Automatically Mining and Proving Generic Invariants on Integer Sequences

Ekaterina Arafailova[1], Nicolas Beldiceanu[1], and Helmut Simonis[2]

[1] TASC (LS2N), IMT Atlantique, FR – 44307 Nantes, France
FirstName.LastName@imt-atlantique.fr
[2] Insight Centre for Data Analytics, University College Cork, Ireland
Helmut.Simonis@insight-centre.org

Abstract. We describe a method for discovering and proving generic instance-independent invariants linking together several characteristics of an integer sequence; *generic invariants* are independent of the integer values used in the sequence, but are possibly parameterised by the sequence size. Conjunctions of time-series constraints on the same sequence serve as a test case: (1) Experiments show that learning from small sequence sizes still allows identifying most generic invariants of bigger sequences; (2) Experiments also indicate that the discovered invariants speed up, both, the proof of infeasibility, and more surprisingly, the generation of solutions for a conjunction of time-series constraints on a common sequence.

1 Introduction

While artificial intelligence has considered from its very beginning the possibility to automate the process of scientific discovery [22], relatively little work has been carried out in this area [24]. One of the main reasons for this situation is that scientific discovery not only needs to establish conjectures, but also requires to prove or to invalidate (and fix) them. While the human process to deal with proofs and refutation has been analysed in the context of mathematics [21], most computer science work has focused on the first part, namely generating conjectures both for specific domains like graph theory [16], or for more general domains [13,23]. The main reason is that the proof part is a key bottleneck, as it is much more challenging to automate as already observed in [10], even if programs that could prove theorems in propositional or first order logic already exist since the fifties [26]. Nowadays there is a renewed interest in proof assistants like Isabelle [27] or Coq [12]; nevertheless such assistants still require describing and formalising a proof based on human insight, which is typically demanding for proving or invalidating a large set of conjectures. More recently, both in the context of circuit design and program verification, mining Boolean expressions, and Boolean combinations of numerical inequalities were respectively done in [15] and [20]. The later uses a theorem prover to verify the proposed invariants. Another strand of research in automated proving is mining of useful lemmas from a database of lemmas, that can be further used for speeding up automated

proving of theorems [18,17]. In the context of CP, declarative frameworks have been proposed for describing propagators, going back to the work on cc(FD) of [28] to the more recent work of [25]. But in all these approaches, propagators had to be conceived by humans and were restricted to one single constraint.

Our contribution is a methodology for *functional constraints on integer sequences* [6], which both proposes conjectures and proves them automatically by using *constant-size automata*, i.e. automata whose size is independent both from the sequence size and from the values in the sequence. A functional constraint is imposed on an integer sequence $\mathcal{X}$ and an integer variable R, which is functionally determined by $\mathcal{X}$. For a conjunction of two functional constraints $\gamma_1(\mathcal{X}, R_1)$ and $\gamma_2(\mathcal{X}, R_2)$ imposed on the same sequence of n *integer variables* $\mathcal{X}$, our method describes sets of infeasible result values pairs for (R_1, R_2). 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 a set 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

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

In order to prove that (1) is universally true we need to show that for every $f_i(R_1, R_2, n)$, there does not exist an integer sequence of length n yielding R_1 (resp. R_2) as the 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 integer sequences satisfying each elementary condition C_i^j of $f_i(R_1, R_2, n)$ as a constant-size automaton $\mathcal{A}_{i,j}$. Then checking that the intersection of all automata $\mathcal{A}_{i,1}, \mathcal{A}_{i,2}, \ldots, \mathcal{A}_{i,k_i}$ is empty implies that $f_i(R_1, R_2, n)$ is indeed infeasible. Note that such proof scheme is independent from the sequence size n and does not explore any search space.

We use time-series constraints [1,5] as a running example and a test case. This invariant generation process is offline: it is done once and for all in order to build a database of generic invariants. The paper is organised as follows:

- Section 2 provides the required background on time-series constraints and register automata.
- Section 3 motivates this work with a running example, which illustrates the need for deriving invariants.
- Section 4 presents our method for deriving invariants for a conjunction of functional 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 4. Its goal is to generate a dataset, from which we will extract invariants.
  2. A *mining phase* is detailed in Section 4.1. It extracts a hypothesis H of Boolean functions of the form $f_1 \vee f_2 \vee \cdots \vee f_m$ from the generated data.
  3. A *proof phase* is detailed in Section 4.2. 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* sequence size, 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 are noise-free, and that the goal of our work is not to discover statistical properties of functional constraints, but rather to extract mathematical theorems, which are always true.

– Section 5 first evaluates the capability of our method to capture most infeasible pairs by using the data mining phase only on small sequence size from 7 to 12, and by checking on the unseen dataset of sequence size from 13 to 24, whether there are uncovered infeasible pairs (R_1, R_2). Second, it evaluates the effect of adding the obtained invariants to a recent state of the art constraint model [3].

2 Background

A time series here is a sequence of integers that represents measurements taken over time, e.g. the production of a power plant or the temperature in a building. A *time-series constraint* [5] $\gamma(\mathcal{X}, R)$ is a functional constraint, where a sequence of integer variables $\mathcal{X}$ is called a *time series*, and an integer variable R is called the *result variable*. For a time series $\mathcal{X} = \langle X_1, X_2, \ldots, X_n \rangle$, its signature $\langle S_1, S_2, \ldots, S_{n-1} \rangle$ is defined by the following constraints: $(X_i < X_{i+1} \Leftrightarrow S_i = \text{`<'}) \wedge (X_i = X_{i+1} \Leftrightarrow S_i = \text{`='}) \wedge (X_i > X_{i+1} \Leftrightarrow S_i = \text{`>'})$ for all $i \in [1, n-1]$. In this work, we focus only on those constraints such that the value of R depends only on the signature of $\mathcal{X}$, but *not* on the values in $\mathcal{X}$. For example, the time series $\langle 1, 8, 4 \rangle$ is equivalent from this point of view to the time series $\langle 3, 4, 2 \rangle$ since both have their signature being $\langle <, > \rangle$. Hence, such time series would yield the same value of R. More precisely, we consider the two following families of time-series constraints, which both depend on a regular expression σ over the alphabet $\Sigma = \{\text{`<'}, \text{`='}, \text{`>'}\}$:

– NB_$\sigma(\mathcal{X}, R)$, where R is constrained to be the number of maximal words wrt inclusion of the language of σ in the signature of $\mathcal{X}$.
– SUM_WIDTH_$\sigma(\mathcal{X}, R)$, where R is constrained to be the number of elements of X_i that correspond to the maximal words wrt inclusion of the language of σ in the signature of $\mathcal{X}$.

Each time-series constraint has a baseline implementation by a *register automaton* [19] generated from a transducer [5] that is itself synthesised from a regular expression [14]. A *register automaton* $\mathcal{A}$ with $p > 0$ registers 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 registers, $A \subseteq Q$ is the *set of accepting states*, and $\alpha \colon \mathbb{Z}^p \to \mathbb{Z}$ is a function, which maps the registers of an accepting state into an integer. If, by consuming the symbols of a word w in Σ^*, the automaton $\mathcal{A}$ triggers a sequence of transitions from q_0, its initial state,

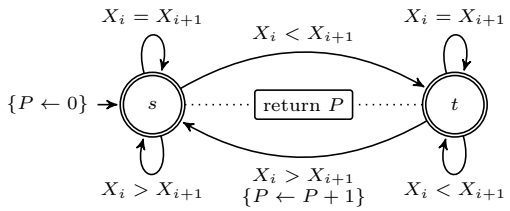

Fig. 1: Register automaton for NB_PEAK

to some accepting state where $\langle D_1, D_2, \ldots, D_p \rangle$ are the values of the registers at this stage, then $\mathcal{A}$ returns $\alpha(D_1, D_2, \ldots, D_p)$, otherwise it *fails*.

A time-series constraint $\gamma(\mathcal{X}, R)$, implemented by a register automaton $\mathcal{A}$, holds iff after consuming the signature of $\mathcal{X} = \langle X_1, X_2, \ldots, X_n \rangle$, $\mathcal{A}$ returns R.

Example 1. Consider the NB_PEAK$(\mathcal{X}, R)$ constraint, where R is restricted to be the number of *peaks* in the sequence $\mathcal{X}$. A *peak* of $\mathcal{X}$ is a maximal subsequence of $\mathcal{X}$ whose signature is a word in the language of the '<(<|=)* (>|=)*>' regular expression. Since $\mathcal{T} = \langle 1, 1, 4, 4, 0, 8, 7, 6, 3, 4 \rangle$ has 2 peaks, NB_PEAK$(\mathcal{T}, 2)$ holds. Figure 1 gives the register automaton $\mathcal{A}$ for the NB_PEAK constraint. The horizontal arrow coming from nowhere indicates the initial state of $\mathcal{A}$. States with double circles are accepting. After consuming the signature $\langle =, <, =, >, <, >, >, >, <\rangle$ of $\mathcal{T}$, $\mathcal{A}$ returns 2. △

3 Motivation and Running Example

Consider a conjunction of time-series constraints $\gamma_1(\mathcal{X}, R_1) \wedge \gamma_2(\mathcal{X}, R_2)$ imposed on the same sequence of integer variables $\mathcal{X}$. In [3], using the representation of γ_1 and γ_2 as *register automata*, they present a method for deriving parameterised linear inequalities linking the values of R_1, R_2. Although, in most cases, the derived inequalities were facet-defining, the experiments revealed 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 solutions. The following example illustrates such a situation.

Example 2. Consider the SUM_WIDTH_DECREASING_SEQUENCE$(\mathcal{X}, R_1)$ and the SUM_WIDTH_ZIGZAG$(\mathcal{X}, R_2)$ time-series constraints on the same sequence $\mathcal{X}$, where a *decreasing sequence* (resp. a *zigzag*) in $\mathcal{X}$ is a subsequence of $\mathcal{X}$ corresponding to an inclusion-wise maximal occurrence of '(>(>|=)*)*>' (resp. '$(<>)^+ < (> |\varepsilon) \mid (><)^+ > (< |\varepsilon)$') in the signature of $\mathcal{X}$, and the width of such a subsequence is the number of its elements. Then R_1 (resp. R_2) is the sum of the widths of decreasing sequences (resp. zigzags) in $\mathcal{X}$. The SUM_WIDTH_ZIGZAG

time-series constraint can be used for limiting the widths of zigzags occurring in the production curve of a power plant [7]. For the sequence size of $\mathcal{X}$ in $\{9, 10, 11, 12\}$, Figure 2 represents 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. △

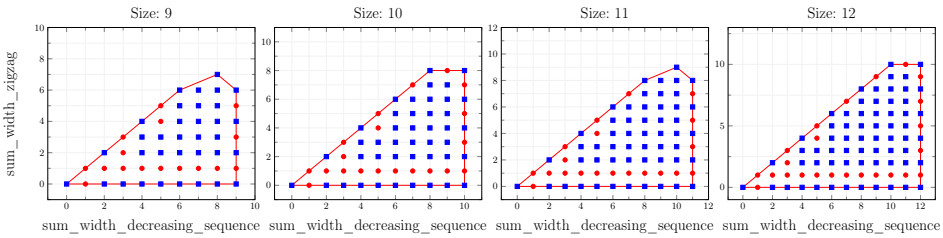

Fig. 2: Blue squares represent feasible combinations of the result variables R_1, R_2 of the SUM_WIDTH_DECREASING_SEQUENCE$(\mathcal{X}, R_1)$ and the SUM_WIDTH_ZIGZAG$(\mathcal{X}, R_2)$ time-series constraints on the same sequence $\mathcal{X}$ whose length is in $\{9, 10, 11, 12\}$; red circles represent infeasible points inside the convex hull of feasible points, while red straight lines depict the facets of the convex hull of feasible points.

Our work focusses on pairs of constraints for the following reasons:

- Given a set of constraints, we want to capture the interaction of constraints without considering an exponential number of subsets of constraints, i.e. the idea is rather to focus on subsets of size 2.
- As we will see in the next section, our technique first computes the convex hull of the set of feasible points. When considering three or more constraints, computing the convex hull of a set of points becomes a difficult task since the number of facets can be exponential.

4 Discovering and Proving Invariants

Consider a conjunction of functional constraints $\gamma_1(\mathcal{X}, R_1)$ and $\gamma_2(\mathcal{X}, R_2)$ imposed on the same sequence of integer variables $\mathcal{X}$. This work focuses on *automatically extracting and proving* invariants that characterise some subsets of infeasible points that are all located inside the convex hull of feasible combinations of R_1 and R_2. Our approach uses three sequential phases.
[GENERATING DATA PHASE] The first phase of our method is a preparatory work, namely *generating data*. For each sequence length n in $[7, 12]$, we generate *all* feasible combinations of the values of R_1 and R_2. For each of the 6 lengths, (i) we compute the convex hull of feasible points of R_1 and R_2 using Graham's

scan, and (ii) we detect the set $\mathcal{I}$ of infeasible combinations of R_1 and R_2 in this convex hull.
[MINING PHASE] The second phase, called the *mining phase*, consists of extracting a hypothesis H describing the set of infeasible points $\mathcal{I}$ from the generated data. We represent this hypothesis as a disjunction of Boolean functions $f_i(R_1, R_2, n)$. The mining phase is described in Section 4.1.
[PROOF PHASE] The third phase, called the *proof phase*, consists of refining the discovered hypothesis H by validating some Boolean functions f_i and by refuting and eliminating others using *constant-size automata* without registers. A refined hypothesis, which is *proved* to be correct in the general case, i.e. for any sequence length, is called a *description* of the set $\mathcal{I}$. The proof phase is described in Section 4.2.

4.1 Mining Phase

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

There is a number of works on learning a disjunction of predicates [8], and some special case, where disjunction corresponds to a geometric concept [9,11]. 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 describe our dataset, which consists of *feasible* and *infeasible* pairs of the result values R_1 and R_2.
- Second, we define the space of concepts, *hypotheses*, we can potentially extract from our dataset.
- Third, we outline the *target hypothesis* for functional constraints, i.e. what we are searching for.
- Finally, we briefly describe the algorithm used for finding the target hypothesis.

Input Dataset We represent our generated data as the union of two sets of triples $\mathcal{D}^-$ (resp. $\mathcal{D}^+$) called the set of feasible (resp. infeasible) examples, such that:

- For every (k, p_1, p_2) (with $k \in [7, 12]$) in $\mathcal{D}^-$, there exists at least one integer sequence of length k that yields p_1 and p_2 as the values of R_1 and R_2, respectively.

– For every (k, p_1, p_2) (with $k \in [7, 12]$) in $\mathcal{D}^+$,
  (i) there does not exist any integer sequence of length k that would yield p_1 and p_2 as the values of R_1 and R_2, respectively.
  (ii) (p_1, p_2) is located within the convex hull of feasible values of R_1 and R_2.

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, called an *atomic relation*, where the main atomic relations are

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

with $j \neq k$ being in $\{1, 2\}$, with c and d being natural numbers, and $\mathtt{up}(R_j, n)$ being the maximum possible value of R_j given the constraint $\gamma_j(\langle X_1, X_2, \ldots, X_n \rangle, R_j)$. The intuition of these atomic relations is now explained:

– (i) stands from the fact that many invariants are only valid for a big enough sequence size.
– (ii) is motivated by the fact that the parity of the sequence size is sometimes relevant.
– (iii) is justified by the fact that the parity of R_j can come into play.
– (iv) and (v) are related to the fact that infeasible points can be located on a ray or on an interval.
– (vi) and (vii) are respectively linked to the fact that quite often infeasible points inside the convex hull are very close to the minimum or the maximum values [4] of R_j (with $j \in [1, 2]$), 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_j and R_k.

Target Hypothesis

Definition 1. *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 2, but the two Boolean functions $R_1 = 13$ and $R_1 = R_2$ are not.

Definition 2. *A Boolean function of $\mathcal{H}$ is* universally true *if it is true for any integer sequence of any length.*

Definition 3. *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 formulae are true.

Mining Algorithm Our mining algorithm filters out all the Boolean functions not consistent with our dataset and returns the disjunction of the remaining Boolean functions. Note that the mining algorithm ignores Boolean functions involving the atomic relation (i) $n \geq c$, which is handled in the proof phase. Remember that we run the algorithm only on the limited dataset $\mathcal{D}$, i.e. the data set generated from the integer sequences of small lengths, which we will be motivated in Section 5.

4.2 Proof Phase

After extracting from $\mathcal{D}$, the target hypothesis $H = f_1 \vee f_2 \vee \cdots \vee f_m$ characterising subsets of infeasible points that are all located inside 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. To do that, the main result of this section, Theorem 1, gives a necessary and sufficient condition for a Boolean function f to be universally true, provided that there exists constant-size automata associated with the atomic relations in f. We will further show how to generate such automata for two types of atomic relations.

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, $R_1 = \mathtt{up}(R_1, n) \wedge R_2 \bmod 2 = 1$ 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.

We first focus on independent Boolean functions, i.e. the ones containing relations from (i) to (vii), and give a necessary and sufficient condition for proving that an independent Boolean function is universally true. Then for a dependent Boolean function containing relation (viii), we describe a method for verifying that the considered dependent Boolean function is universally true.

Proof of Independent Boolean Functions Since most atomic relations are independent, i.e. cases (i) to (vii), this paper focuses primarily on a necessary and sufficient condition for proving that an independent Boolean function is universally true.

Definition 4. *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 an integer sequence satisfying C, whose signature is this word.*

Definition 5. *For an independent Boolean function* $f = C_1 \wedge C_2 \wedge \cdots \wedge C_p$ *of* $\mathcal{S}$*, 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 sequence length, and thus the set $\mathcal{T}_f$ is empty.

For any atomic relation C from (i) to (vii), i.e. independent atomic relation, the corresponding set of supporting signatures is represented as the language of a constant-size automaton without registers $\mathcal{A}_C$. Constant size means that the number of states of this automaton does not depend on the length of the input integer sequence. 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 $\mathcal{A}_{C_i}$ with i in $[1, p]$. This provides a necessary and sufficient condition for proving that a Boolean function f is universally true.

Theorem 1. *Consider two functional constraints* $\gamma_1(\mathcal{X}, R_1)$ *and* $\gamma_1(\mathcal{X}, R_2)$ *on the same sequence* $\mathcal{X}$*, 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 without registers* $\mathcal{A}_{C_i}$*. The function* f *is universally true iff the intersection of all automata for* $\mathcal{A}_{C_i}$ *(with* $i \in [1, p]$*) is empty.*

The proof of Theorem 1 follows from Definitions 4 and 5.

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.

Generation of a Constant-Size Automaton for Independent Atomic Relations Generating a constant-size automaton for atomic relations that impose a restriction like (i) or (ii) on the sequence length is straightforward. For space reason we now focus on the generation of constant-size automata for atomic relations of the form (vi) $R_j = c$ and (iii) $R_j \bmod c = d$ from register automata of γ_1 and γ_2. Let us start with an illustrating example.

Example 3. Consider the SUM_WIDTH_DECREASING_SEQUENCE$(\mathcal{X}, R)$ time-series constraint, whose register automaton is given in Part (A) of Figure 3, and the atomic relation C defined by $R = 3$. The minimal automaton representing the atomic relation C is given in Part (C) of Figure 3. △

In order to generate constant-size automata for the atomic relations (iii) and (iv) we require that for each γ_j there exists a register automaton $\mathcal{A}_j$ with at most two registers A_1 and A_2 satisfying the following conditions:

1. The initial values of both registers are 0.
2. On every transition of $\mathcal{A}_j$, A_1 is either incremented by a natural number, or by the current value of A_2 plus a natural number. In this latter case A_2 is reset to 0.

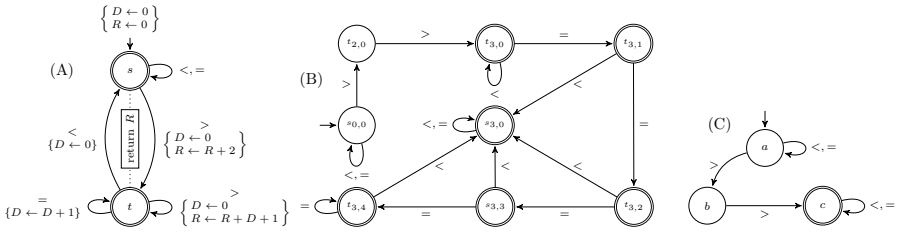

Fig. 3: (A) Register automaton for SUM_WIDTH_DECREASING_SEQUENCE$(\mathcal{X}, R)$; (B) Automaton for the atomic relation $R = 3$ and (C) corresponding minimised automaton.

3. On every transition of $\mathcal{A}_j$, A_2 is either incremented by a natural number, or reset to 0.
4. The acceptance function of $\mathcal{A}_j$ returns the last value of A_1.

For every NB_σ and SUM_WIDTH_σ time-series constraint, there exists a register automaton [2] satisfying Conditions (1)–(4).

We now explain how to construct constant-size automata for the relations (vi) $R_j = c$ and (iii) $R_j \bmod c = d$ when $\mathcal{A}_j$ satisfies Conditions (1)–(4).

Construction of Constant-Size Automata for R = c. Consider the atomic relation C defined by $R = c$. The automaton $\mathcal{A}_C$ is built from $\mathcal{A}$ in the following way:

[STATES OF $\mathcal{A}_C$] For every state q of $\mathcal{A}$, there are $c \cdot (c+1)$ states in $\mathcal{A}_C$ named q_{i_1,i_2} (with $i_1 \in [0, c], i_2 \in [0, c+1]$), i.e. each state of $\mathcal{A}_C$ corresponds to a state of $\mathcal{A}$ labelled by the potential values of the registers A_1 and A_2, with the restriction that values of A_2 that are strictly greater than c are all mapped to $c+1$.

[INITIAL STATE OF $\mathcal{A}_C$] Since both registers A_1 and A_2 of $\mathcal{A}$ are initialised to 0, the state $q_{0,0}$ is the initial state of $\mathcal{A}_C$.

[ACCEPTING STATES OF $\mathcal{A}_C$] All states where the returned value A_1 is set to the expected value c are accepting states of $\mathcal{A}_C$. Consequently, all states of the form q_{c,i_2} (with $i_2 \in [0, c+1]$) are accepting.

[TRANSITIONS OF $\mathcal{A}_C$]

- Within $\mathcal{A}_C$, there is a transition from q_{i_1,i_2} to $q^*_{i_1^*,i_2^*}$ (with $i_1, i_1^*, i_2^* \in [0, c], i_2 \in [0, c+1]$) labelled with s, a letter in the input alphabet of $\mathcal{A}$, if there exists in $\mathcal{A}$ a transition from q to q^* labelled with s such that, when A_1 and A_2 have values i_1 and i_2, A_1 and A_2 are respectively set to i_1^* and i_2^* when triggering that transition.
- Within $\mathcal{A}_C$, there is a transition from q_{i_1,i_2} to $q^*_{i_1^*,c+1}$ labelled with s, a letter in the input alphabet of $\mathcal{A}$, if there exists in $\mathcal{A}$ a transition from q to q^* labelled with s such that, when A_1 and A_2 have values i_1 and i_2, A_1 and A_2 are respectively set to i_1^* and $i_2^* > c$ when triggering that transition.

It is easy to see that when $\mathcal{A}$ satisfies Conditions (1)–(4) the set of signatures recognised by the constructed automaton $\mathcal{A}_C$ is indeed the set of supporting signatures $\mathcal{T}_C$.

Turning back to Example 3, Part (B) of Figure 3 shows the automaton built by the previous algorithm for the relation $R = 3$, ignoring isolated states.

Construction of a Constant-Size Automaton for R mod c = d. Consider the atomic relation C defined by $R \bmod c = d$. The automaton $\mathcal{A}_C$ is built from $\mathcal{A}$ in the following way:
[STATES OF $\mathcal{A}_C$] For every state q of $\mathcal{A}$, there are c^2 states in $\mathcal{A}_C$ named q_{i_1,i_2} (with $i_1, i_2 \in \{0, 1, \ldots, c-1\}$), i.e. each state of $\mathcal{A}_C$ corresponds to a state of $\mathcal{A}$ labelled by the potential values of remainder of the registers A_1 and A_2.

[INITIAL STATE OF $\mathcal{A}_C$] Since both registers A_1 and A_2 of $\mathcal{A}$ are initialised to 0, the state $q_{0,0}$ is the initial state of $\mathcal{A}_C$.

[ACCEPTING STATES OF $\mathcal{A}_C$] All states where the remainder of the returned value A_1 is set to the expected value d are accepting states of $\mathcal{A}_C$. Consequently, all states of the form q_{d,i_2} (with $i_2 \in \{0, 1, \ldots, c-1\}$) are accepting.

[TRANSITIONS OF $\mathcal{A}_C$] Within $\mathcal{A}_C$, there is a transition from q_{i_1,i_2} to $q^*_{i_1^*,i_2^*}$ (with $i_1, i_2, i_1^*, i_2^* \in \{0, 1, \ldots, c-1\}$) labelled with s, a letter in the input alphabet of $\mathcal{A}$, if there exists in $\mathcal{A}$ a transition from q to q^* labelled with s such that, when the remainder modulo c of A_1 (resp. A_2) is i_1 (resp. i_2), the remainder modulo c of A_1 (resp. A_2) becomes i_1^* (resp. i_2^*) after triggering that transition.

It is easy to see that when $\mathcal{A}$ satisfies Conditions (1)–(4) the set of signatures recognised by the constructed automaton $\mathcal{A}_C$ is indeed $\mathcal{T}_C$.

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 [3].

Consider two constraints $\gamma_1(\mathcal{X}, R_1)$ and $\gamma_2(\mathcal{X}, R_2)$ on the same integer sequence $\mathcal{X}$ such that, for both γ_1 and γ_2, the method of [3] for generating linear invariants applies. 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 three following steps:

1. **Assumption.** Assume that there exists an integer sequence 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 a 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

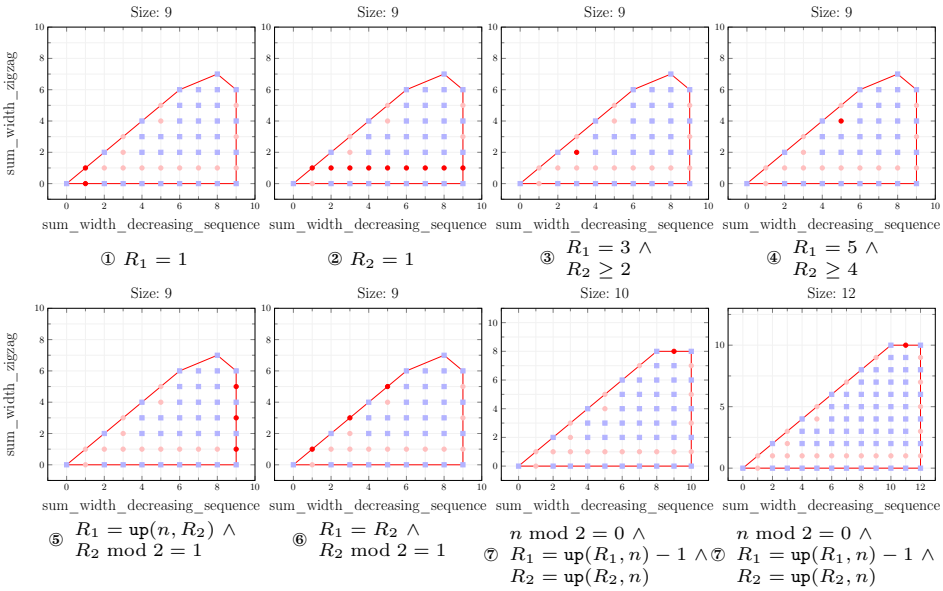

Fig. 4: Seven groups of infeasible combinations of R_1 and R_2, where R_1 and R_2 are, respectively, constrained by SUM_WIDTH_DECREASING_SEQUENCE$(\mathcal{X}, R_1)$ and SUM_WIDTH_ZIGZAG$(\mathcal{X}, R_2)$ on the same sequence $\mathcal{X}$ of length 9 (all plots on top and the two plots on bottom left) and of lengths 10 and 12 (the two plots on bottom right).

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 [3] produces an invariant $R_1 - d \cdot R_2 \geq c$ with c being strictly greater than 1.

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 [3] did indeed generate a desired linear invariant, which proved that the considered Boolean function is universally true.

Example 4. Consider the SUM_WIDTH_DECREASING_SEQUENCE$(\mathcal{X}, R_1)$ and the SUM_WIDTH_ZIGZAG$(\mathcal{X}, R_2)$ time-series constraints on the same sequence $\mathcal{X}$, introduced in Example 2. 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 $R_1 = \mathtt{up}(R_1, n) \ \wedge \ R_2 \bmod 2 = 1$ ⑥ $R_1 \bmod 2 = 1 \ \wedge \ R_1 = R_2$
  ⑦ $n \bmod 2 = 0 \ \wedge \ \ R_1 = \mathtt{up}(R_1, n) - 1 \ \wedge \ R_2 = \mathtt{up}(R_2, n)$

All four upper plots and the two lower plots on the left of Figure 4 contain the groups of infeasible points corresponding to the Boolean functions from ① to ⑥ for n being 9. The two lower plots on the right of Figure 4 contain the infeasible points 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. For example, the automata for both atomic relations of the ⑦ are given in Figure 5. One can take their intersection to check it is empty.

In order to prove 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 presented by an automaton with or without registers satisfying the required properties of the method of [3], 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 $\mathtt{up}(R_1, n)$ as the value of R_1, every occurrence of ZIGZAG in its signature must start and end with '>', and the width of every word in 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 $\mathtt{up}(R_2, n)$ 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, $\mathtt{up}(R_1, n) - 1 = n - 1$ is odd, when n is even. △

5 Evaluation

Consider a conjunction of two time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$, imposed on the same sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$. After performing the

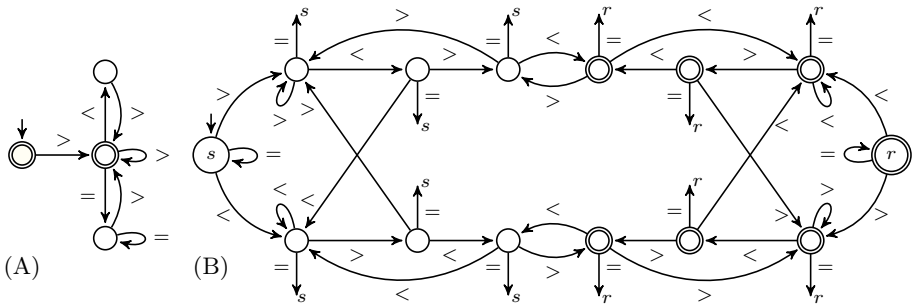

Fig. 5: (A) Automaton for the $R_1 = \texttt{up}(R_1, n)$ atomic relation, where R_1 is constrained by SUM_WIDTH_DECREASING_SEQUENCE$(\mathcal{X}, R_1)$. (B) Automaton for the $R_2 \bmod 2 = 1$ atomic relation, where R_2 is constrained by SUM_WIDTH_ZIGZAG$(\mathcal{X}, R_2)$.

mining and proof phases, we obtain a disjunction describing a subset of infeasible combinations of R_1 and R_2. Recall that this disjunction is called a *description of infeasible set.* The exploitation phase includes the two following procedures:

First, we filter the Boolean functions in the obtained description of infeasible set in order to obtain a *non-dominated description*, i.e. a disjunction of Boolean functions that are mutually non subsumable.

Second, we evaluate the obtained description of infeasible points from two perspectives:

- While the description of infeasible set is correct for any sequence size, it is unclear whether learning from small sequence size allows to also identify all infeasible combinations of R_1 and R_2 for larger sequence sizes. We investigate this question empirically by comparing the set of infeasible combinations of R_1 and R_2 learned by only using small sequence sizes (from 7 to 12) to the set of infeasible combinations of R_1 and R_2 generated by a systematic procedure for larger sequence sizes (from 13 to 24). Such choice was motivated by the following observations:
  - Learning from very small sequence sizes does not make sense since the corresponding convex hulls are too small: many infeasible points do not show up with very small sequence sizes.
  - Learning from very few sequence sizes, e.g. two, is not enough since some infeasible points within the convex hull are only present when $n \bmod k = r$ with $r \in [0, k-1]$ for some small k greater than 2.
  - In practice generating the data for sizes 7 to 12 was very fast, while generating additional data from 13 to 24 took more than one month.
- We evaluate the impact of our learned description of infeasible set in terms of time and number of backtracks for finding a solution or proving infeasibility for a conjunction of time-series constraints.

We consider all pairs of constraints for which infeasible points exist in the convex hull of feasible points, and for which we have the full baseline implemen-

tation. For the 303 pairs considered, there are 68,145 feasible points and 12,103 infeasible points in the training set. From these points we generate 16,310 hypotheses, of which 11,827 are proven. Removing dominated invariants, we are left with 517 non-dominated, proven invariants which are then used in the evaluation. It takes 10 minutes 29 seconds to create once and for all our database of invariants, i.e. to generate the hypotheses, to prove them, and to find the non-dominated set. Each generated invariant consists typically of a disjunction of no more than five conditions.

We use the generated invariants in our test data (sizes 13 to 24), by adding them to a baseline consisting of the previous state-of-the-art implementation based on [3]. Table 1 compares the baseline to our improved method. We checked independently that for the test data set there are 559,224 feasible points, and 50,823 infeasible points. For each test case, we either find the first feasible solution, or show that no solution exists. The results show that only 130 infeasible points (0.26 % of all infeasible points) in the test set are not covered by one of the generated hypotheses.

Table 1: Comparing the state-of-the-art baseline and the baseline with the generated invariants

Measure	Case	Success	Failure
Backtrack	Baseline	289,321,218	465,049,474
Backtrack	New	190,452,242	1,954
Backtrack	%New/Base	65.83	0.00042
Time	Baseline	107,630	89,800
Time	New	78,521	0.7
Time	%New/Base	72.95	0.00078

As we can see, the generated invariants cover the infeasible points nearly perfectly, reducing the time spent from 89,800 seconds to less than one second. Perhaps more surprisingly, the generated invariants also help with feasible cases, by removing infeasible subtrees from the search of feasible solutions. The number of backtracks for the feasible cases is reduced by one third, and the time for finding the solutions is reduced by 27%.

6 Conclusion

For time series on integer sequences, this paper proposes a systematic approach to extract and prove invariants denoting infeasible combinations of pairs of sequence characteristics. To avoid being instance specific these invariants are parameterised by the sequence size.

The approach relies on the fact that infeasible pairs are quite often located at a small distance from the envelope of the convex hull of all feasible pairs, and can therefore be described by intersecting constant-size finite automata.

While we still use generated datasets to extract our invariants, this line of work contributes to explainable AI, since each obtained proved invariant can be fully concisely described as a conjunction of constant-size structured automata.

References

1. Arafailova, E., Beldiceanu, N., Douence, R., Carlsson, M., Flener, P., Rodríguez, M.A.F., Pearson, J., Simonis, H.: Global constraint catalog, volume ii, time-series constraints. CoRR **abs/1609.08925** (2016), http://arxiv.org/abs/1609.08925
2. Arafailova, E., Beldiceanu, N., Douence, R., Flener, P., Francisco Rodríguez, M.A., Pearson, J., Simonis, H.: Time-series constraints: Improvements and application in CP and MIP contexts. In: Quimper, C.G. (ed.) CP-AI-OR 2016. LNCS, vol. 9676, pp. 18–34. Springer (2016)
3. Arafailova, E., Beldiceanu, N., Simonis, H.: Generating linear invariants for a conjunction of automata constraints. In: Beck, C. (ed.) Principles and Practice of Constraint Programming - CP 2017, 23rd International Conference, CP 2017. pp. 21–37. LNCS, Springer International Publishing (2017)
4. Arafailova, E., Beldiceanu, N., Simonis, H.: Deriving generic bounds for time-series constraints based on regular expressions characteristics. Constraints **23**(1), 44–86 (Jan 2018). https://doi.org/10.1007/s10601-017-9276-z
5. Beldiceanu, N., Carlsson, M., Douence, R., Simonis, H.: Using finite transducers for describing and synthesising structural time-series constraints. Constraints **21**(1), 22–40 (January 2016), journal fast track of CP 2015: summary on p. 723 of LNCS 9255, Springer, 2015
6. Beldiceanu, N., Carlsson, M., Flener, P., Pearson, J.: On the reification of global constraints. Constraints **18**(1), 1–6 (January 2013)
7. Beldiceanu, N., Ifrim, G., Lenoir, A., Simonis, H.: Describing and generating solutions for the EDF unit commitment problem with the ModelSeeker. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 733–748. Springer (2013)
8. Bshouty, N.H., Drachsler-Cohen, D., Vechev, M., Yahav, E.: Learning disjunctions of predicates. In: Kale, S., Shamir, O. (eds.) Proceedings of the 2017 Conference on Learning Theory. Proceedings of Machine Learning Research, vol. 65, pp. 346–369. PMLR, Amsterdam, Netherlands (07–10 Jul 2017)
9. Bshouty, N.H., Goldberg, P.W., Goldman, S.A., Mathias, H.D.: Exact learning of discretized geometric concepts. SIAM J. Comput. **28**(2), 674–699 (Feb 1999). https://doi.org/10.1137/S0097539794274246
10. Charnley, J.W., Colton, S., Miguel, I.: Automatic generation of implied constraints. In: ECAI 2006. Frontiers in AI and Applications, vol. 141, pp. 73–77. IOS Press (2006)
11. Chen, Z., Ameur, F.: The learnability of unions of two rectangles in the two-dimensional discretized space. Journal of Computer and System Sciences **59**(1), 70 – 83 (1999). https://doi.org/10.1006/jcss.1999.1621
12. Coquand, T., Huet, G.P.: Constructions: A higher order proof system for mechanizing mathematics. In: Buchberger, B. (ed.) EUROCAL '85, European Conference on Computer Algebra, Linz, Austria, April 1-3, 1985, Proceedings Volume 1: Invited Lectures. LNCS, vol. 203, pp. 151–184. Springer (1985)
13. Fajtlowicz, S.: On Conjectures of Graffiti. Annals of Discrete Mathematics **38**, 113–118 (1988)
14. Francisco Rodríguez, M.A., Flener, P., Pearson, J.: Automatic generation of descriptions of time-series constraints. In: Brodsky, A. (ed.) ICTAI 2017. IEEE Computer Society (2017)
15. Goel, N., Hsiao, M.S., Ramakrishnan, N., Zaki, M.J.: Mining Complex Boolean Expressions for Sequential Equivalence Checking. In: Proceedings of the 19th IEEE Asian Test Symposium, ATS 2010, 1-4 December 2010, Shanghai, China. pp. 442–447. IEEE Computer Society (2010)

16. Hansen, P., Caporossi, G.: Autographix: An automated system for finding conjectures in graph theory. Electronic Notes in Discrete Mathematics **5**, 158–161 (2000)
17. Kaliszyk, C., Chollet, F., Szegedy, C.: Holstep: a machine learning dataset for higher-order logic theorem proving (2017)
18. Kaliszyk, C., Urban, J.: Lemma mining over hol light. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning. pp. 503–517. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
19. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**(2), 329–363 (1994). https://doi.org/10.1016/0304-3975(94)90242-9
20. Krishna, S., Puhrsch, C., Wies, T.: Learning Invariants using Decision Trees. CoRR **abs/1501.04725** (2015), http://arxiv.org/abs/1501.04725
21. Lakatos, I.: Proofs and Refutations. Cambridge University Press (1976)
22. Langley, P.W., Simon, H.A., Bradshaw, G., Zytkow, J.M.: Scientific Discovery – Computational Explorations of the Creative Process. MIT Press (1987)
23. Larson, C.E., Cleemput, N.V.: Automated conjecturing I: Fajtlowicz's Dalmatian heuristic revisited. Artif. Intell. **231**, 17–38 (2016)
24. Lenat, D.B.: On automated scientific theory formation: a case study using the AM program. Machine intelligence **9**, 251–286 (1979)
25. Monette, J.N., Flener, P., Pearson, J.: A propagator design framework for constraints over sequences. In: Brodley, C.E., Stone, P. (eds.) AAAI 2014. pp. 2710–2716. AAAI Press (2014)
26. Newell, A., Simon, H.A.: The logic theory machine – A complex information processing system. IRE Transactions on Information Theory **2**(3), 61–79 (1956)
27. Paulson, L.C.: The foundation of a generic theorem prover. J. Autom. Reasoning **5**(3), 363–397 (1989)
28. Van Hentenryck, P., Saraswat, V., Deville, Y.: Design, implementation, and evaluation of the constraint language cc(FD). Tech. Rep. CS-93-02, Brown University, Providence, USA (January 1993), revised version in *Journal of Logic Programming* 37(1–3):293–316, 1998. Based on the unpublished manuscript *Constraint Processing in cc(FD)*, 1991.