha^t_{i,j,j} = 0, \\ \min(\alpha^0_{i,j}, \min\limits_{t \in \mathcal{T'}_q} \alpha^t_{i,j,0}) & q \text{ is the initial state of } \mathcal{I}, \text{ and } \forall t \in \mathcal{T'}_q,\ \alpha^t_{i,j,j} > 0, \\ \min\limits_{t \in \mathcal{T'}_q} \alpha^t_{i,j,0} & \text{otherwise,} \end{cases}$$

where $\alpha^t_{i,j,j}$ (resp. $\alpha^t_{i,j,0}$) denotes the coefficient of the counter A_j (resp. the free term) in the update of A_j in the automaton $\mathcal{M}_i$.

Example 7 (list of delays of a state) Consider two counter automata $\mathcal{M}_1$ and $\mathcal{M}_2$ such that their intersection $\mathcal{I}$ is given in Part (A) of Figure 3. The counter automaton $\mathcal{M}_1$ has one counter R_1, and $\mathcal{M}_2$ has two counters D_2 and R_2. Let us compute the list of delays of every state of $\mathcal{I}$. Since only $\mathcal{M}_1$ does not have any potential counters then for any state q of $\mathcal{I}$, the array $d_q[1]$ is empty. The following table gives the list of delays of every potential counter of $\mathcal{I}$.

state	a	b	c
d_q	[[], [0]]	[[], [0]]	[[], [1]]

It implies that, when the counter automaton $\mathcal{I}$ is either in state a or state b, we only know that its potential counter D_2 is non-negative. However, when $\mathcal{I}$ is in the state c, the value of its potential counter is at least 1. △

Definition 5 (delayed intersection) Consider the counter automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$. The delayed intersection $\mathcal{I}^*$ of $\mathcal{M}_1$, $\mathcal{M}_2$ is obtained from $\mathcal{I}$ using the following rules:

- The set of states and accepting states of $\mathcal{I}^*$ coincide with those of $\mathcal{I}$.
- The set of transitions of $\mathcal{I}^*$ coincide with the one of $\mathcal{I}$.
- The number of counters of $\mathcal{I}^*$ is the same as for $\mathcal{I}^*$, and is denoted by r.
- The initial values of main counters of $\mathcal{I}^*$ are the same as for $\mathcal{I}^*$. For every potential counter $A^*_{i,j}$ of $\mathcal{I}^*$, its initial value equals $\alpha^0_{i,j} - d_q[i][j]$, where q is the initial state of $\mathcal{I}^*$ and $\alpha^0_{i,j}$ is the initial value of $A_{i,j}$ of $\mathcal{I}$.

- For every transition t from a state q_1 to a state q_2 and for any counter $\mathcal{M}_{i,j}$ of $\mathcal{I}$, the update of $A_{i,j}$ on t is equal to $\alpha^t_{i,j,0} + \sum_{k=1}^{r} \alpha^t_{i,j,k} \cdot A_{i,k}$, while the update of the corresponding counter $\mathcal{M}^*_{i,j}$ on the corresponding transition of $\mathcal{I}^*$ is equal to $\gamma^t_{i,j,0} + \sum_{k=1}^{r} \alpha^t_{i,j,k} \cdot A^*_{i,k}$, where $\gamma^t_{i,j,0}$ is defined as follows:
  - If $A_{i,j}$ is a main counter of $\mathcal{I}$, then $\gamma^t_{i,j,0} = \alpha^t_{i,j,0} + \sum_{k=1}^{r_i-1} \alpha^t_{i,j,k} \cdot d_{q_1}[i][k]$, where r_i is the number of counters of the counter automaton $\mathcal{M}_i$.
  - If $A_{i,j}$ is a potential counter of $\mathcal{I}$, then $\gamma^t_{i,j,0} = \alpha^t_{i,j,0} + d_{q_1}[i][j] - d_{q_2}[i][j]$.
- The acceptance function of $\mathcal{I}^*$ is the same as for $\mathcal{I}$.

Example 8 (delayed intersection) Consider two counter automata $\mathcal{M}_1$ and $\mathcal{M}_2$ such that their intersection $\mathcal{I}$ is given in Part (A) of Figure 3. The delayed intersection $\mathcal{I}^*$ constructed according to Definition 5 is given in Part (B) of Figure 3. The main difference between $\mathcal{I}^*$ and $\mathcal{I}$ is that the counter D_2 is no longer updated on the transition from b to c, but its contribution is integrated directly to R_2 on the transition from state c to state a. △

Theorem 2 (properties of delayed intersection) *Consider the counter automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2$ and the corresponding delayed intersection $\mathcal{I}^*$. The three following conditions are satisfied:*

1. *The set of accepting sequence wrt $\mathcal{I}$ coincides with the set of accepting sequence wrt $\mathcal{I}^*$.*
2. *For every accepting sequence X wrt $\mathcal{I}$, the counter automata $\mathcal{I}$ and $\mathcal{I}^*$ return the same tuple of values.*
3. *For any accepting sequence X, the weight of X in $G^v_{\mathcal{I}^*}$ is greater than or equal to the weight of X in $G^v_{\mathcal{I}}$, where v is $e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$.*

Proof We prove each of the three statements separately.

[Proof of (1)]. Since $\mathcal{I}$ have the same sets of states, transitions and accepting states, and every $\mathcal{M}_i$ has the incremental-automaton property, then the sets of accepting sequences of $\mathcal{I}$ and $\mathcal{I}^*$ are the same.

[Proof of (2)]. Since the acceptance function of both $\mathcal{I}$ and $\mathcal{I}^*$ returns a tuple of main counters, we will show that after consuming the signature S of any accepting sequence, the main counters of $\mathcal{I}$ and $\mathcal{I}^*$ contain the same values. Let us prove this statement by induction on the length of S.

Base case. Let us consider a sequence $S = \langle S_1 \rangle$ consumed by $\mathcal{I}^*$. The counter automaton $\mathcal{I}^*$ triggered one transition t from its initial state q to some other state q'. Then, let us consider a main counter A^*_{i,r_i}. By definition, its value equals $\alpha^t_{i,j,0} + A^*_{i,r_i,r_i} + \sum_{k=1}^{r_i-1} \alpha^t_{i,j,k} \cdot (A^*_{i,k} + d_q[i][k])$. Since any potential counter $A^*_{i,k}$ has not been updated, its contains the initial value, which equals $\alpha^0_{i,j} - d_q[i][k]$. Furthermore, the value of A^*_{i,r_i} after one transition is equal to $\alpha^t_{i,j,0} + \alpha^0_{i,r_i} + \sum_{k=1}^{r_i-1} \alpha^t_{i,j,k} \cdot (\alpha^0_{i,j} - d_q[i][k] + d_q[i][k]) = \alpha^t_{i,j,0} + \alpha^0_{i,r_i} + \sum_{k=1}^{r_i-1} \alpha^t_{i,j,k} \cdot \alpha^0_{i,j}$, which coincides with the value of the corresponding counter $A_{i,j}$ of $\mathcal{I}$.

Induction step. Assume that after having consumed a sequence $S = \langle S_1, S_2, \ldots, S_{m-1} \rangle$, the main counters of $\mathcal{I}^*$ contain the same values as the main counter of $\mathcal{I}$ after having consumed the same sequence. Let us show that after consuming one another symbol S_m, which triggers a transition t, the main counters of $\mathcal{I}^*$ and $\mathcal{I}$ will have the same value. The update of A^*_{i,r_i} on t is equal to $\alpha^t_{i,j,0} + A^*_{i,r_i} + \sum_{k=1}^{r_i-1} \alpha^t_{i,j,k} \cdot (A^*_{i,k} + d_q[i][k])$. By the induction hypothesis the value of A^*_{i,r_i} in $\mathcal{I}$ and A_{i,r_i} in $\mathcal{I}^*$ are the same after consuming S. Hence, we only need to show after having consumed S, that the value of the

potential counter $A_{i,k}$ of $\mathcal{I}$ equals $A^*_{i,k} + d_q[i][k]$. This can also be shown by induction, starting from a state that is a destination of a triggered transition t' such that $\alpha^{t'}_{i,k,k} = 0$.

[Proof of (3)]. We now prove the last statement. Let us consider the invariant digraphs $G^v_{\mathcal{I}^*}$ and $G^v_{\mathcal{I}}$, where $v = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot R_i$. We now show that for every accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I}$, its weight in $G^v_{\mathcal{I}^*}$ is greater than or equal to its weight in $G^v_{\mathcal{I}}$. The weight of X in $G^v_{\mathcal{I}}$ is the constant $e + e_0 \cdot (p-1) + \sum_{i=1}^{2} e_i \cdot \beta^0_i$ (see Definition 3) plus the weight of the walk of X, which is in total $e + e_0 \cdot (p-1) + \sum_{i=1}^{2} e_i \cdot \beta^0_i + e_0 \cdot (n-p+1) + \sum_{j=1}^{n-p+1} \sum_{i=1}^{2} e_i \cdot \beta^{t_j}_i = e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot \left(\beta^0_i + \sum_{j=1}^{n-p+1} \beta^{t_j}_i \right)$, where p is the arity of the considered signature, and $t_1, t_2, \ldots t_{n-p+1}$ is the sequence of transitions of $\mathcal{I}$ triggered upon consuming the signature of X. Similarly, the weight of X in $G^v_{\mathcal{I}^*}$ is equal to $e + e_0 \cdot n + \sum_{i=1}^{2} e_i \cdot \left(\delta^0_i + \sum_{j=1}^{n-p+1} \delta^{t_j}_i \right)$, where δ^0_i is the initialisation weight in $\mathcal{I}^*$, and every $\delta^{t_j}_i$ is the weight of an arc t_j in $G^v_{\mathcal{I}^*}$.

We now show that the value $e_i \cdot \left(\beta^0_i + \sum_{j=1}^{n-p+1} \beta^{t_j}_i \right)$ is not greater than $e_i \cdot \left(\delta^0_i + \sum_{j=1}^{n-p+1} \delta^{t_j}_i \right)$. This will imply that the weight of the walk of X in $G^v_{\mathcal{I}}$ is less than or equal to the weight of the walk of X in $G^v_{\mathcal{I}^*}$.

By Definition 2, the weight of every arc of $G^v_{\mathcal{I}}$ (resp. $G^v_{\mathcal{I}^*}$), corresponding to a transition t of $\mathcal{I}$, (resp. $\mathcal{I}^*$) is equal to $\sum_{i=1}^{2} e_i \cdot \beta^t_i$ (resp. $\sum_{i=1}^{2} e_i \cdot \delta^t_i$).

As in Theorem 1, we consider the function $v_i = e_i \cdot R_i$. Depending on the sign of e_i we have two cases:

Case (1): $e_i \geq 0$. Then, the weight of X in $G^{v_i}_{\mathcal{I}}$ (resp. $G^{v_i}_{\mathcal{I}^*}$) is equal to $e_i \cdot \alpha$ (resp. $e_i \cdot \gamma$), where α denotes $\beta^0_i + \sum_{j=1}^{n-p+1} \beta^{t_j}_i = \sum_{k=1}^{r_i} \alpha^0_{i,k} + \sum_{\ell=1}^{n-p+1} \alpha^{t_\ell}_{i,r_i,0}$ (resp. γ denotes $\delta^0_i + \sum_{j=1}^{n-p+1} \delta^{t_j}_i = \sum_{k=1}^{r_i} \gamma^0_{i,k} + \sum_{\ell=1}^{n-p+1} \gamma^{t_\ell}_{i,r_i,0}$). Since every $\gamma^{t_\ell}_{i,r_i,0} = \alpha^{t_\ell}_{i,r_i,0} + \sum_{k=1}^{r_i-1} d_q[i][k]$, it implies that $\gamma^{t_\ell}_{i,r_i,0} \geq \alpha^{t_\ell}_{i,r_i,0}$. Then, $\alpha \leq \gamma$, and when $e_i > 0$, we have $e_i \cdot \gamma \geq e_i \cdot \alpha$.

Case (2): $e_i < 0$. Then, the weight of X in $G^{v_i}_{\mathcal{I}}$ (resp. $G^{v_i}_{\mathcal{I}^*}$) is equal to $e_i \cdot \alpha$ (resp. $e_i \cdot \gamma$), where α denotes $\beta^0_i + \sum_{j=1}^{n-p+1} \beta^{t_j}_i = \sum_{k=1}^{r_i} \alpha^0_{i,k} + \sum_{\ell=1}^{n-p+1} \sum_{k=1}^{r_i} \alpha^{t_\ell}_{i,k,0}$ (resp. γ denotes $\delta^0_i + \sum_{j=1}^{n-p+1} \delta^{t_j}_i = \sum_{k=1}^{r_i} \gamma^0_{i,k} + \sum_{\ell=1}^{n-p+1} \sum_{k=1}^{r_i} \gamma^{t_\ell}_{i,k,0}$). Further, by construction of $\mathcal{I}^*$, every $\gamma^{t_\ell}_{i,k,0}$ (with $i \in [1, r_i]$) is equal to $\alpha^{t_\ell}_{i,k,0} + d_{q_1}[i][k] - d_{q_2}[i][k]$, where q_1 and q_2 are the source and the destination of the transition t_ℓ, respectively. In addition, $\gamma^{t_\ell}_{i,r_i,0} = \alpha^{t_\ell}_{i,r_i,0}$. By replacing every $\gamma^{t_\ell}_{i,k,0}$ with its expression, and simplifying the sum, we obtain $\sum_{k=1}^{r_i} \alpha^0_{i,k} + \sum_{\ell=1}^{n-p+1} \sum_{k=1}^{r_i} (\alpha^{t_\ell}_{i,k,0} - d_{q'}[i][k])$, where q' is the last state visited by $\mathcal{I}$ upon consuming X. Since every $d_{q'}[i][k]$ is non-negative, $\alpha^{t_\ell}_{i,k,0} - d_{q'}[i][k] \leq \alpha^{t_\ell}_{i,k,0}$. This implies that $\gamma \leq \alpha$, and when $e_i < 0$, $e_i \cdot \gamma \geq e_i \cdot \alpha$. □

Note that in the counter automaton $\mathcal{I}^*$, all the constants $\gamma^t_{i,j,0}$ introduced in Definition 5 are non-negative by definition of the delay (see Definition 4). It means that the reasoning used in the proof of Theorem 1 requiring the non-negativity of these constants remains valid for the invariant digraph $G^v_{\mathcal{I}^*}$.

Example 9 (generating stronger invariants) Consider two counter automata $\mathcal{M}_1$ and $\mathcal{M}_2$ such that their intersection $\mathcal{I}$, and their delayed intersection $\mathcal{I}^*$ are respectively given in Parts (A) and (B) of Figure 3. The invariant digraph $G^v_{\mathcal{I}^*}$ is given in Part (C) of Figure 3 when $e_0 > 0$, $e_1 > 0$, and $e_2 < 0$. By stating the minimisation problem from Section 4.2, we obtain the following coefficients: $e_0 = 0$, $e_1 = -2$, and $e_2 = 1$. The constant e is found to be 0, and we obtain the invariant $2 \cdot R_1 \leq R_2$, which could not be found with the invariant digraph $G^v_{\mathcal{I}}$. △

4.5 Generating Conditional Linear Invariants with the Non-Default Value Condition

Quite often a counter automaton $\mathcal{M}_i$ (with $i \in [1, 2]$) returns the default value only when the signature of X does not contain any occurrence of some regular expression σ_i. For example, for NB_PEAK(X, R), the default value of R is 0. The corresponding automaton will return 0 iff X does not contain any peaks. This may lead to a convex hull of points of coordinates (R_1, R_2) returned by $\mathcal{I}$ containing infeasible integer points, e.g. see Part (A) of Figure 4. Some of these infeasible points can be eliminated by stronger invariants subject to a condition, called the *non-default value* condition, that the final value returned by the counter automaton is not the default value of the corresponding constraint. We first illustrate the motivation for such conditional linear invariants.

Example 10 (motivation for conditional invariants) Consider the NB_DECREASING_TERRACE(X, R_1) and the SUM_WIDTH_INCREASING_TERRACE(X, R_2) constraints, where X is a time series of length n, R_1 is restricted to be the number of maximal occurrences of DECREASING_TERRACE = '>=$^+$>' in the signature of X, and R_2 is restricted to be the sum of the number of elements in subseries of X whose signatures correspond to words of the language of INCREASING_TERRACE = '<=$^+$<'. In Figure 4, for $n = 12$, the squared points represent feasible pairs (R_1, R_2), while the circled points stand for infeasible pairs (R_1, R_2) inside the convex hull. The linear invariant $2 \cdot R_1 + R_2 \leq n - 2$ is a facet of the polytope, which does not eliminate the points $(1, 8)$, $(2, 6)$, $(3, 4)$, $(4, 2)$. However, if we assume that both $R_1 > 0$ and $R_2 > 0$, then we can add a linear invariant eliminating these four infeasible points, namely $2 \cdot R_1 + R_2 \leq n - 3$, shown in Part (B) of Figure 4. In addition, the infeasible points on the straight line $R_2 = 1$ will also be eliminated by the restriction $R_2 = 0 \vee R_2 \geq 2$ given in [3, p. 2962]. △

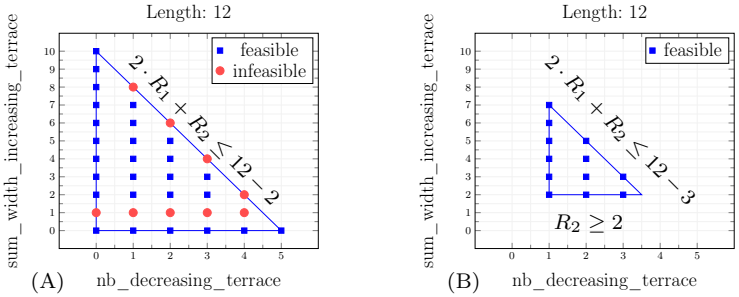


Fig. 4: Invariants on the result values R_1 and R_2 of NB_DECREASING_TERRACE and SUM_WIDTH_INCREASING_TERRACE for a sequence length of 12 (A) with the general linear invariants, and (B) with the Non-Default Value condition.

We now outline how to adapt our method of generating linear invariants in case the result values of the constraints of interest cannot be the corresponding default values. Consider that each counter automaton $\mathcal{M}_i$ (with $i \in [1, 2]$) returns the default value of the corresponding constraint after consuming

the signature of an accepting sequence X wrt $\mathcal{M}_i$ iff the signature of X does not contain any occurrence of some regular expression σ_i over the alphabet Σ. Let $\mathcal{M}'_i$ denote the counter automaton which accepts the words of the language $\Sigma^*\sigma_i\Sigma^*$, where Σ^* denotes any word over Σ. Then, using the method described in Sections 4.1, 4.2 and 4.3 we generate the linear invariants for $\mathcal{M}'_1 \cap \mathcal{M}'_2$. These linear invariants hold when the non-default value condition is satisfied.

4.6 Facet Analysis of Linear Invariants

Consider two time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$ imposed on the same sequence X of length n. After having generated linear and conditional linear invariants linking R_1, R_2 and n, an essential question is whether these invariants are facets of the convex hull of feasible combinations R_1 and R_2, or not. Given a linear invariant $f = e + e_0 \cdot n + e_1 \cdot R_1 + e_2 \cdot R_2 \geq 0$ and a infinite set A of values of n such that the set of sequences whose length is in A can be represented by a constant-size automaton, this section presents a two-step method for answering the question whether this invariant is a facet of the convex hull when n is in A:

1. Find two distinct integer points P_1 and P_2, possibly parameterised by $n \in A$, laying on the straight line $f = 0$.
2. Prove that P_1 and P_2 are feasible for any $n \in A$.

The challenge here is the second step, which requires to prove the feasibility of P_1 and P_2 for an infinite set of values of n. Let $\text{up}_{R_i}(n)$ denote the maximum value of R_i among all time series of length n, let a_1, a_2 be in $\{0, 1\}$ and let b_1 and b_2 be natural numbers. It turns out that for points of the form $\begin{pmatrix} h_x, \\ h_y \end{pmatrix} = \begin{pmatrix} a_1 \cdot \text{up}_{R_1}(n) + (1 - 2 \cdot a_1) \cdot b_1, \\ a_2 \cdot \text{up}_{R_2}(n) + (1 - 2 \cdot a_2) \cdot b_2 \end{pmatrix}$ we can represent the set of time series corresponding to such a point as the intersection of three constant-size automata, namely (i) the automaton representing the assumed condition on n, (ii) the automaton that accepts only and only all time series yielding h_x as the value of R_1, and (iii) the automaton that accepts only and only all time series yielding h_y as the value of R_2. The constant-size automata representing a condition on R_1 and R_2 can be synthesised from the seed transducers for the regular expressions associated with γ_1 and γ_2, as shown in Section 6. We now give in Sections 4.6.1, 4.6.2 and 4.6.3 more details for each of the two steps, and also how to pick the set A.

4.6.1 Choosing the set A of sequence lengths

Some of the invariants we generate are facets of the convex hull only for a subset of values of n, e.g. only even-length sequences. This requires to assume a condition on n that can be represented by a constant-size automaton. We start with the less restrictive condition and try to prove that an invariant is a facet, and then gradually restrict the condition if we cannot prove it in full generality.

4.6.2 Step One: Finding Two Integer Points on a Straight Line

To find two distinct points on the straight line $f = 0$, we assume a value of R_1 as $a_1 \cdot \text{up}_{R_1}(n) + (1 - 2 \cdot a_1) \cdot b_1$, which by [5] is equal to $a_1 \cdot \frac{n - c_1 - (n - c_1) \bmod d_1}{d_1} + (1 - 2 \cdot a_1) \cdot b_1$, with c_1 and d_1 being integer constants depending on the regular expression associated with γ_1. If the coefficient of R_2 in f is 0, then the value of R_2 is not relevant and we can take, for example, 0 or 1 as the value of R_2. Otherwise, by isolating R_2 from the equation $f = 0$ we obtain:

$$R_2 = \frac{(-e_0 \cdot d_1 - e_1 \cdot a_1) \cdot n + (-e \cdot d_1 + e_1 \cdot a_1 \cdot c_1 - e_1 \cdot (1 - 2 \cdot a_1) \cdot b_1 \cdot d_1) + e_1 \cdot a_1 \cdot (n - c_1) \bmod d_1}{d_1 \cdot e_2} \quad (12)$$

Then we verify that the right-hand side of (12) is of the form $a_2 \cdot \frac{n-c_2-(n-c_2) \bmod d_2}{d_2} + (1-2\cdot a_2)\cdot b_2$, with c_2 and d_2 being integer constants depending on the regular expression associated with γ_2, with a_2 being in $\{0,1\}$, and with b_2 being a natural number. This is done by solving a system of constraints assuming that n belongs to A. The solutions of such system are the candidate points of the next step.

4.6.3 Step Two: Proving Feasibility of an Integer Point

Once we found two distinct integer points laying on the straight line $f = 0$, we show that both points are feasible for any n in A.

For a point of coordinates (h_x, h_y) we construct two constant-size automata $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\mathcal{M}_1$ (resp. $\mathcal{M}_2$) is an automaton recognising the signatures of all and only time series yielding h_x (resp. h_y) as the value of R_1 (resp. R_2). Let $\mathcal{M}_n$ be a constant-size automaton representing the $n \in A$ condition, and d denote the smallest difference between two values in A. If, in the intersection $\mathcal{M}$ of $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_n$ there are cycles of length d, then the point (h_x, h_y) is feasible for any sequence whose length is in A. From this intersection we also compute the smallest value of n, for which these two points are feasible. This is the length of the shortest path from the initial state of $\mathcal{M}$ to an accepting state of $\mathcal{M}$ that goes through a state belonging to a cycle of length d.

If we cannot prove the feasibility of our two current points, then we try a different combination of a_1 and b_1, and obtain two other distinct points. Since the set of values of b_1 is, potentially, unbounded we limit ourselves only to the values of b_1 belonging to the set $\{0,1,2,3\}$.

Example 11 Consider the conjunction of the NB_PEAK(X,P) and the NB_VALLEY(X,V) time-series constraints imposed on the same time series $X = \langle X_1, X_2, \ldots, X_n\rangle$, and the linear invariant $P+V \leq n-2$. Let us now analyse whether this invariant is facet defining or not. By [5], both $\mathrm{up}_P(n)$ and $\mathrm{up}_V(n)$ are equal to $\frac{n-1-(n-1) \bmod 2}{2}$.

- When P is equal to $\mathrm{up}_P(n)$, then by (12), V is equal to $\frac{n-3+(n-1) \bmod 2}{2}$; we consider two cases:
  i. If $(n-1) \bmod 2 = 0$, then $\frac{n-3+(n-1) \bmod 2}{2} = \frac{(n-1)-2}{2} = \mathrm{up}_V(n) - 1$.
  ii. If $(n-1) \bmod 2 = 1$, then $\frac{n-3+(n-1) \bmod 2}{2} = \frac{(n-2)-2}{2} = \mathrm{up}_V(n) - 1$.

  In both cases, we obtain the candidate point $P_1 = (\mathrm{up}_P(n), \mathrm{up}_V(n) - 1)$.
- When P is equal to $\mathrm{up}_P(n) - 1$, then by (12), V is $\frac{n-1+(n-1) \bmod 2}{2}$; we consider two cases:
  i. If $(n-1) \bmod 2 = 0$, then $\frac{n-1+(n-1) \bmod 2}{2} = \frac{n-1}{2} = \mathrm{up}_V(n)$ and we obtain the candidate point $P_2 = (\mathrm{up}_P(n) - 1, \mathrm{up}_V(n))$.
  ii. If $(n-1) \bmod 2 = 1$, then $\frac{n-1+(n-1) \bmod 2}{2} = \frac{(n-2)+2}{2} = \mathrm{up}_V(n) + 1$ and we obtain the candidate $(\mathrm{up}_P(n) - 1, \mathrm{up}_V(n) + 1)$. This candidate is not feasible since its second coordinate is strictly greater than the maximum value of the second coordinate of any feasible point.

Hence, for the case $(n-1) \bmod 2 = 0$, we obtain two distinct candidate points P_1 and P_2 located on the straight line $P + V = n - 2$. To prove that $P_2 = (\mathrm{up}_P(n) - 1, \mathrm{up}_V(n))$ is feasible, we construct and intersect the automata for the $R_1 = \mathrm{up}_P(n)$, $R_2 = \mathrm{up}_V(n) - 1$, and $(n-1) \bmod 2 = 0$ conditions, and observe that the intersection has a cycle of length 2, which implied the feasibility of P_2 for any odd sequence size. The same procedure is used for proving the feasibility of P_1 for any odd sequence size.

Since both P_1 and P_2 lay on the straight line $R_1 + R_2 = n - 2$, and are feasible for any odd length, then the straight line $R_1 + R_2 = n - 2$ is a facet of the convex hull of feasible points, when n is odd. △

5 Synthesising Non-Linear Invariants

The contribution of this section is a methodology for two families of time-series constraints, namely the NB_σ and the SUM_WIDTH_σ families, which both proposes conjectures and proves them automatically by using *constant-size automata*, i.e. automata whose number of states, and whose input alphabet size are independent both from an input time-series length and from the values in an input time series.

For a conjunction of two time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$ imposed on the same time series $X = \langle X_1, X_2, \ldots, X_n \rangle$, our method describes sets of infeasible result-value pairs for (R_1, R_2). We assume that every time-series constraint mentioned in this section belongs either to the NB_σ or to the SUM_WIDTH_σ family. Each set of infeasible pairs is described by a formula $f_i(R_1, R_2, n)$ expressed as a conjunction $C_i^1 \wedge C_i^2 \wedge \ldots \wedge C_i^{k_i}$ of elementary conditions C_i^j between R_1, R_2 and n. The learned Boolean function $f_1 \vee f_2 \vee \cdots \vee f_m$ represents the union of sets of infeasible pairs (R_1, R_2), while its negation $\neg f_1 \wedge \neg f_2 \wedge \cdots \wedge \neg f_m$ corresponds to an implied constraint, which is a *universally true Boolean formula*, namely

$$\forall X,\ \gamma_1(X, R_1) \wedge \gamma_2(X, R_2) \Rightarrow \bigwedge_{i=1}^{m} \neg f_i(R_1, R_2, n) \tag{13}$$

In order to prove that (13) is universally true we need to show that for every $f_i(R_1, R_2, n)$, there does not exist a time series of length n yielding R_1 (resp. R_2) as the result value of γ_1 (resp. γ_2) and satisfying $f_i(R_1, R_2, n)$. The key idea of our proof scheme is to represent the infinite set of time series satisfying each elementary condition C_i^j of $f_i(R_1, R_2, n)$ as a constant-size automaton $\mathcal{M}_{i,j}$. Then checking that the intersection of all automata $\mathcal{M}_{i,1}, \mathcal{M}_{i,2}, \ldots, \mathcal{M}_{i,k_i}$ is empty implies that $f_i(R_1, R_2, n)$ is indeed infeasible. Note that such proof scheme is independent of the time-series length n; moreover, it does not explore any search space.

As for the linear invariants, the generation process of non-linear invariants is offline: it is done once and for all to build a reusable database of generic invariants. This section is organised as follows:

- Section 5.1 motivates this work with a running example, which illustrates the need for deriving non-linear invariants.
- Section 5.2 presents our method for deriving non-linear invariants for a conjunction of time-series constraints. It starts with an overview of the three phases of our method, and then details each phase:
  1. A *generating data phase* is detailed in the introduction of Section 5.2. Its goal is to generate a dataset, from which we will extract non-linear invariants.
  2. A *mining phase* is detailed in Section 5.2.2. It extracts, from the data generated in the mining phase, a hypothesis H consisting of Boolean functions of the form $f_1 \vee f_2 \vee \cdots \vee f_m$.
  3. A *proof phase* is detailed in Section 5.2.3. For every Boolean function f_i (with $i \in [1, m]$) in the extracted hypothesis H, the proof phase either proves its validity for *every* time-series length, or refute it by generating a counter example. The counter example is used to modify the current hypothesis and the process is repeated.

  Note that our generated data is noise-free, and that our goal is not to discover statistical properties of time-series constraints, but rather to extract non-linear invariants, which are always true.

5.1 Motivation for Generating Non-Linear Invariants and Running Example

Consider a conjunction of time-series constraints $\gamma_1(X, R_1) \wedge \gamma_2(X, R_2)$ imposed on the same time series X. In Section 4, using the representation of γ_1 and γ_2 as *counter automata*, we presented a method for deriving linear invariants linking the values of R_1, R_2. Although, in most cases the derived inequalities were proven to be facet-defining, we observe that in some cases, even when using these invariants, the solver could still take a lot of time to find a feasible solution or to prove infeasibility. This happens because of some infeasible combinations of values of the result variables that were located *inside* the convex hull of all feasible combinations. The following example illustrates such a situation.

Example 12 (running example) Consider the conjunction of SUM_WIDTH_DECREASING_SEQUENCE(X, R_1) and SUM_WIDTH_ZIGZAG(X, R_2) time-series constraints imposed on the same time series X of length n, where a decreasing sequence and a zigzag respectively correspond to '$(> (> \mid =)^*)^* >$' and '$(<>)^+ < (> \mid \varepsilon) \mid (><)^+ > (< \mid \varepsilon)$'. For the values of n in the interval $[9, 12]$, Figure 5 represents

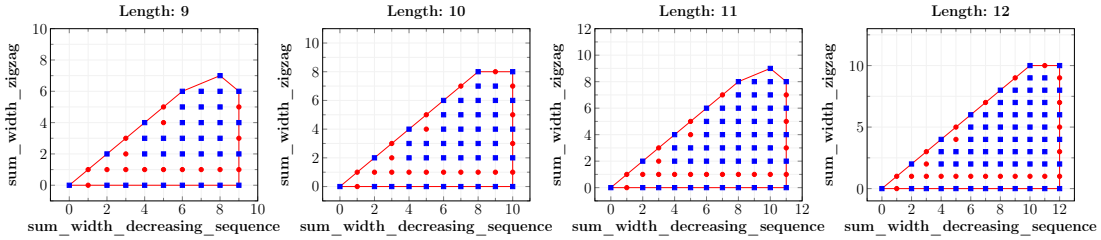


Fig. 5: Feasible points, shown as blue squares, for the result variables R_1, R_2 of the conjunction of SUM_WIDTH_DECREASING_SEQUENCE(X, R_1) and SUM_WIDTH_ZIGZAG(X, R_2) on the same time series $X = \langle X_1, X_2, \ldots, X_n \rangle$ for the values of n in $\{9, 10, 11, 12\}$; red circles represent infeasible points inside the convex hull of feasible points, while red straight lines depict the facets of the convex hull of feasible points.

feasible pairs of (R_1, R_2) as blue squares, and infeasible pairs lying inside the convex hull of feasible (blue) points as red circles. The convex hull contains a significant number of infeasible (red) points, which we want to characterise automatically. △

Next section develops a systematic approach for generating non-linear invariants characterising infeasible combinations of R_1 and R_2 located within the convex hull of feasible combinations.

5.2 Discovering and Proving Invariants

Consider a conjunction of time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$ imposed on the same time series X. This work focuses on *automatically extracting and proving* invariants that characterise some subsets of infeasible combinations of R_1 and R_2 that are all located inside the convex hull $\mathcal{CH}$ of feasible combinations of R_1 and R_2. Our approach uses three sequential phases.

• [GENERATING DATA PHASE] The first phase is a preparatory work, namely *generating data*, in which we generate feasible and infeasible combinations of pair R_1 and R_2 inside $\mathcal{CH}$.

• [MINING PHASE] The second phase, called the *mining phase*, consists of extracting a hypothesis H describing the set $\mathcal{I}$ of infeasible combinations of R_1 and R_2 from the generated data. We represent this hypothesis as a disjunction of Boolean functions $f_i(R_1, R_2, n)$.

• [PROOF PHASE] The third phase, called the *proof phase*, consists in refining the discovered hypothesis H by validating some Boolean functions f_i and by refuting and eliminating others using *constant-size automata*. A refined hypothesis, which is *proved* to be correct in the general case, i.e. for *any time-series length*, is called a *description* of the set $\mathcal{I}$.

5.2.1 Data Generation Phase

The data generation phase consists of:

1. Generating a set of feasible combinations of the values of R_1 and R_2 for different time-series lengths.
2. Compute the convex hull of feasible points of R_1 and R_2 using Graham's scan [28] for each considered time series length k.
3. Represent our generated input data set as the union of two sets of triples $\mathcal{D}^+$ (resp. $\mathcal{D}^-$) called the set of feasible (resp. possibly infeasible) examples, such that:
   (a) For every (k, p_1, p_2) in $\mathcal{D}^+$, there exists at least one time series of length k that yields p_1 and p_2 as the values of R_1 and R_2, respectively.
   (b) For every (k, p_1, p_2) in $\mathcal{D}^-$,

i. We could not find any time series of length k that would yield p_1 and p_2 as the values of R_1 and R_2, respectively, even if such time series may exist.
ii. (p_1, p_2) is located within the convex hull of feasible combinations of R_1 and R_2.

Note that our method for generating non-linear invariants does not require to identify all infeasible points located within the convex hull of feasible combinations of R_1 and R_2. While this may lead to generating some wrong hypotheses, such incorrect hypotheses would be discarded in the proof phase.

In the context of the $\frac{35\cdot(35-1)}{2}$ constraint pairs we consider in this article, we generate *all* feasible combinations of the values of R_1 and R_2 for each time-series length n in $[7, 12]$. This can be achieved overnight on a single computer, and only needs to be performed once to generate the invariants.

5.2.2 Mining Phase

Consider a conjunction of two time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$, imposed on the same time series X. This section shows how to extract a hypothesis in the form of a *disjunction of Boolean functions*, describing the infeasible combinations of values of R_1 and R_2 that are located within the convex hull of feasible combinations.

There exist a number of works on learning a disjunction of predicates [17], and some special case, where disjunction corresponds to a geometric concept [18,20]. Usually, the learner interacts with an oracle through various types of queries or with the user by receiving positive and negative examples; the learner tries to minimise the number of such interactions to speed up convergence.

In our case, the input data consists of the set of positive, called *infeasible*, and negative, called *feasible*, examples, which is finite and which is completely produced by our generating phase. This allows exploring all possible inputs without any interaction.

We now present the components of our mining phase:

- First, we define the space of concepts, *hypotheses*, we can potentially extract from our dataset.
- Second, we outline the *target hypothesis* for time-series constraints, i.e. what we are searching for.
- Finally, we briefly describe the algorithm used for finding the target hypothesis.

Space of Hypotheses Every element of our hypothesis space is a disjunction of Boolean functions from a finite predefined set $\mathcal{H}$. Each element of $\mathcal{H}$ is a conjunction $C_1 \wedge C_2 \wedge \cdots \wedge C_p$ with every C_i being a predicate, called an *atomic relation*, where the main atomic relations are:

(i)	$n \geq c$,	(iii)	$R_j \bmod c = d$,	(v)	$R_j \leq d$,	(vii)	$R_j = \mathrm{up}_{R_j}(n) - c$,
(ii)	$n \bmod c = d$,	(iv)	$R_j \geq d$,	(vi)	$R_j = c$,	(viii)	$R_j = c \cdot R_k + d$,

with c and d being natural numbers, and $\mathrm{up}_{R_k}(n)$ being the maximum possible value of R_k given the constraint $\gamma_k(\langle X_1, X_2, \ldots, X_n \rangle, R_k)$. The intuition of these atomic relations is now explained:

- (i) stems from the fact that many invariants are only valid for long enough time series.
- (ii) is motivated by the fact that the parity of the length of a time series is sometimes relevant.
- (iii) is justified by the fact that the parity of R_1 or R_2 can come into play.
- (iv) and (v) are related to the fact that infeasible combinations of R_1 and R_2 can be located on a ray or an interval.
- (vi) and (vii) are respectively linked to the fact that quite often infeasible combinations of R_1 and R_2 within the convex hull are very close to the minimum or the maximum values [5] of R_k (with $k \in [1, 2]$), i.e. c is a very small constant, typically 0 or 1.
- (viii) denotes the fact that some invariants correspond to a linear combination of R_1 and R_2.

These atomic relations were conceived with two guidelines in mind. First the atomic relations reflect subsets of infeasible points we could observe in our generated dataset. Second, since we want to do proofs that do not depend on the sequence size, we focus on atomic relations which could be represented by a constant size automaton.

Target Hypothesis

Definition 6 (Boolean function consistent wrt a dataset) A Boolean function of $\mathcal{H}$ is *consistent* wrt a dataset $\mathcal{D}$ iff it is true for at least one infeasible example of $\mathcal{D}$, and false for every feasible example of $\mathcal{D}$.

For example, $R_1 = R_2 \wedge R_1 \bmod 2 = 1$ is consistent with the dataset of Figure 5, but the two Boolean functions $R_1 = 13$ and $R_1 = R_2$ are not.

Definition 7 (universally true Boolean function) A Boolean function of $\mathcal{H}$ is *universally true* if it is true for any time series of any length.

Definition 8 (target hypothesis) The *target hypothesis* H is the disjunction of all Boolean functions of $\mathcal{H}$ consistent with $\mathcal{D}$.

Note that in the target hypothesis some Boolean functions can be subsumed by other Boolean functions. We cannot do the subsumption analysis at this point since we do not yet know which Boolean functions are true or not.

Mining Algorithm Our mining algorithm generates possible conjunctions of the atomic relations in a bottom-up manner, filtering out those Boolean functions not consistent with our dataset. The set of atomic relations to consider is finite, as we only have to consider small integer values for constants c and d. The algorithm returns the disjunction of the remaining, consistent Boolean functions. Note that the mining algorithm ignores Boolean functions involving the atomic relation (i) $n > c$, which is handled in the proof phase. Remember that we run the algorithm only on the limited dataset $\mathcal{D}_{[7,12]}$, i.e. the dataset generated from time series of length in $[7, 12]$. This is because sizes that are too small lead to degenerate polytopes, while sizes that are too large are too expensive in terms of computation.

5.2.3 Proof Phase

After extracting from $\mathcal{D}_{[7,12]}$ the target hypothesis $H = f_1 \vee f_2 \vee \cdots \vee f_m$ characterising subsets of infeasible combinations of R_1 and R_2 that are all located within the convex hull of feasible combinations of R_1 and R_2, we refine this hypothesis, by keeping only universally true Boolean functions f_i.

Before presenting our proof technique, we look at the structure of the hypothesis H. Every Boolean function f in H is of the form $f = C_1 \wedge C_2 \wedge \cdots \wedge C_p$ and can be classified into one of the two following categories:

- Independent Boolean Function means that every C_i is an *independent atomic relation*, i.e. depends either on R_1 or R_2, but not on both. For instance, $R_1 = \mathrm{up}_{R_1}(n) \wedge R_2 \bmod 2 = 1$ is an independent Boolean function.

- Dependent Boolean Function means that there exists at least one C_i that is a *dependent atomic relation*, i.e. mentions both R_1 and R_2. For instance, $R_1 \bmod 2 = 1 \wedge R_1 = R_2 + 1$ is a dependent Boolean function.

The proof of an invariant depends on its category. We now show how to prove that an independent (resp. dependent) Boolean function is universally true.

Proof of Independent Boolean Functions Since most atomic relations are independent, i.e. cases (i) to (vii), we first focus on a necessary and sufficient condition for proving that an independent Boolean function is universally true. Such necessary and sufficient condition is given in the main result of this section, namely Theorem 3, provided that there exists constant-size automata associated with the atomic relations in f.

Definition 9 (set of supporting signatures for an atomic relation) For an atomic relation C, the *set of supporting signatures* $\mathcal{T}_C$ is the set of words in Σ^* such that, for every word in $\mathcal{T}_C$ there exists a time series satisfying C, whose signature is this word.

Definition 10 (set of supporting signatures for a Boolean function) For an independent Boolean function $f = C_1 \wedge C_2 \wedge \cdots \wedge C_p$, we define the *set of supporting signatures* $\mathcal{T}_f$ as $\bigcap_{i=1}^{p} \mathcal{T}_{C_i}$.

A Boolean function f is universally true iff it describes infeasible combinations of R_1 and R_2 for any time-series length, and thus the set $\mathcal{T}_f$ is empty.

For any atomic relation C from (i) to (vii), i.e. an independent atomic relation, the corresponding set of supporting signatures is represented as the language of a constant-size automaton $\mathcal{M}_C$. Constant size means that the number of states of this automaton does not depend on the length of the input time series. For a Boolean function $f = C_1 \wedge C_2 \wedge \cdots \wedge C_p$, $\mathcal{T}_f$ is simply the set of signatures recognised by the automaton obtained after intersecting all $\mathcal{M}_{C_i}$ (with $i \in [1, p]$). This provides a necessary and sufficient condition for proving that a Boolean function f is universally true.

Theorem 3 (necessary and sufficient condition for an independent Boolean function to be universally true) *Consider two time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$ on the same time series X, and a Boolean function $f(R_1, R_2, n) = C_1 \wedge C_2 \wedge \cdots \wedge C_p$ such that, for every C_i there exists a constant-size automaton $\mathcal{M}_{C_i}$. The function f is universally true iff the intersection of all automata for $\mathcal{M}_{C_i}$ (with $i \in [1, p]$) is empty.*

The proof of Theorem 3 follows from Definitions 9 and 10.

For some Boolean function $f = C_1 \wedge C_2 \wedge \cdots \wedge C_p$, the set $\mathcal{T}_f = \bigcap_{i=1}^{p} \mathcal{T}_{C_i}$ may not be empty, but finite. In this case, we compute the length c of the longest signature in $\mathcal{T}_f$, and obtain a new Boolean function $f' = f \wedge n \geq c + 1$. By construction, the set $\mathcal{T}_{f'}$ is empty, thus f' is universally true.

Section 6 will further show how to generate automata for independent atomic relations. Every such automaton is called a *conditional automaton*.

Proof of Dependent Boolean Functions Some dependent Boolean functions, i.e. case (viii), can be handled by adapting the technique for generating linear invariants described in Section 4.

Consider two time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$ on the same time series X. We present here a method for verifying that the dependent Boolean function $R_1 - d \cdot R_2 = 1$, with d being either 1 or 2, is universally true. Note that such Boolean function was extracted during the mining phase for 17 pairs of time-series constraints.

We prove by contradiction that the corresponding Boolean function is universally true. Our proof consists of the following steps:

1. **Assumption.** Assume that there exists a time series X such that $R_1 - d \cdot R_2 = 1$.
2. **Implication for the parity of R_1 and $d \cdot R_2$.** When $R_1 - d \cdot R_2 = 1$, then R_1 and $d \cdot R_2$ have different parity.
3. **Obtaining a contradiction.** Since R_1 and $d \cdot R_2$ must have different parity, there exists a value of b that is either 0 or 1 such that the conjunction $R_1 - d \cdot R_2 = 1 \wedge R_1 \bmod 2 = b \wedge d \cdot R_2 \bmod 2 = 1 - b$ holds. In order to prove that $R_1 - d \cdot R_2 = 1$ is infeasible, for either value of parameter b, we need to show that, either the obtained conjunction is infeasible, e.g. when $d = 2$ and b is 0, or the method of Section 4 produces a linear invariant $R_1 - d \cdot R_2 \geq c$, with c being strictly greater than 1.

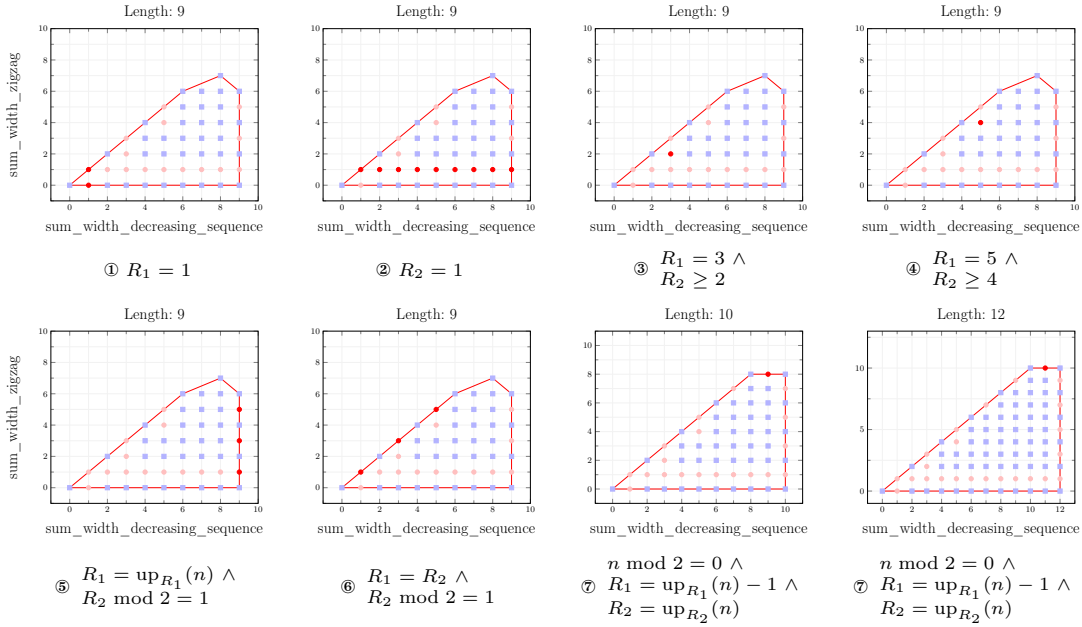


Fig. 6: Seven groups of infeasible combinations of R_1 and R_2, where R_1 and R_2 are, respectively, constrained by SUM_WIDTH_DECREASING_SEQUENCE(X, R_1) and SUM_WIDTH_ZIGZAG(X, R_2) on the same sequence X of length 9 (all plots except the two plots at the bottom right) and of lengths 10 and 12 (the two plots at the bottom right).

If at this third step of our proof method the considered conjunction is feasible, and the desired invariant $R_1 - d \cdot R_2 \geq c$ was not obtained, then we cannot draw any conclusion about the infeasibility of $R_1 - d \cdot R_2 = 1$.

In practice, for the 17 pairs of time-series constraints, for which we extracted the Boolean function $R_1 - d \cdot R_2 = 1$, the method of Section 4 did indeed generate the desired linear invariant, which proved that the considered Boolean function is universally true.

Example 13 (mining, proving and filtering non-linear invariants for the running example) Consider the conjunction of the SUM_WIDTH_DECREASING_SEQUENCE(X, R_1) and the SUM_WIDTH_ZIGZAG(X, R_2) time-series constraints on the same time series X, introduced in Example 12. For this conjunction, we now describe the result of the mining and the proving phases of our method, as well as the dominance filtering, i.e. discarding Boolean functions subsumed by some other Boolean function.

- During the mining phase we extracted a disjunction of 156 Boolean functions. Most Boolean functions, even if they are true, are redundant. For example, the Boolean function $R_1 = 1 \ \wedge \ R_2 = 1$ is subsumed by $R_1 = 1$, and thus can be discarded. However, at this point we cannot do the dominance filtering since we do not yet know which Boolean functions are universally true.
- During the proof phase we proved that 95 out of the extracted 156 Boolean functions are universally true.
- Finally, after the dominance filtering of the 95 proved Boolean functions we obtain the disjunction of the following seven Boolean functions:
  ① $R_1 = 1$, ② $R_2 = 1$, ③ $R_1 = 5 \ \wedge \ R_2 \geq 4$,
  ④ $R_1 = 3 \ \wedge \ R_2 \geq 1$, ⑤ $R_1 = \mathrm{up}_{R_1}(n) \ \wedge \ R_2 \bmod 2 = 1$, ⑥ $R_1 \bmod 2 = 1 \ \wedge \ R_1 = R_2$,
  ⑦ $n \bmod 2 = 0 \ \wedge \ R_1 = \mathrm{up}_{R_1}(n) - 1 \ \wedge \ R_2 = \mathrm{up}_{R_2}(n)$.

All four upper plots and the two lower plots on the left of Figure 6 contain the groups of infeasible combinations of R_1 and R_2 corresponding to the Boolean functions from ① to ⑥ for n being 9. The two lower plots on the right of Figure 6 contain the infeasible combinations of R_1 and R_2 corresponding to the ⑦ Boolean function for n being 10 and 12, respectively.

The Boolean functions from ① to ⑤ and ⑦ were proved by intersecting the automata for the atomic relations in these Boolean functions, and check that it was empty.

In order to prove the dependent Boolean function ⑥, we consider the conjunction of three constraints, namely $R_1 \bmod 2 = 1$, SUM_WIDTH_DECREASING_SEQUENCE, and SUM_WIDTH_ZIGZAG. Each of the three constraints can be represented by an automaton or by a counter automaton satisfying the required properties of the method of Section 4, which generates for this conjunction the invariant $R_1 \geq R_2 + 2$. This proves that ⑥ is a universally true Boolean function.

We now give an interpretation of five of those Boolean functions:

- ① and ② means that, in the languages of DECREASING_SEQUENCE and ZIGZAG, respectively, there is no word consisting of one letter.
- ⑤ means that, when a time series yields $\text{up}_{R_1}(n)$ as the value of R_1, every occurrence of ZIGZAG in its signature must start and end with '>', and the length of every word in the language of ZIGZAG starting and ending with the same letter is even.
- ⑥ is related to the fact that every word in the language of ZIGZAG contains at least one word of the language of DECREASING_SEQUENCE as a factor, and every such factor is of even length.
- ⑦ means that, when a time series yields $\text{up}_{R_2}(n)$ as the value of R_2, then its signature is a word in the language of ZIGZAG, and every occurrence of DECREASING_SEQUENCE is of even length, and thus R_1 must be even. At the same time, $\text{up}_{R_1}(n) - 1 = n - 1$ is odd, when n is even. △

6 Synthesising Conditional Automata

For the time-series constraints considered in this work we need to generate constant-size finite automata representing a certain condition, e.g. an automaton recognising the signatures of all and only all time series with the maximum number of peaks. Such automata are required for proving non-linear invariants parameterised by the time-series length, described in Section 5, and also for the facet analysis of linear invariants, described in Section 4.6. This section shows how to synthesise a constant-size automaton, i.e. an automaton whose number of states is independent, both from the input time-series length and from the values in an input time series, accepting the signatures of all, and only all, time series satisfying atomic relations of Section 5.2.2. For brevity, we only consider the atomic relation (vii) $R = \text{up}_R(n) - d$, where R is constrained by some time-series constraint $\gamma(\langle X_1, X_2, \ldots, X_n\rangle, R)$, with γ being NB_σ or SUM_WIDTH_σ, and where $\text{up}_R(n)$ is the maximum possible value of R yielded by a time series of length n. This atomic relation is indeed the most difficult case for generating a constant-size automaton. The construction associated with other atomic relations are described in [2]. We start with an illustrative example.

Example 14 (automaton for a gap atomic relation) Consider the NB_PEAK$(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint and a gap atomic relation C defined by $R = \text{up}_R(n)$. We showed in [5] that the maximum value of R for a given time-series length n is $\max\left(0, \left\lfloor \frac{n-1}{2} \right\rfloor\right)$. Hence, the automaton for C must recognise the signatures of all and only time series yielding $\max\left(0, \left\lfloor \frac{n-1}{2} \right\rfloor\right)$ as the value of R.

Part (A) of Figure 7 gives the minimal automaton accepting the set of signatures reaching this upper bound, while Part (B) lists all words of length 4 and 5 over the alphabet {'<','=','>'} having the maximum number of peaks, 2 in this case, that can be obtained from the corresponding automaton. △

The rest of this section is organised as follows:

- **[Gap Automaton]** In the context of time-series constraints of the form NB_σ or SUM_WIDTH_σ, Section 6.1 first introduces the notion of *gap of a time series* X, which indicates how far apart the

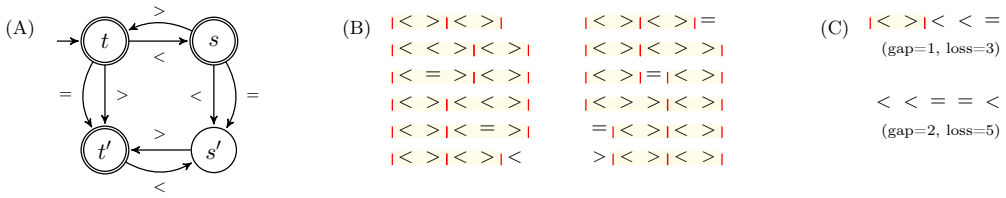


Fig. 7: (A) Automaton achieving the maximum number of peaks in a time series of length n, i.e. $\max(0, \lfloor \frac{n-1}{2} \rfloor)$, and (B) all corresponding accepted words for $n-1 \in \{4,5\}$, where each peak is surrounded by two vertical bars, and is highlighted in yellow. (C) The signatures of time series with gap 1 and 2, and with loss 3 and 5.

result value of a time-series constraint yielded by X is from the given upper bound; it then presents the *main contribution* of this section, namely, the notion of *δ-gap automaton* for a time-series constraint, i.e. a constant-size automaton that only accepts integer sequences whose gap is δ. Second, it gives a sufficient condition on the time-series constraint for the existence of such an automaton. Third, it describes how to synthesise such δ-gap automaton.

1. Section 6.1.1 introduces an intermediate notion, the *loss of a time series* wrt a time-series constraint, which is the maximum difference between the length of this time series and the length of the shortest time series yielding the same result value of a time-series constraint. For example, all words of length 4 (resp. 5) in Part (B) of Figure 7 are the signatures of time series whose gap is 0 and whose loss is 0 (resp. 1). Part (C) of Figure 7 gives two signatures of time series with gap (resp. loss) 1 and 2 (resp. 3 and 5).
   Finally, it introduces the notion of *loss automaton*, i.e. a counter automaton used to compute the loss. How to synthesise a loss automaton will be explained in Section 6.2.
2. Section 6.1.2 introduces a sufficient condition in the form of a conjunction of four conditions on a time-series constraint, called *principal conditions* that, when satisfied, guarantee the existence of the δ-gap automaton.

   – When the first three principal conditions hold, describing the set of time series whose gap is δ is equivalent to describing the set of time series whose loss belongs to a certain interval, depending on δ.
   – When the fourth principal condition holds, there exists a loss automaton whose counters can either be monotonously increased or reset to a natural number.

3. For a given time-series constraint satisfying the four principal conditions and for any non-negative integer δ, Section 6.1.3 constructively proves the existence of the δ-gap automaton, i.e. assuming the loss automaton is known it shows how to construct the δ-gap automaton.

– [**Loss Automaton**] For space reason Section 6.2 focuses only on the construction of the loss automaton for the NB_σ family, the construction for the SUM_WIDTH_σ family being described in [2]. It introduces a sufficient condition on a regular expression σ such that, when σ satisfies this condition, the NB_σ family satisfies the principal conditions of Section 6.1.2. It also shows how to obtain a loss automaton for a NB_σ time-series constraint from the seed transducer [9] for σ. The main idea is to compute the *regret* of every transition of the seed transducer as a special case of *minimax regret* [26, 35] from decision theory, which gives the minimum additional cost to pay when one action is chosen instead of another. In CP, the minimax regret has been used for assessing an extra cost when a variable is assigned to a given value [15].

6.1 Synthesising a δ-gap Automaton for a Time-Series Constraint

We present the main contribution of this section namely a systematic method for deriving a δ-gap automaton for a time-series constraint, see Definition 12, satisfying certain conditions that will be given in Definition 16. We first introduce the *gap of a ground time series* in Definition 11, and the *δ-gap automaton for a time-series constraint* in Definition 12. Let $\mathbb{S}$ denote the set of time-series constraints of the NB_σ and SUM_WIDTH_σ families.

Definition 11 (gap of a ground time series) Consider a time-series constraint γ and a ground time series X of length n. The *gap* of X wrt γ, denoted by $\text{gap}_\gamma(X)$, is a function that maps an element of $\mathbb{S} \times \mathbb{Z}^*$ to $\mathbb{N}$. It is the difference between the maximum value of R that could be yielded by a time series of length n, and the value of R yielded by X.

Example 16 will illustrate the notion of gap for different time series.

Definition 12 (δ-gap automaton) Consider a time-series constraint γ and a natural number δ. The *δ-gap automaton for γ* is a minimal automaton that accepts the signatures of all, and only all, ground time series whose gap wrt γ is δ.

Definition 16 will further give a sufficient condition on a time-series constraint γ for the existence of a δ-gap automaton for γ.

Example 15 (0-gap automaton) The 0-gap automaton for NB_PEAK was given in Part (A) of Figure 7. It only recognises the signatures of ground time series containing the maximum number of peaks. △

To construct the δ-gap automaton for a time-series constraint γ we introduce the notion of *loss of a time series*. For a time series of length n, its loss is the difference between n and the length of a shortest time series yielding the same result value of γ. The main idea of our method for generating δ-gap automata is that by knowing the loss of a time series, and whether it contains at least one σ-pattern or not, we can determine its gap.

We now describe how to derive the δ-gap automaton for a time-series constraint γ.

6.1.1 Defining the Loss and the Loss Automaton

Consider a time-series constraint γ and a natural number δ. Definition 13 introduces the *loss of a time series* wrt γ, and Definition 14 presents the notion of *loss automaton* for γ.

Definition 13 (loss of a time series) Consider a time-series constraint γ and a ground time series X of length n. The *loss* of X wrt γ, denoted by $\text{loss}_\gamma(X)$, is a function that maps an element of $\mathbb{S} \times \mathbb{Z}^*$ to $\mathbb{N}$. It is the difference between n and the length of a shortest time series that yields the same result value of γ as X.

Example 16 (gap and loss of a time series) Now we illustrate the computation of the gap and the loss. Consider the NB_PEAK time-series constraint. From [5], the maximum number of peaks in a time series of length n is $\max\left(0, \lfloor \frac{n-1}{2} \rfloor\right)$.

- The time series $X^1 = \langle 1, 2, 1, 2, 1, 2, 1 \rangle$ has a gap of 0 since it contains three peaks, which is maximum, and a loss of 0 since any shorter time series has a smaller number of peaks.
- The time series $X^2 = \langle 1, 2, 1, 2, 1, 1, 1, 1 \rangle$ has a gap of 1 since it has only two peaks, when three is the maximum, and a loss of 3 since a shortest time series with 2 peaks is of length 5.
- The time series $X^3 = \langle 1, 1, 1, 0, 0, 1, 1, 1, 1 \rangle$ has a gap of 4 since it has no peaks, when the maximum is 4, and a loss of 8 since a shortest time series without any peaks is of length 1. △

Definition 14 (loss automaton for a time-series constraint) Consider a time-series constraint γ. A *loss automaton* for γ is a counter automaton over the alphabet $\{<, =, >\}$ with a constant number of counters such that, for any ground time series X, it returns $\text{loss}_\gamma(X)$ after having consumed the signature of X.

For the NB_σ and SUM_WIDTH_σ families, a loss automaton can be synthesised from the seed transducer of the regular expression σ. For the NB_σ family, this will be explained in Section 6.2.

6.1.2 Principal Conditions for Deriving a δ-Gap Automaton

Consider a g_f_σ time-series constraint, denoted by γ, and a natural number δ. Definition 16 formulates a sufficient condition, consisting of a conjunction of four conditions, named *principal conditions*, for the existence of the δ-gap automaton for γ. The first three principal conditions express the idea that, knowing the loss of a time series and, whether it has at least one σ-pattern or not, fully determines the gap of this time series. The fourth condition requires the existence of a loss automaton $\mathcal{M}$ for γ, whose counters may either monotonously increase, or be reset to a natural number, and each accepting state of $\mathcal{M}$ either accepts only signatures with at least one occurrence of σ, or accepts only signatures without any occurrence of σ.

Before formulating the principal conditions, Definition 15 introduces the notions of before-found and after-found state of a loss automaton.

Definition 15 (before-found and after-found states) Consider a loss automaton $\mathcal{M}$ for the g_f_σ time-series constraint. An accepting state q of $\mathcal{M}$ is a *before-found* (resp. *after-found*) state, if there exists a time series X without any σ-patterns (resp. with at least one σ-pattern) such that, after having consumed the signature of X, q is the final state of $\mathcal{M}$.

Note that an accepting state of a loss automaton can have both statuses.

Definition 16 (principal conditions) Consider a $\gamma(X, R)$ time-series constraint. The *four principal conditions on* γ are defined as follows:

1. **Gap-to-loss condition.** There exists a function $h_\gamma : \mathbb{S} \times \mathbb{N} \times \{0, 1\} \times \mathbb{N} \to \mathbb{N}$, called the *gap-to-loss function*, such that for any ground time series $X = \langle X_1, X_2, \ldots, X_n \rangle$, we have $\text{loss}_\gamma(X)$ being equal to $h_\gamma(\text{gap}_\gamma(X), \text{sgn}(R), n)$, where sgn is the signum function. Hence, in order to compute the loss of a ground time series it is enough to know (i) its gap, (ii) whether it has at least one σ-pattern or not, and (iii) the length of this time series.
2. **Boundedness condition.** For given values of $\text{gap}_\gamma(X)$ and $\text{sgn}(R)$, and for any n in $\mathbb{N}$, the value of the gap-to-loss function $h_\gamma(\text{gap}_\gamma(X), \text{sgn}(R), n)$ belongs to a bounded integer interval, called the *loss interval wrt* $\langle \text{gap}_\gamma(X), \text{sgn}(R) \rangle$.
3. **Disjointedness condition.** For a given value of $\text{sgn}(R)$, and two different values of gap, δ_1 and δ_2, the loss intervals wrt $\langle \delta_1, \text{sgn}(R) \rangle$ and wrt $\langle \delta_2, \text{sgn}(R) \rangle$ are disjoint.
4. **Loss-automaton condition.** There exists a loss automaton $\mathcal{M}$ for γ satisfying all the following conditions:
   (a) Every counter update of $\mathcal{M}$ has one of the following forms:
      i. The counter is incremented by a natural number, or by the value of another counter.
      ii. The value of the counter is reset to a natural number.
   (b) The initial values of the counters of $\mathcal{M}$ are natural numbers.
   (c) The acceptance function of $\mathcal{M}$ is a weighted sum with natural number coefficients of the last values of the counters of $\mathcal{M}$ after having consumed an input signature.
   (d) The sets of before-found states and after-found states of $\mathcal{M}$ are disjoint. It means that, by knowing the final state of $\mathcal{M}$ after having consumed the signature of any ground time series X, we also know the value of $\text{sgn}(R)$ yielded by X.

Conditions 1., 2., 3. are called the *gap-loss-relation conditions*, Conditions 4a, 4b, 4c are called the *non-negativity conditions*, while Condition 4d is called the *separation condition* on $\mathcal{M}$.

Example 17 (principal conditions) Consider a $\gamma(X, R)$ time-series constraint. For the time series X^1, X^2, and X^3 of Example 16, Figure 8 shows the relation between the gap, the loss, the time-series lengths, and R when γ is NB_PEAK. For any time series X^i (with $i \in [1, 3]$) of length n_i yielding R_i as the value of R, its gap (resp. loss) is equal to the length of the violet (resp. blue) dotted line segment starting from the point X^i of coordinates (n_i, R_i). Note that the boundedness and the disjointedness conditions are satisfied for NB_PEAK. △

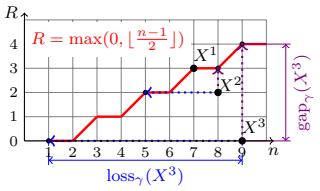


Fig. 8: The horizontal (resp. vertical) axis represents the length of the sequence n (resp. the result value R of $\gamma = \text{NB\_PEAK}$). The red curve shows the maximum value of R for a given n; any point X^i with coordinates (n_i, R_i) denotes all time series of length n_i yielding R_i as the value of R. The length of the blue (resp. violet) dotted line-segments starting from X^i equals the loss (resp. gap) of X^i.

6.1.3 Synthesising the δ-Gap Automaton

Consider a γ time-series constraint satisfying all four principal conditions of Section 6.1.2, and a natural number δ. We prove that the δ-gap automaton for γ exists. First, Lemma 1 states a necessary and sufficient condition in terms of loss for a ground time series to have its gap being a given constant when the gap-loss-relation condition is satisfied. This lemma allows one to describe in terms of loss the set of ground time series whose gap is δ. Then using the result of Lemma 1, Theorem 4 constructively proves that the δ-gap automaton for γ exists.

Lemma 1 (relation between gap and loss) *Consider a $\gamma(X, R)$ time-series constraint such that the gap-loss-relation conditions, see Definition 16, are all satisfied, and a natural number δ. Then, for a time series X, $\text{gap}_\gamma(X)$ is δ iff $\text{loss}_\gamma(X)$ belongs to the loss interval wrt $\langle \delta, \text{sgn}(R) \rangle$.*

Proof The necessity follows from the boundedness condition, see Condition 2, and the sufficiency follows from the disjointedness condition, see Condition 3 of Definition 16.

Theorem 4 (existence of the δ-gap automaton) *Consider a $g\_f\_\sigma(X, R)$ time-series constraint, denoted by γ, such that all four principal conditions, described in Definition 16, are satisfied. Then the δ-gap automaton for γ exists.*

Proof Let us denote by $\mathcal{M}$ the loss automaton for γ, satisfying the non-negativity and the separation conditions. Note that such automaton necessarily exists since the loss-automaton condition, see Condition 4 of Definition 16, is satisfied. We prove the theorem by explicitly constructing a constant-size automaton $\mathcal{A}_\mathcal{M}$ using $\mathcal{M}$; after minimising $\mathcal{A}_\mathcal{M}$ we obtain the sought δ-gap automaton.

[Construction of $\mathcal{A}_\mathcal{M}$] By Lemma 1, there exist a loss interval $\mathcal{L}_{\delta,0}$ wrt $\langle \delta, 0 \rangle$ and a loss interval $\mathcal{L}_{\delta,1}$ wrt $\langle \delta, 1 \rangle$ such that any ground time series X, whose gap is δ, belongs to one of the following types:

- **Type 1.** The time series X has no σ-patterns and the value of $\text{loss}_\gamma(X)$ is in $\mathcal{L}_{\delta,0}$.
- **Type 2.** The time series X has at least one σ-pattern and the value of $\text{loss}_\gamma(X)$ is in $\mathcal{L}_{\delta,1}$.

Hence, our goal is to construct a constant-size automaton $\mathcal{A}_{\mathcal{M}}$ that recognises the signatures of all, and only all, ground time series that belongs either to Type 1 or to Type 2.

Let $\langle A_1, A_2, \ldots, A_p \rangle$ denote the p counters of the loss automaton $\mathcal{M}$, whose initial values are $\langle v_1, v_2, \ldots, v_p \rangle$, let $\alpha(A_1, A_2, \ldots, A_p)$ denote the acceptance function of $\mathcal{M}$, let $\hat{\delta}$ be the transition function of $\mathcal{M}$, and let ϕ be the maximum element in $\mathcal{L}_{\delta,0} \cup \mathcal{L}_{\delta,1}$. Then, the states, the initial state, the accepting states, and the transitions of $\mathcal{A}_{\mathcal{M}}$ are defined as follows:

- **States.** For every state q of $\mathcal{M}$, there are $(\phi + 2)^p$ states in $\mathcal{A}_{\mathcal{M}}$, each of which is labelled with $q_{i_1,i_2,\ldots,i_p}$, with every i_j (with $j \in [1, p]$) being in $[0, \phi + 1]$.
- **Initial state.** If q^0 is the initial state of $\mathcal{M}$, then $q^0_{v_1,v_2,\ldots,v_p}$ is the initial state of $\mathcal{A}_{\mathcal{M}}$.
- **Accepting states.** A state $q_{i_1,i_2,\ldots,i_p}$ of $\mathcal{A}_{\mathcal{M}}$ is accepting iff either
  1. q is a before-found state of $\mathcal{M}$ and the value of $\alpha(i_1, i_2, \ldots, i_p)$ is within $\mathcal{L}_{\delta,0}$, or
  2. q is an after-found state of $\mathcal{M}$ and the value of $\alpha(i_1, i_2, \ldots, i_p)$ is within $\mathcal{L}_{\delta,1}$.
- **Transitions.** There is a transition from state $q_{i_1,i_2,\ldots,i_p}$ (with $i_1, i_2, \ldots, i_p \in [0, \phi + 1]$) to state $q^*_{k_1,k_2,\ldots,k_p}$ labelled with s in {'<', '=', '>'}, if the value of the transition function $\hat{\delta}(q, \langle i_1, i_2, \ldots, i_p \rangle, s)$ is equal to $(q^*, \langle i^*_1, i^*_2, \ldots, i^*_p \rangle)$, where every k_j is equal to $\min(\phi + 1, i^*_j)$, with j in $[1, p]$.

[Interpretation of the states of $\mathcal{A}_{\mathcal{M}}$] If after consuming the signature of some ground time series, the automaton $\mathcal{A}_{\mathcal{M}}$ arrives in a state $q_{i_1,i_2,\ldots,i_p}$, then after consuming the same signature, the loss automaton $\mathcal{M}$ arrives in state q; for every $j \in [1, p]$, when $i_j \leq \phi$ (resp. $i_j = \phi + 1$), the counter A_j has value i_j (resp. is strictly greater than ϕ). Hence, the states of $\mathcal{A}_{\mathcal{M}}$ encode the counter values of $\mathcal{M}$ when consuming the same input signature.

[Size of $\mathcal{A}_{\mathcal{M}}$] By construction, the automaton $\mathcal{A}_{\mathcal{M}}$ has a constant size, i.e. its number of states is $m \cdot (\phi + 2)^p$, where m, p and ϕ are parameters, i.e. independent from the time-series length, respectively defined as:

- the number of states of $\mathcal{M}$,
- the number of counters of $\mathcal{M}$,
- the maximum value of $\mathcal{L}_{\delta,0} \cup \mathcal{L}_{\delta,1}$, where $\mathcal{L}_{\delta,0}$ and $\mathcal{L}_{\delta,1}$ are bounded intervals depending only on the constraint γ and the gap δ.

We explain why $\mathcal{A}_{\mathcal{M}}$ needs only $m \cdot (\phi + 2)^p$ states to recognise the signatures of all, and only all, ground time series of either Type 1 or Type 2. By the boundedness condition (Condition 2 of Definition 16) and by definition of ϕ, for any ground time series whose gap is δ, its loss cannot exceed ϕ. We show that if, when consuming the signature of some ground time series, the value of some counter of $\mathcal{M}$ becomes greater than ϕ, then we no longer need to know its exact value.

Recall that the acceptance function α of $\mathcal{M}$ is a weighted sum with natural coefficients of the last values of the counters of $\mathcal{M}$. If, for a counter A_j, the corresponding coefficient in α is zero, then it does not affect the value of α, and the exact value of A_j is irrelevant. Otherwise, once the value of A_j exceeds ϕ, the value of α also exceeds ϕ, and the loss of such a time series is greater than ϕ. By the non-negativity conditions, if the value of A_j exceeds ϕ it can either increase even more, or it can be reset to a natural constant. In either case, the exact value of A_j is irrelevant, and it is enough to know a lower bound, $\phi + 1$ of its value.

[Correctness of $\mathcal{A}_{\mathcal{M}}$] We now prove that the constructed automaton $\mathcal{A}_{\mathcal{M}}$ is sound, i.e. it recognises the signatures of *only* ground time series of either Type 1 or Type 2, and complete i.e. it recognises the signatures of *all* ground time series of either Type 1 or Type 2.

- **Soundness of $\mathcal{A}_{\mathcal{M}}$.** We prove the soundness of $\mathcal{A}_{\mathcal{M}}$ by contradiction. Assume there exists a ground time series X recognised by $\mathcal{A}_{\mathcal{M}}$ and whose gap is not δ. Let $q_{i_1,i_2,\ldots,i_p}$ be the final state of $\mathcal{A}_{\mathcal{M}}$ after consuming the signature S of X. Due to the non-negativity conditions, by construction of $\mathcal{A}_{\mathcal{M}}$ this means that, after consuming S, the counter automaton $\mathcal{M}$ finishes in the state q of $\mathcal{M}$, and for every $j \in [1, p]$, if $i_j \leq \phi$ (resp. $i_j = \phi + 1$), then the counter A_j has value i_j (resp. is strictly

greater than ϕ). By the separation condition on $\mathcal{M}$, the state q of $\mathcal{M}$ is either a before-found or an after-found state. Since $q_{i_1,i_2,\ldots,i_p}$ is an accepting state of $\mathcal{A}_\mathcal{M}$, then either q is a before-found state and $\alpha(i_1, i_2, \ldots, i_p) \in \mathcal{L}_{\delta,0}$, or q is an after-found state and $\alpha(i_1, i_2, \ldots, i_p) \in \mathcal{L}_{\delta,1}$. In the former (resp. latter) case, X belongs to Type 1 (resp. Type 2), and by Lemma 1, the gap of X is δ, a contradiction.

– **Completeness of $\mathcal{A}_\mathcal{M}$.** We prove the completeness of $\mathcal{A}_\mathcal{M}$ also by contradiction. Assume there exists a ground time series X whose gap is δ, i.e. it belongs either to Type 1 or to Type 2, but its signature S is not recognised by $\mathcal{A}_\mathcal{M}$. Then,
  1. either the final state $q_{i_1,i_2,\ldots,i_p}$ of $\mathcal{A}_\mathcal{M}$ after consuming S is not accepting,
  2. or the automaton $\mathcal{A}_\mathcal{M}$ cannot consume the full signature S.

  We show that both situations are impossible.
  – **Impossibility of Situation 1.** Due to the non-negativity conditions, and by construction of $\mathcal{A}_\mathcal{M}$, after having consumed the signature of X, the automaton $\mathcal{M}$ ends in state q of $\mathcal{M}$, and the value of the acceptance function is equal to $\alpha(i_1, i_2, \ldots, i_p)$. Since the gap of X is δ, by Lemma 1 and by the separation condition, either q is a before-found state of $\mathcal{M}$ and $\alpha(i_1, i_2, \ldots, i_p)$ belongs to $\mathcal{L}_{\delta,0}$ or q is an after-found state of $\mathcal{M}$ and $\alpha(i_1, i_2, \ldots, i_p)$ belongs to $\mathcal{L}_{\delta,1}$. In either case, the state $q_{i_1,i_2,\ldots,i_p}$ of $\mathcal{A}_\mathcal{M}$ must be accepting by construction, thus Situation 1 is impossible.
  – **Impossibility of Situation 2.** Assume that (1) at a state $q_{i_1,i_2,\ldots,i_p}$ of $\mathcal{A}_\mathcal{M}$, there does not exist a transition labelled with some input symbol s, and that (2) $\mathcal{A}_\mathcal{M}$ needs to trigger this transition when consuming the signature of X. Then, at state q of $\mathcal{M}$, there does not exist a transition labelled with s. This contradicts the nature of the loss automaton $\mathcal{M}$ since it must compute the loss of any ground time series, and thus accept any time series. Hence, Situation 2 is also impossible.

  Therefore, both situations are impossible, which implies that the time series X does not exist, and thus the automaton $\mathcal{A}_\mathcal{M}$ is complete.

Since $\mathcal{A}_\mathcal{M}$ is sound and complete, the minimisation of $\mathcal{A}_\mathcal{M}$ gives the sought δ-gap automaton. □

6.2 Synthesising the Loss Automaton for the NB_σ Family

First, for the NB_σ family, we show that, when σ has a property, named the HOMOGENEITY *property,* the first three principal conditions of Definition 16 are satisfied. Second, based on the HOMOGENEITY property we show how to satisfy the fourth principal condition by constructing from the seed transducer for σ a loss automaton satisfying the loss-automaton condition. Consequently, the constructive proof of Theorem 4 can be used to derive the δ-gap automaton.

1. Section 6.2.1 introduces the HOMOGENEITY property. Sections 6.2.2 and 6.2.3 both assume the HOMOGENEITY property.
2. Section 6.2.2 proves three theorems stating that, the gap-to-loss, the boundedness, and the disjointedness conditions are satisfied for NB_σ.
3. Section 6.2.3 gives a systematic method for constructing a loss automaton $\mathcal{M}$ satisfying the non-negativity and the separation conditions.

6.2.1 The HOMOGENEITY *Property*

*Property 2 (*HOMOGENEITY *property)* A regular expression σ has the HOMOGENEITY property if the following conditions are both satisfied:

1. The pair $\langle \sigma, b_\sigma \rangle$ is a *recognisable pattern* [25]. This implies that the seed transducer $\mathcal{T}_\sigma$ for σ exists and can be constructed by the method of [25].
2. For any state q of $\mathcal{T}_\sigma$ that is the destination state of a `found`-transition, the number of transitions in the shortest `found`-path starting from q is a constant that does not depend on q.

For a regular expression σ with the HOMOGENEITY property, the following lemma gives the maximum number of σ-patterns in a time series of length n.

Lemma 2 (maximum of the result value) *Consider a time-series constraint* NB_σ *such that* σ *has the* HOMOGENEITY *property, and* $\mathcal{T}_\sigma$ *denotes the seed transducer for* σ*. Let* d_σ *denote the length of shortest* `found`*-path in* $\mathcal{T}_\sigma$ *starting from any state that is the destination of a* `found`*-transition, and let* c_σ *denote the difference between* d_σ *and the length of shortest* `found`*-path in* $\mathcal{T}_\sigma$ *starting from the initial state of* $\mathcal{T}_\sigma$*. Then, the maximum number of* σ*-patterns in a time series of length* n *is computed as*

$$\left\lfloor \frac{n - c_\sigma}{d_\sigma} \right\rfloor . \tag{14}$$

Proof For any time series $X = \langle X_1, X_2, \ldots, X_n \rangle$, there is a bijection between its set of σ-patterns and the `found` symbols in the output sequence of $\mathcal{T}_\sigma$ after consuming the signature of X. Hence, we need to show that $\left\lfloor \frac{n-c_\sigma}{d_\sigma} \right\rfloor$ is the maximum number of the `found` symbols in the output sequence T of $\mathcal{T}_\sigma$ after having consumed the signature of any time series of length n. The first `found` symbol in T cannot occur before the position ℓ, where ℓ is the length of the shortest `found`-path starting from the initial state. Since $\mathcal{T}_\sigma$ has the HOMOGENEITY property then every other `found` symbol can occur in T with the interval of d_σ. Such an T output sequence has the number of `found` symbols being equal to $\left\lfloor \frac{n-(\ell-d_\sigma)}{d_\sigma} \right\rfloor$. We replace $\ell - d_\sigma$ with c_σ and obtain Formula (14). □

6.2.2 Verifying the Gap-Loss-Relation Conditions

This section shows that the gap-loss-relation conditions, see Definition 16, for a NB_σ time-series constraint are satisfied, assuming σ has the HOMOGENEITY property. Theorem 5 proves the gap-to-loss condition and derives the formula for the gap-to-loss function; Theorem 6 proves the boundedness condition and derives the formula of loss interval for a given gap and sign of the result value, and, finally, Theorem 7 proves the disjointedness condition.

Theorem 5 (gap-to-loss condition) *Consider a* $\gamma(X, R)$ *time-series constraint that belongs to the* NB_σ *family with* σ *having the* HOMOGENEITY *property. First, the gap-to-loss condition is satisfied for* γ*. Second, for any ground time series* X *of length* n*, the gap-to-loss function is defined by:*

$$\text{loss}_\gamma(X) = \text{gap}_\gamma(X) \cdot d_\sigma + (1 - \text{sgn}(R)) \cdot (\min(n, c_\sigma) - 1) + \max(0, n - c_\sigma) \bmod d_\sigma, \tag{15}$$

where sgn *is the signum function, and* c_σ *and* d_σ *are the constants from the maximum value of* R *given in Lemma 2.*

Proof We successively consider two disjoint cases wrt $\text{sgn}(R)$.

[sgn(R) is zero] We need to prove that $\text{loss}_\gamma(X)$ is equal to $\text{gap}_\gamma(X) \cdot d_\sigma + \min(n, c_\sigma) - 1 + \max(0, n - c_\sigma) \bmod d_\sigma$. When R is zero, the loss of X is $n-1$ since a shortest time series without any σ-patterns is of length 1. Thus, we need to show that $\text{gap}_\gamma(X) \cdot d_\sigma + \min(n, c_\sigma) - 1 + \max(0, n - c_\sigma) \bmod d_\sigma$ is equal to $n - 1$. From the maximum value of R, given by the HOMOGENEITY property, we have the following equality:

$$\text{gap}_\gamma(X) = \max\left(0, \left\lfloor \frac{n - c_\sigma}{d_\sigma} \right\rfloor\right) - R = \max\left(0, \left\lfloor \frac{n - c_\sigma}{d_\sigma} \right\rfloor\right). \tag{16}$$

Let us consider two cases wrt the value of $\text{gap}_\gamma(X)$, namely:

- $\text{gap}_\gamma(X)$ is zero. By (16), $n < c_\sigma + d_\sigma$, and the value of the right-hand side of (15) is equal to $\min(n, c_\sigma) - 1 + \max(0, n - c_\sigma)$, which is $n - 1$.
- $\text{gap}_\gamma(X)$ is positive. Then, by (16), $n \geq c_\sigma + d_\sigma$, and we have the following equality:

$$\text{gap}_\gamma(X) = \left\lfloor \frac{n - c_\sigma}{d_\sigma} \right\rfloor = \frac{n - c_\sigma - (n - c_\sigma) \bmod d_\sigma}{d_\sigma} \tag{17}$$

From (17) we obtain the expression for $n - 1$, which is $\text{gap}_\gamma(X) \cdot d_\sigma + c_\sigma - 1 + (n - c_\sigma) \bmod d_\sigma$.

[sgn(R) is one] We need to prove that $\text{loss}_\gamma(X)$ is equal to $\text{gap}_\gamma(X) \cdot d_\sigma + \max(0, n - c_\sigma) \bmod d_\sigma$. Since R is positive, n is strictly greater than c_σ, and thus $\max(0, n - c_\sigma)$ is equal to $n - c_\sigma$. Further, by definitions of gap and loss, we have:

$$\text{gap}_\gamma(X) = \left\lfloor \frac{n - c_\sigma}{d_\sigma} \right\rfloor - R = \frac{n - c_\sigma - (n - c_\sigma) \bmod d_\sigma}{d_\sigma} - \frac{(n - \text{loss}_\gamma(X)) - c_\sigma}{d_\sigma} \tag{18}$$

Since on the right-hand side of (18), both divisions are integer divisions we obtain:

$$\text{gap}_\gamma(X) = \frac{\text{loss}_\gamma(X) - (n - c_\sigma) \bmod d_\sigma}{d_\sigma}. \tag{19}$$

By isolating $\text{loss}_\gamma(X)$ from (19) we obtain the formula of the theorem. □

Example 18 (gap-to-loss condition) Consider a NB_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with σ being the PEAK regular expression, which has the HOMOGENEITY property. Hence, we can apply Theorem 5 for computing the gap-to-loss function for NB_σ. By Lemma 2, the maximum value of R is $\max\left(0, \left\lfloor \frac{n-1}{2} \right\rfloor\right)$, and thus c_σ and d_σ, are 1 and 2, respectively. Then the gap-to-loss function for NB_σ is

$$\text{loss}_\gamma(X) = 2 \cdot \text{gap}_\gamma(X) + \max(0, n - 1) \bmod 2.$$ △

Theorem 6 (boundedness condition) *Consider a $\gamma(X, R)$ time-series constraint that belongs to the* NB_σ *family with σ having the* HOMOGENEITY *property. First, the boundedness condition is satisfied for γ; second, for any given gap δ and any value of* $\text{sgn}(R)$*, the loss interval $[\ell_{min}, \ell_{max}]$ wrt $\langle \delta, \text{sgn}(R) \rangle$ is defined by:*

(i) $\ell_{min} = \delta \cdot d_\sigma + (1 - \text{sgn}(R)) \cdot \text{sgn}(\delta) \cdot (c_\sigma - 1)$,
(ii) $\ell_{max} = d_\sigma \cdot (\delta + 1) - 1 + (1 - \text{sgn}(R)) \cdot (c_\sigma - 1)$.

Proof Let X be a ground time series of length n whose gap is δ. From Theorem 5, we have that $\text{loss}_\gamma(X)$ is $\delta \cdot d_\sigma + (1 - \text{sgn}(R)) \cdot (\min(n, c_\sigma) - 1) + \max(0, n - c_\sigma) \bmod d_\sigma$. By case analysis wrt the value of $\text{sgn}(R)$, i.e. either 0 or 1, we now show that $\ell_{min} \leq \text{loss}_\gamma(X) \leq \ell_{max}$.

[sgn(R) is zero] In this case, $\text{loss}_\gamma(X)$ simplifies to $\delta \cdot d_\sigma + \min(n, c_\sigma) - 1 + \max(0, n - c_\sigma) \bmod d_\sigma$. Since $\delta \cdot d_\sigma - 1$ is a constant, in order to prove that ℓ_{min} (resp. ℓ_{max}) is a lower (resp. upper) bound on $\text{loss}_\gamma(X)$, we need to find the minimum (resp. maximum) of the function $z(n) = \min(n, c_\sigma) + \max(0, n - c_\sigma) \bmod d_\sigma$.

(i) $\boldsymbol{\ell_{min} \leq \text{loss}_\gamma(X)}$. We prove that $\text{loss}_\gamma(X) = \delta \cdot d_\sigma + z(n) \geq \ell_{min}$ by case analysis on δ:
  (a) **[sgn(δ) is zero]** As shown in the proof of Theorem 5, $n < c_\sigma + d_\sigma$ and the minimum value of the function $z(n)$ is 1, and is reached for n being 1.
  (b) **[sgn(δ) is one]** We have $n \geq c_\sigma + d_\sigma$, and thus $\min(n, c_\sigma)$ is equal to c_σ, and the minimum value of the function $z(n)$ is c_σ.

  Hence, $\delta \cdot d_\sigma + \text{sgn}(\delta) \cdot (c_\sigma - 1)$ is indeed a lower bound on $\text{loss}_\gamma(X)$ when $\text{sgn}(R)$ is zero.

(ii) $\boldsymbol{\ell_{max} \geq \text{loss}_\gamma(X)}$. We prove that $\text{loss}_\gamma(X) \leq \ell_{max}$. The maximum value of $z(n)$ is $c_\sigma + d_\sigma - 1$. Hence, $d_\sigma \cdot (\delta + 1) - 1 + c_\sigma - 1$ is indeed an upper bound on $\text{loss}_\gamma(X)$.

[sgn(R) is one] In this case, $\text{loss}_\gamma(X)$ simplifies to $\delta \cdot d_\sigma + \max(0, n - c_\sigma) \bmod d_\sigma$. A lower (resp. upper) bound on $(n - c_\sigma) \bmod d_\sigma$ is zero (resp. $d_\sigma - 1$). Hence, ℓ_{min} and ℓ_{max} are, respectively, a lower and an upper bound on $\text{loss}_\gamma(X)$. □

Example 19 (boundedness condition) Consider a NB_$\sigma(X, R)$ time-series constraint with σ being the PEAK regular expression. Since σ has the HOMOGENEITY property we can apply Theorem 6 for computing the loss interval for NB_σ. Recall that the values of c_σ and d_σ, are respectively, 1 and 2. Then, for any value δ of gap and any value of $\text{sgn}(R)$, the loss interval wrt $\langle \delta, \text{sgn}(R) \rangle$ is $[2 \cdot \delta, 2 \cdot \delta + 1]$. △

Theorem 7 (disjointedness condition) *Consider a* NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ *time-series constraint such that* σ *has the* HOMOGENEITY *property. Then the disjointedness condition is satisfied for* NB_σ.

Proof The disjointedness condition can be proved using the formula of the loss interval of Theorem 6. For each value of $\mathrm{sgn}(R)$, i.e. either 0 or 1, we take two different values of gap, w.l.o.g. δ and $\delta + t$ with a non-negative integer t, and show that the upper limit of the loss interval wrt $\langle \delta, \mathrm{sgn}(R)\rangle$ is strictly less than the lower limit of the loss interval wrt $\langle \delta + t, \mathrm{sgn}(R)\rangle$. This implies the disjointedness condition. □

6.2.3 Verifying the Loss-Automaton Condition

We focus on the loss-automaton condition for the NB_σ time-series constraints, i.e. we construct a loss automaton $\mathcal{M}$ for NB_σ satisfying the non-negativity and the separation conditions. This is done by deriving $\mathcal{M}$ from a seed transducer for σ, which exists assuming σ has the HOMOGENEITY property [25]. In order to satisfy the separation condition for the loss automaton for NB_σ, we require the seed transducer for σ to have a specific form that we now introduce in Definition 17.

Definition 17 (separated seed transducer) Given a regular expression σ, a seed transducer $\mathcal{T}_\sigma$ for σ is *separated* iff for any state q of $\mathcal{T}_\sigma$, one of the two following conditions holds:

1. Any path from the initial state of $\mathcal{T}_\sigma$ to q is a `found`-path.
2. There are no `found`-paths from the initial state of $\mathcal{T}_\sigma$ to q.

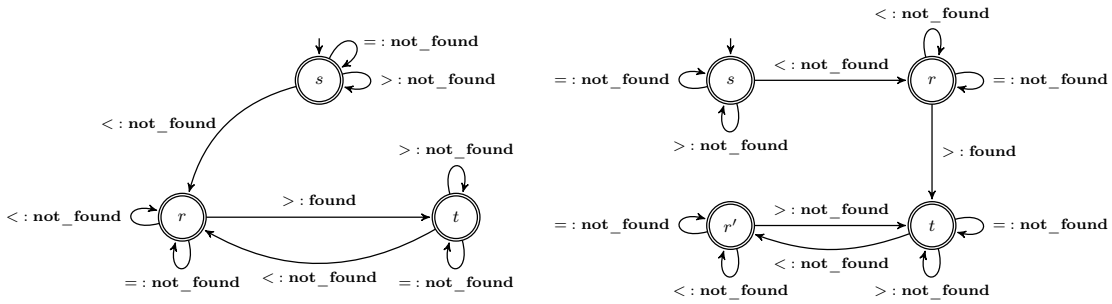


Fig. 9: (A) Seed transducer and (B) separated seed transducer for the PEAK regular expression.

Example 20 (separated seed transducer) Part (B) of Figure 9 gives the separated seed transducer for PEAK obtained from the seed transducer in Part (A). △

Note that, even if the seed transducer for σ constructed by the method of [25] is not separated, it can be easily made so by duplicating some of its states. Subsequently we assume that the seed transducer for σ is separated, and we derive the loss automaton $\mathcal{M}$ in the same way as we generate counter automata for time-series constraints [9], namely:

1. First, we identify the required counters of $\mathcal{M}$ and their role.
2. Second, to each phase letter of the output alphabet of the seed transducer for σ, we associate a set of instructions, i.e. counter updates. The loss automaton $\mathcal{M}$ is obtained by replacing every phase letter of the seed transducer for σ by the corresponding set of instructions.

Identifying the Required Counters of the Loss Automaton Consider a NB_σ time-series constraint. Intuitively, when consuming the signature of a ground time series, every transition triggered by the seed transducer $\mathcal{T}_\sigma$ for σ has a certain impact on the loss of this time series. To quantify this impact for the case of NB_σ time-series constraints, Definition 18 introduces the notion of *regret of a transition* of a seed transducer for σ. The regret of a transition t gives how many additional transitions $\mathcal{T}_\sigma$ has to trigger, before it can trigger the next `found`-transition, if it triggers t rather than the transition on a shortest `found`-path.

Definition 18 (regret of a transition) Consider a regular expression σ and its seed transducer $\mathcal{T}_\sigma$. For any transition t of $\mathcal{T}_\sigma$ from state q_1 to state q_2, the *regret* of t equals one plus the difference between the lengths of the shortest `found`-paths from q_2, respectively q_1.

Example 21 (regret of a transition) Consider the PEAK regular expression, whose separated seed transducer is given in Part (B) of Figure 9. We denote by $q_1 \xrightarrow{a} q_2$ a transition of the seed transducer from state q_1 to state q_2 whose input symbol is a. All transitions in $\{s \xrightarrow{<} r, r \xrightarrow{>} t, t \xrightarrow{>} r', r' \xrightarrow{<} t\}$ between two distinct states have a regret of 0, while all transitions in $\{s \xrightarrow{>} s, s \xrightarrow{=} s, r \xrightarrow{<} r, r \xrightarrow{=} r, t \xrightarrow{>} t, t \xrightarrow{=} t, r' \xrightarrow{<} r', r' \xrightarrow{=} r'\}$ have a regret of 1. △

Lemma 3 shows the connection between the loss of a ground time series X and the regret of the transitions triggered by the seed transducer for σ when consuming the signature of X.

Lemma 3 (regret-loss relation) *Consider a $\gamma(X, R)$ time-series constraint with γ being* NB_σ *such that σ has the* HOMOGENEITY *property. Let $t = \langle t_1, t_2, \ldots, t_{n-1}\rangle$ denote the sequence of transitions triggered by the seed transducer $\mathcal{T}_\sigma$ for σ upon consuming the signature of $X = \langle X_1, X_2, \ldots, X_n\rangle$, and let t^* denote the index of the last* `found`*-transition in t, if no such transition exists, t^* is zero. The following equality holds:*

$$\text{loss}_\gamma(X) = n - 1 - t^* + \sum_{i=1}^{t^*} \rho(t_i), \text{ where } \rho(t_i) \text{ denotes the regret of transition } t_i.$$

Proof Since $\langle t_{t^*+1}, t_{t^*+2}, \ldots, t_{n-1}\rangle$ does not contain any `found`-transition, it implies that the loss of X is at least $n - 1 - t^*$. Then, the sum $\sum_{i=1}^{t^*} \rho(t_i)$ shows how many additional transitions were triggered to achieve the same number of `found`-transitions in the output sequence. Hence, the loss of X is the sum of $n - 1 - t^*$ and $\sum_{i=1}^{t^*} \rho(t_i)$. □

Example 22 (regret-loss transition) Consider the PEAK regular expression, whose separated seed transducer $\mathcal{T}_{\text{PEAK}}$ is given in Part (B) of Figure 9. Upon consuming the signature of the time series $X = \langle 1, 1, 2, 1, 2,\ 1, 1, 2, 1, 2\rangle$, the seed transducer $\mathcal{T}_{\text{PEAK}}$ triggers the following sequence of transitions $\langle s \xrightarrow{=} s, s \xrightarrow{<} r, r \xrightarrow{>} t, t \xrightarrow{<} r',\ r' \xrightarrow{>} t, t \xrightarrow{=} t, t \xrightarrow{<} r', r' \xrightarrow{>} t, t \xrightarrow{<} r'\rangle$. The index of the last triggered `found`-transition is 8. From Lemma 3, we obtain $\text{loss}_\gamma(X) = 10-1-8+(1+0+0+0+0+1+0+0+0) = 3$. △

From Lemma 3, three counters are needed for the loss automaton. Given a prefix of a signature consumed by the seed transducer, let t^* denote the last triggered `found`-transition:

- Counter R gives the sum of the regrets of the transitions triggered before t^*. Note that the regret of t^* is zero.
- Counter D gives the sum of the regrets of the transitions triggered after t^*.
- Counter C gives the number of transitions triggered after t^*.

The initial value of these three counters is zero. The decoration table, given in the next section, follows from Lemma 3.

Decoration Table of a Loss Automaton As stated before, a loss automaton for NB_σ has three counters C, D and R. Given a prefix of some signature consumed by the seed transducer $\mathcal{T}_\sigma$, let t^* denote the last triggered `found`-transition. When $\mathcal{T}_\sigma$ triggers the transition t, we have one of the two following cases:

1. [t is not a `found`-transition] Then t^* is still the last triggered `found`-transition. There is one more transition triggered after t^*, and the counter C must be increased by 1. Further, the value of D should be increased by the regret of t. Finally, counter R remains unchanged.
2. [t is a `found`-transition] Then t becomes the last triggered `found`-transition. Since there is no transition triggered after t, counters C and D must both be reset to 0. Counter R must be increased by the sum of the regrets of all the transitions triggered after t^* and before t, i.e. the value of D.

By Lemma 3, the loss of a time series is the sum between the sum of the regrets of all the triggered transitions before the last `found`-transition and the number of transitions triggered after the last `found`-transition. This is the sum of the last values of C and R. Part (A) of Figure 10 summarises how counters are updated.

initial values	$C \leftarrow 0$	$D \leftarrow 0$	$R \leftarrow 0$
acceptance function	$R + C$		
phase letters	*update of* C	*update of* D	*update of* R
`found`	$C \leftarrow 0$	$D \leftarrow 0$	$R \leftarrow R + D$
`not_found`	$C \leftarrow C + 1$	$D \leftarrow D + \rho(t)$	

(A)

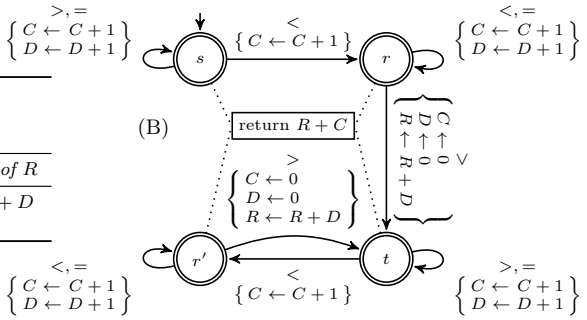


Fig. 10: (A) Decoration table for the loss automaton for NB_σ time-series constraints, where $\rho(t)$ denotes the regret of a transition t of the seed transducer for σ; (B) Loss automaton for NB_PEAK; the initial value of the counters C, D, and R is zero; as the regret of the `not_found` transitions $s \xrightarrow{\leq} r$ and $t \xrightarrow{\leq} r'$ of the seed transducer for σ is zero, the counter D remains unchanged while triggering these two transitions.

To obtain the loss automaton for a NB_σ time-series constraint, we replace every output letter in the separated seed transducer for σ with the corresponding set of counter updates according to the decoration table shown in Part (A) of Figure 10. The initial value of all three counters is zero, and the acceptance function is $C + R$.

Example 23 (loss automaton) The loss automaton for NB_PEAK, obtained from the seed transducer in Part (B) of Figure 9 and from the decoration table in Part (A) of Figure 10, is given in Part (B) of Figure 10. △

6.3 Generalization

We presented a systematic approach for generating δ-gap automata for time-series constraints, and demonstrated its applicability for the NB_σ family. We used the obtained automata both (i) for proving that 70% of our synthesised linear invariants were facet defining, and (ii) for proving the correctness of all non-linear invariants of a database of invariants on conjunctions of time-series constraints.

Although, we did this work in the context of time series, the same method can be used for generating δ-gap automata for any constraint satisfying the four principal conditions. As an example, consider the NB_GROUP(X, R, P) constraint [21,11], where X is a sequence of n integer variables, R is an integer variable, and P is a non-empty finite set of integer numbers. This constraint restricts R to be the number of maximal subsequences of X whose elements are in P. For example, NB_GROUP$(\langle 1,3,4,1,0,9,0\rangle, 3, \{0,1\})$ holds. Then a sharp upper bound on R is $\lfloor \frac{n}{2} \rfloor$, and it can be shown that all the four principal conditions are satisfied for NB_GROUP. Hence by Theorem 4 for any natural δ, the δ-gap automaton for NB_GROUP exists and can be constructed by the method given in the proof of Theorem 4.

7 Evaluation

To test the generated invariants, we use real-world electricity demand data from an industrial partner. The dataset contains time series of length 96 (2 days in half-hour resolution) for multiple years. We use fixed size prefixes of the data to show scaleability of our methods. Before presenting the two experiments we have done, we first sketch the general scheme used to set up our constraints in all our experiments. Our base line, depicted in the first point of the next enumeration, is the state of the art before this article, which uses all the techniques described previously on how to improve the propagation of the individual time-series constraints [5,6,7]. Our improved version uses, in addition to the base line, the invariants of this article as explained in the second point of the next enumeration.

1. For each time-series constraint $C(X, R)$ used on a sequence $X = X_1, X_2, \ldots, X_n$ we post two constraints: the constraint $C(X, R)$ itself, and the constraint $C^r(\langle X_n, X_{n-1}, \ldots, X_1\rangle, R)$ where C^r is the reverse of the constraint C; a constraint C^r is the reverse constraint of a constraint C if, for any sequence $X_1, X_2, \ldots, X_n$, we have the equivalence $C(\langle X_1, X_2, \ldots, X_n\rangle, R) \Leftrightarrow C^r(\langle X_n, X_{n-1}, \ldots, X_1\rangle, R)$. The time-series constraints are encoded as counter automata of SICStus [8] where all intermediate counters and traversed states are made visible so that they can be used in other constraints. While stating only two time-series constraints this allows one to get the result variables on all prefixes and suffixes of X and to enforce lower and upper bounds [5] on these results variables. Finally, for each prefix $X_1, X_2, \ldots X_i$ and corresponding suffix $X_n, X_{n-1}, \ldots, X_i$ we link the corresponding exposed intermediate result variables with the result variable on the full sequence X with a glue constraint [6]. The intuition behind the glue constraint is to channel information from the prefix to the suffix given the result variable on the full sequence. We use the optimised versions of the time-series constraints [7] available in the time-series catalogue [3], i.e. the version using fewer counters, as well as the corresponding glue matrices also available in the catalogue.
2. For each pair of time-series constraints $C_1(X, R_1)$, $C_2(X, R_2)$ used on a same sequence X, we state all invariant constraints that were derived from the pair C_1, C_2 on the intermediate results variables of the different suffixes of the sequence X. For this purpose we used the invariant constraints database we generate by using the methods of this article and that is available in the time-series catalogue. Each constraint of the database was expressed as a linear constraint or a logical formula involving linear constraints, e.g. disjunction of linear constraints.

In a first experiment we consider prefixes of length 25 and test all binary combinations of the considered constraints both with our baseline implementation of the individual constraints (version *pure*) and with the added, generated invariants applied to each suffix (version *incremental*). From the dataset, we extract as features the observed values for a pair of constraints for a time-series instance, and then try to find an assignment that achieves these values. Each problem is feasible, as it is based on an existing assignment. Any improvement of the propagation is due to detecting failures in partial assignments more quickly by applying the invariants to suffixes of the complete series. Our default search strategy labels the signature variables first, followed by the decision variables, always starting with the smallest values. As all constraints used here operate on the signature variables only, we can always find an assignment of the decision variables once a feasible assignment of the signatures is found.

Figure 11 shows the results, with the *pure* baseline above the main diagonal, and the results with the added invariants (*incremental*) below the main diagonal. Each box represents the results for 100 time series. The number in the box, if present, shows how many of the 100 experiments timed out (limit 2 seconds) with the default search strategy. The colour of the cell indicates the average number of backtracks required for the solved instances, based on the legend below the matrix. All experiments were run using SICStus Prolog 4.3.5 on a Windows 10 laptop with 64 GB of memory, using a single core of the Intel i7 processor running at 2.9 GHz base speed.

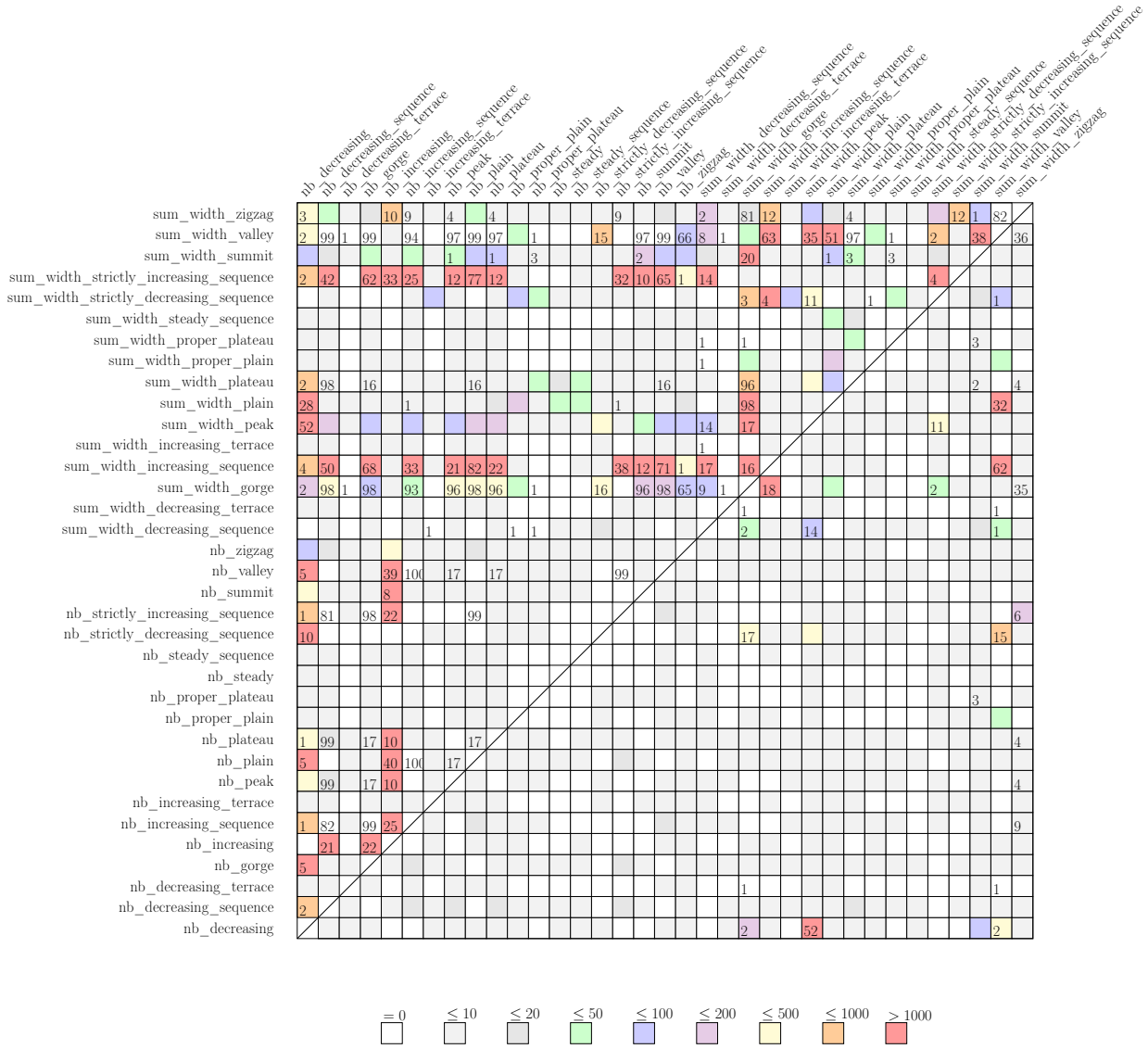


Fig. 11: Comparing baseline (top left) and added invariants (bottom right) models on all binary combinations of considered constraints; Length 25 variables; 100 feasible samples, Number of timeouts as numbers, average number of backtracks of solved problems as cell colour.

Adding the invariants decreases both the number of timeouts and the number of backtracks for most, but not all, constraint combinations. While some constraint combinations are easily solved even without the invariants, there are many cases where the baseline constraints are not able to find a solution quickly, but the added invariants reduce the backtrack count close to zero. It is interesting to note that all combinations of the `nb_` constraints are solved with less than 20 backtracks when the invariants are added, while the baseline constraint do not find any solutions for several combinations of such constraints.

We repeat the experiments, but now for time-series length increasing from 20 to 90, to investigate scaleability of the approach. Figure 12 shows the baseline results on the left, the results with added invariants on the right. We plot the percentage of instances solved as a function of execution time. For the baseline, we see that with increasing problem size the percentage of problems solved steadily drops from 93.9% for size 20 to 75.9% for size 90 with a timeout of 2 seconds. Adding the invariants improves the percentage to 99.3% for size 20, while still achieving 97.9% for size 90.

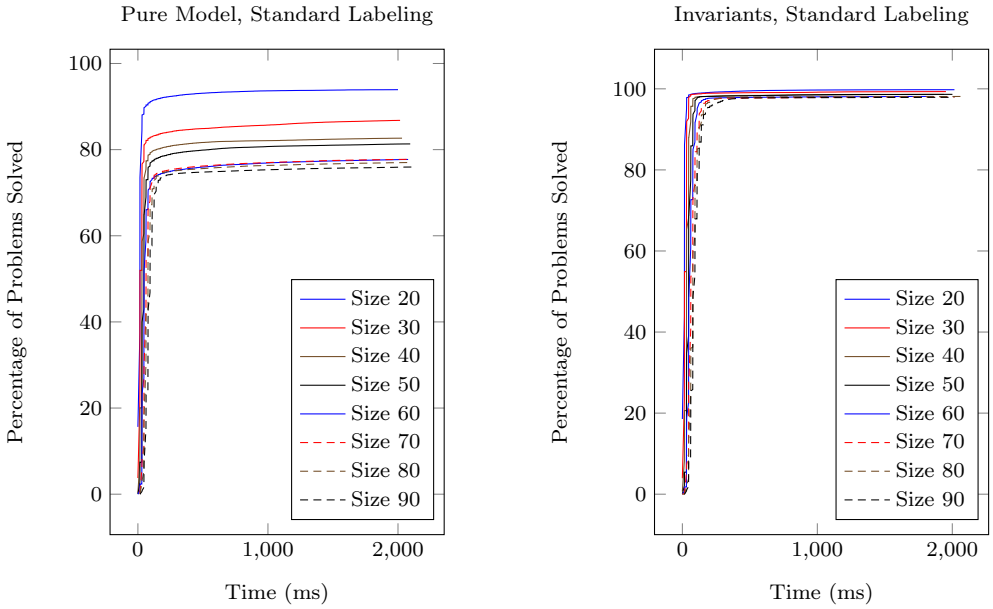
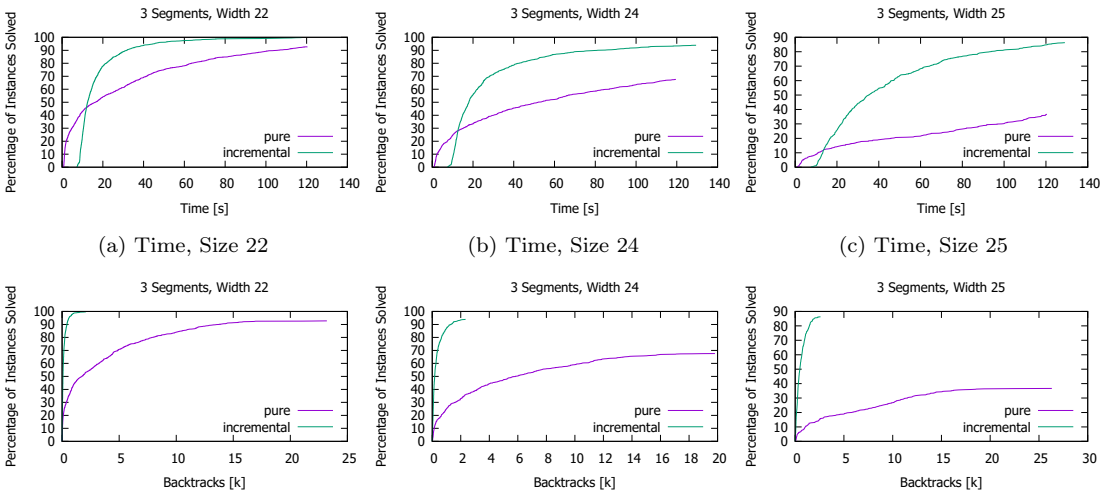


Fig. 12: Comparing baseline (left) and added invariants (right) models on time series of sizes 20-90; 100 feasible sample; Showing cumulative percentage of problems solved as a function of execution time, timeout 2 seconds.

To test the method in a realistic setting, we consider the conjunction of all 35 considered time-series constraints on the dataset. To capture the shape of the time series more accurately, we split the series into overlapping segments from 00-12, 06-18, and 12-24 hours, each segment containing 24 data points, overlapping in 12 data points with the previous segment. We then set up the conjunction of the 35 time-series constraints for each segment, using the *pure* and *incremental* variants described above. This leads to $3 \times 35 \times 2 = 210$ automaton constraints with decision variables. The invariants are created for every pair of constraints, and every suffix, leading to a large number of inequalities. The search routine assigns all signature variables from left to right, and then assigns the decision variables, with a timeout of 120 seconds.

In order to understand the scaleability of the method, we also consider time series of 44 resp. 50 data points (three segments of length 22 and 25), extracted from the daily data stream covering a four-year period (1448 samples). In Figure 13 we show the time and backtrack profiles for finding a first solution.



(a) Time, Size 22 (b) Time, Size 24 (c) Time, Size 25

(d) Backtracks, Size 22 (e) Backtracks, Size 24 (f) Backtracks, Size 25

Fig. 13: Percentage of Problems Solved for 3 Overlapping Segments of Lengths 22, 24, and 25; Execution time in top row, backtracks required in bottom row.

The top row shows the percentage of instances solved within a given time budget, the bottom row shows the percentage of problems solved within a backtrack budget. For easy problems, the *pure* variant finds solutions more quickly, but the *incremental* version pays off for more complex problems, as it reduces the number of backtracks required sufficiently to account for the large overhead of stating and pruning all invariants. The problems for segment length 20 (not shown) can be solved without timeout for both variants, as the segment length increases, the number of timeouts increases much more rapidly for the *pure* variant.

The results show that adding the generated invariants drastically improves the propagation, even for feasible problems. The improvement is due to detecting infeasibility of a generated sub-problem for the remaining suffix of the unassigned variables more rapidly, and therefore avoiding having to explore this infeasible subtree in the overall search.

8 Conclusion

Using the operational view of time-series constraints, i.e. the seed transducers for each regular expression and counter automata, we presented systematic methods for synthesising 1) linear and 2) non-linear invariants linking the result values of several time-series constraints and parameterised by the time-series length, and 3) conditional automata representing a condition on the result value of a time-series constraint. Since all these conditional automata have a number of states and an input alphabet that do not depend on the length of an input sequence, these automata allow us to prove both the fact that linear invariants are facet defining or not, and the validity of non-linear invariants, for *any long enough sequence length*. All the 2000 synthesised parametrised invariants were put in a publicly available database of invariants [3] linked to the time-series catalogue that was used to automatically enhance short-term electricity production models that were acquired from real production data.

References

1. Gautam Appa, Dimitris Magos, and Ioannis Mourtos. LP relaxations of multiple ALL_DIFFERENT predicates. In Jean-Charles Régin and Michel Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, First International Conference, CPAIOR 2004*, volume 3011 of *LNCS*, pages 364–369. Springer, 2004.
2. Ekaterina Arafailova. *Functional Description of Sequence Constraints and Synthesis of Combinatorial Objects*. PhD Thesis, IMT Atlantique, LS2N, Sep 2018.
3. Ekaterina Arafailova, Nicolas Beldiceanu, Rémi Douence, Mats Carlsson, Pierre Flener, María Andreína Francisco Rodríguez, Justin Pearson, and Helmut Simonis. Global Constraint Catalog, Volume ii, Time-Series Constraints. *CoRR*, abs/1609.08925, 2018.
4. Ekaterina Arafailova, Nicolas Beldiceanu, and Helmut Simonis. Generating linear invariants for a conjunction of automata constraints. In Chris Beck, editor, *Principles and Practice of Constraint Programming - CP 2017*, volume 10416 of *LNCS*, pages 21–37. Springer, 2017.
5. Ekaterina Arafailova, Nicolas Beldiceanu, and Helmut Simonis. Deriving generic bounds for time-series constraints based on regular expressions characteristics. *Constraints*, 23(1):44–86, 2018.
6. Ekaterina Arafailova, Nicolas Beldiceanu, Mats Carlsson, Pierre Flener, María Andreína Francisco Rodríguez, Justin Pearson and Helmut Simonis. Systematic derivation of bounds and glue constraints for time-series constraints. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - CP 2016*, volume 9892 of *LNCS*, pages 13–29. Springer, 2016.
7. Ekaterina Arafailova, Nicolas Beldiceanu, Rémi Douence, Pierre Flener, María Andreína Francisco Rodríguez, Justin Pearson and Helmut Simonis. Time-Series Constraints: Improvements and Application in CP and MIP Contexts. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming - CPAIOR 2016*, volume 9676 of *LNCS*, pages 18–34. Springer, 2016.
8. Nicolas Beldiceanu, Mats Carlsson, Romuald Debruyne, and Thierry Petit. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4):339–362, 2005.
9. Nicolas Beldiceanu, Mats Carlsson, Rémi Douence, and Helmut Simonis. Using finite transducers for describing and synthesising structural time-series constraints. *Constraints*, 21(1):22–40, January 2016. Journal fast track of CP 2015: summary on p. 723 of LNCS 9255, Springer, 2015.
10. Nicolas Beldiceanu, Georgiana Ifrim, Arnaud Lenoir and Helmut Simonis. Describing and generating solutions for the EDF unit commitment problem with the ModelSeeker. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - CP 2013*, volume 8124 of *LNCS*, pages 733–748. Springer, 2013.
11. Nicolas Beldiceanu, Mats Carlsson, Jean-Xavier Rampon, and Charlotte Truchet. Graph invariants as necessary conditions for global constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *LNCS*, pages 92–106. Springer, 2005.
12. Nicolas Beldiceanu and Evelyne Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97–123, 1994.
13. Nicolas Beldiceanu, Pierre Flener, Justin Pearson, and Pascal Van Hentenryck. Propagating regular counting constraints. In Carla E. Brodley and Peter Stone, editors, *AAAI 2014*, pages 2616–2622. AAAI Press, 2014.
14. Nicolas Beldiceanu, Carlsson Mats, and Thierry Petit. Deriving filtering algorithms from constraint checkers. In Mark Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004*, volume 3258 of *LNCS*, pages 107–122. Springer, 2004.
15. Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Constraint-based optimization with the minimax decision criterion. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003*, volume 2833 of *LNCS*, pages 168–182. Springer, 2003.
16. Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
17. Nader H. Bshouty, Dana Drachsler-Cohen, Martin Vechev, and Eran Yahav. Learning disjunctions of predicates. In Satyen Kale and Ohad Shamir, editors, *Proceedings of the 2017 Conference on Learning Theory*, volume 65 of *Proceedings of Machine Learning Research*, pages 346–369, Amsterdam, Netherlands, Jul 2017. PMLR.
18. Nader H. Bshouty, Paul W. Goldberg, Sally A. Goldman, and H. David Mathias. Exact learning of discretized geometric concepts. *SIAM J. Comput.*, 28(2):674–699, feb 1999.
19. John Charnley, Simon Colton, and Ian Miguel. Automatic generation of implied constraints. In *ECAI 2006*, volume 141 of *Frontiers in AI and Applications*, pages 73–77. IOS Press, 2006.
20. Zhixiang Chen and Foued Ameur. The learnability of unions of two rectangles in the two-dimensional discretized space. *Journal of Computer and System Sciences*, 59(1):70–83, 1999.
21. COSYTEC. *CHIP Reference Manual*, release 5.1 edition, 1997.
22. Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.
23. Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *ECAI*, pages 290–295, 1988.
24. María Andreína Francisco Rodríguez, Pierre Flener, and Justin Pearson. Implied constraints for Automaton constraints. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015*, volume 36 of *EPiC Series in Computing*, pages 113–126. EasyChair, 2015.

25. María Andreína Francisco Rodríguez, Pierre Flener, and Justin Pearson. Automatic generation of descriptions of time-series constraints. In Alexander Brodsky, editor, *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017*, pages 102–109. IEEE Computer Society, 2017.
26. Simon French, editor. *Decision Theory: An Introduction to the Mathematics of Rationality*. Halsted Press, New York, NY, USA, 1986.
27. Alan Frisch, Ian Miguel, and Toby Walsh. Extensions to proof planning for generating implied constraints. In *9th Symp. on the Integration of Symbolic Computation and Mechanized Reasoning*, 2001.
28. Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972.
29. Pierre Hansen and Gilles Caporossi. Autographix: An automated system for finding conjectures in graph theory. *Electronic Notes in Discrete Mathematics*, 5:158–161, 2000.
30. John N. Hooker. *Integrated Methods for Optimization*. Springer Publishing Company, Incorporated, 2nd edition, 2011.
31. Jon Lee. All-different polytopes. *J. Comb. Optim.*, 6(3):335–352, 2002.
32. Julien Menana. *Automata and Constraint Programming for Personnel Scheduling Problems*. PhD Thesis, Université de Nantes, Oct 2011.
33. Julien Menana and Sophie Demassey. Sequencing and counting with the MULTICOST-REGULAR constraint. In Willem Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009*, volume 5547 of *LNCS*, pages 178–192. Springer, 2009.
34. Gilles Pesant. A filtering algorithm for the STRETCH constraint. In Toby Walsh, editor, *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001*, volume 2239 of *LNCS*, pages 183–195. Springer, 2001.
35. Leonard Jimmie Savage. The theory of statistical decision. *Journal of the American Statistical Association*, 46(253):55–67, 1951.
36. Margus Veanes, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaj Bjørner. Symbolic finite state transducers: algorithms and applications. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012*, pages 137–150. ACM, 2012.