Beyond the Holy Grail – Automatically Generating Constraint Propagators for Conjunctions of Time-Series Constraints

Ekaterina Arafailova, **Nicolas Beldiceanu**, and Helmut Simonis

24th November 2017

1ère journée CAVIAR

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom
LS2N
cnrs
UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh
Insight

The Question Motivating this Work

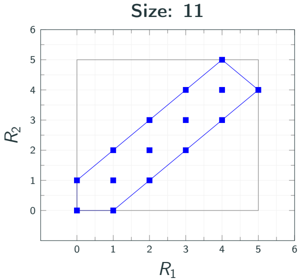
Consider two constraints
$\gamma_1(\langle X_1, X_2, \ldots, X_n \rangle, R_1) \wedge \gamma_2(\langle X_1, X_2, \ldots, X_n \rangle, R_2)$,

where R_1 and R_2 are constrained to be
the result of some computations over $\langle X_1, X_2, \ldots, X_n \rangle$
depending **only** on the relations $<, =, >$ between X_i and X_{i+1}.

For example,
R_1 is the number of peaks in $\langle X_1, X_2, \ldots, X_n \rangle$ and
R_2 is the number of valleys in $\langle X_1, X_2, \ldots, X_n \rangle$.

What is the set of feasible pairs of R_1 and R_2?

Example of Sets of Feasible Pairs of R_1 and R_2: Convex Case

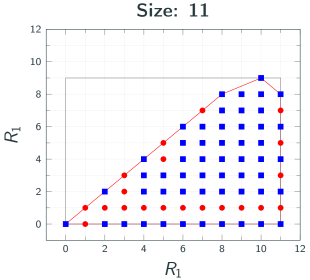


$\gamma_1 = \text{nb\_peak}$

$\gamma_2 = \text{nb\_valley}$

- The set of feasible (blue) points is convex.
- Characterised by a set of **parametrised** linear inequalities (where R_1, R_2 are the variables and n the parameter)

Example of Sets of Feasible Pairs of R_1 and R_2: Non-Convex Case



$\gamma_1 =$ sum_width_decreasing_sequence

$\gamma_2 =$ sum_width_zigzag

- The set of feasible (blue) points is non-convex.
- A conjunction of linear inequalities of is **not enough**.
- Need also for a **non-linear** characterisation.

Two Emerging Problems for Characterising Infeasible Combinations

1. Generate linear inequalities depending on R_1, R_2 and parameterised by $f(n) \in \{n, n \mod p, \sqrt{n}, \dots\}$, which represent the facets of the convex hull.
2. Generate non-linear parameterised invariants eliminating infeasible points on (or inside) the convex hull.

Two Emerging Problems for Characterising Infeasible Combinations

1. Generate linear inequalities depending on R_1, R_2 and parameterised by $f(n) \in \{n, n \mod p, \sqrt{n}, \ldots\}$, which represent the facets of the convex hull.
2. Generate non-linear parameterised invariants eliminating infeasible points on (or inside) the convex hull.

How to solve these two problems in a systematic way for a large family of constraints?

Main Insight $\cdots$

Two Emerging Problems for Characterising Infeasible Combinations

1. Generate linear inequalities depending on R_1, R_2 and parameterised by $f(n) \in \{n, n \mod p, \sqrt{n}, \ldots\}$, which represent the facets of the convex hull.
2. Generate non-linear parameterised invariants eliminating infeasible points on (or inside) the convex hull.

How to solve these two problems in a systematic way for a large family of constraints?

Main Insight $\cdots$

Use **register automata** and **parameterised** characterisation.

Take-Away Message

- **Convex Case:**
  - A **compositional** way of generating cuts from register automata [**CP17implied**].
- **Non-Convex Case:**
  - **Data Mining** for generating **conjectures**,
  - **Proof** using **transducers** and **automata**.

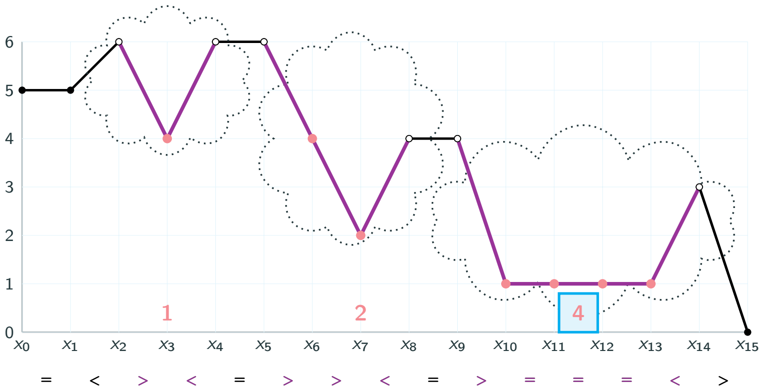
Case Study: Time-Series Constraints

- Described by:
  - Declaratively : **quantitative regular expressions**,
  - Operationally: **finite transducers**.
- Baseline implementation as **register automata**.
- Missing propagation for conjunction of constraints.

Work on improving propagators
for **all** constraints at the same time.

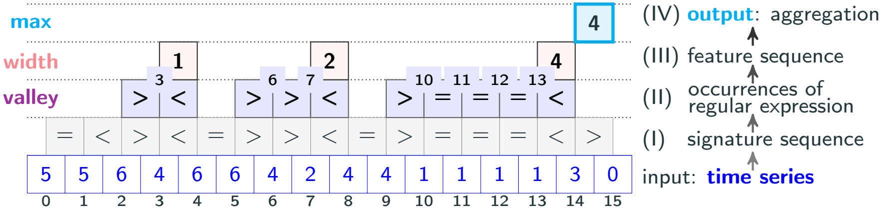
Example of a Time-Series Constraint

Constrain the maximum of the widths of the valleys
in the time series $X = \langle 5, 5, 6, 4, 6, 6, 4, 2, 4, 4, 1, 1, 1, 1, 3, 0 \rangle$.



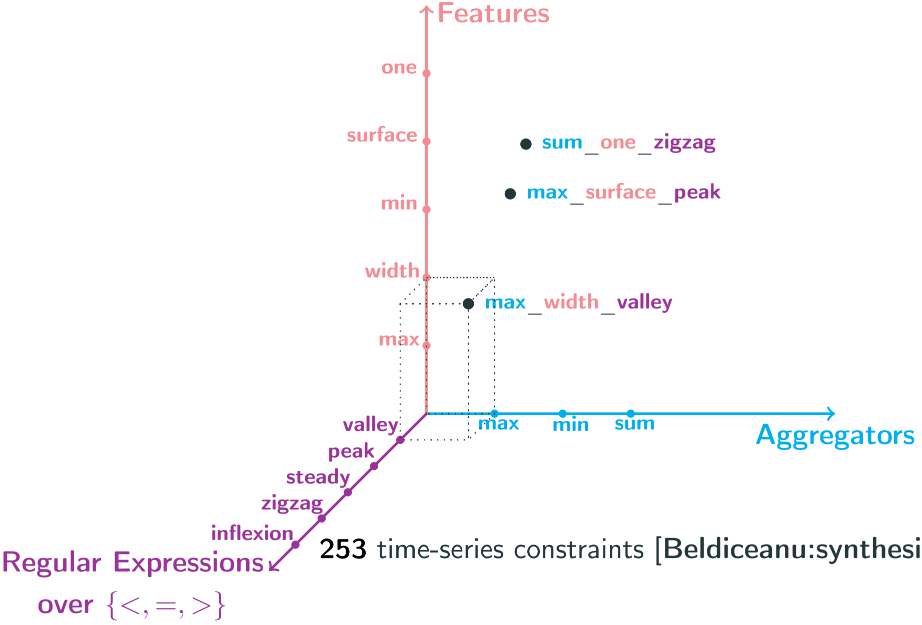
A subsequence $\langle X_i, \ldots, X_j \rangle$ of $\langle X_0, \ldots, X_m \rangle$ is a **valley** if the signature of $\langle X_{i-1}, \ldots, X_{j+1} \rangle$ is a maximal word matching **'>(>|=)*(<|=)*<'**.

Compositional Time-Series Definition by Multiple Layers of Functions



$$\text{max\_width\_valley}(\langle 5,5,6,4,6,6,4,2,4,4,1,1,1,1,3,0\rangle, 4)$$

Space of Time-Series Constraints



253 time-series constraints [**Beldiceanu:synthesis**]

Time-Series Constraints Families of This Work

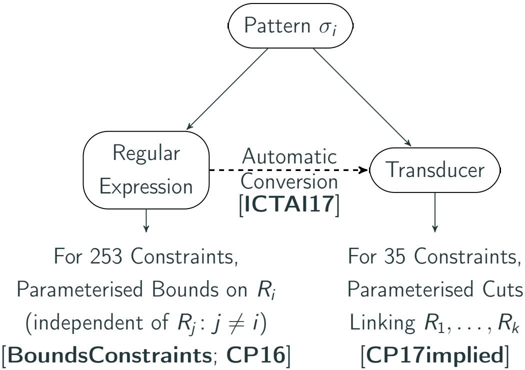
- Only topological constraints,
  i.e. nb_$\sigma(X, R)$ and sum_width_$\sigma(X, R)$
  (R depends only on the relations $<, =, >$
  between consecutive X variables).

- Representation as register automata
  with linear register updates.

- 35 constraints in the two families.

Synthesis of Services (Parameterised Bounds and Cuts)

$$g_1\_f_1\_\sigma_1(X, R_1) \wedge \cdots \wedge g_k\_f_k\_\sigma_2(X, R_2),\ X = \langle X_1, X_2, \ldots, X_n \rangle$$



Example of Obtained Bounds and Generated Invariants for a Conjunction of Two Constraints

$\text{nb\_peak}(X, R_1) \wedge \text{nb\_valley}(X, R_2)$ with $X = \langle X_1, X_2, \ldots, X_n \rangle$, $n \geq 2$

Bounds obtained from a generic formula for nb_σ:

$0 \leq R_1 \leq \left\lfloor \frac{n-1}{2} \right\rfloor$

$0 \leq R_2 \leq \left\lfloor \frac{n-1}{2} \right\rfloor$

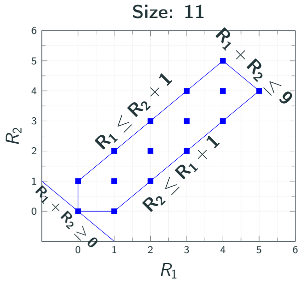
Generated cuts:

$R_2 \leq R_1 + 1$

$R_1 \leq R_2 + 1$

$R_1 + R_2 \leq n - 2$

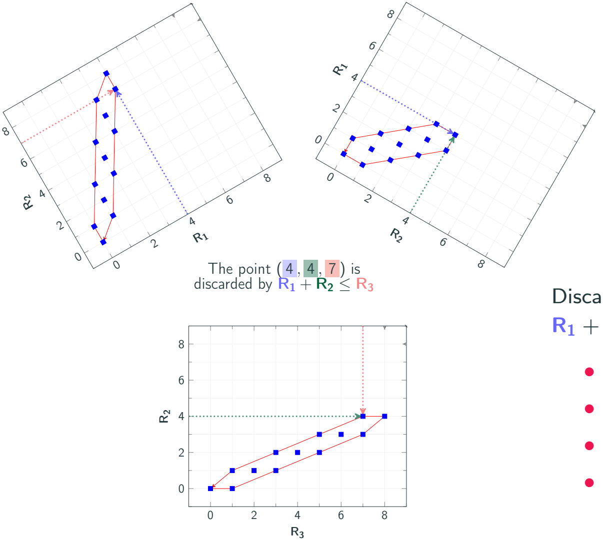
$R_1 + R_2 \geq 0$



Bounds are sharp and 3 out of the 4 found inequalities are facet-defining!

Example of a Generated Invariant for a Conjunction of Three Constraints

$$\text{nb\_peak}(X, R_1) \wedge \text{nb\_valley}(X, R_2) \wedge \text{nb\_inflexion}(X, R_3)$$



Discarded by $R_1 + R_2 \le R_3$:

- $(4, 4, 7)$
- $(3, 3, 5)$
- $(2, 2, 3)$
- $(1, 1, 1)$

Generating Non-Linear Invariants that Deal With Missing, Infeasible Cases

Three Phases of our Method:

1. **Generation of Data:** generate **all feasible combinations** of $R_1, R_2, \ldots, R_k$ for a given range of n values.
2. **Mining Phase:** generate **hypothesis** covering subsets of infeasible points using the generated data.
3. **Proving Phase:** **prove** the generated hypothesis using transducers and automata.

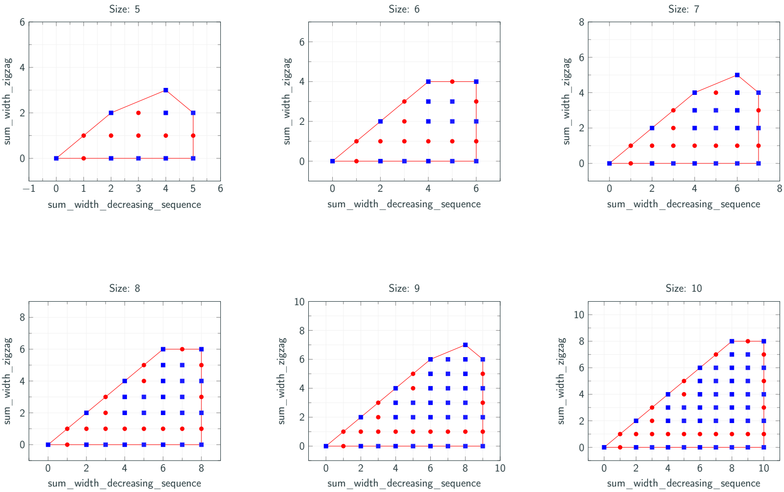
The three phases are **offline**.

Generation of Data

- Pairs of different time-series constraints $\gamma_1(\langle X_1, X_2, \ldots, X_n \rangle, R_1)$ and $\gamma_2(\langle X_1, X_2, \ldots, X_n \rangle, R_2)$.
- Generate **all feasible pairs** (R_1, R_2) for $n \in \{1, 2, \ldots, 12\}$.
- Compute the **convex hull** using Graham's scan.
- Collect **all infeasible points** inside the convex hull.

Example of Samples of Generated Data

$\gamma_1 = \text{sum\_width\_decreasing\_sequence}$, $\gamma_2 = \text{sum\_width\_zigzag}$

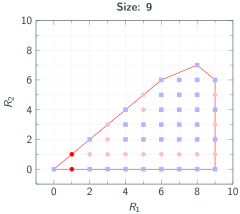


Mining Phase: Generation of Hypothesis

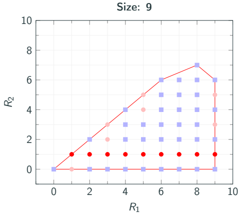
- Consider only samples of sizes from 7 to 12.
- Hypothesis of type $C_1 \wedge C_2 \wedge \cdots \wedge C_p$
  to cover infeasible points inside the convex hull.
- Every C_k is a relation from our bias.
- Examples of relations in our bias:
  - $R_i = c$, $c \in \mathbb{Z}$,
  - $R_i = upper\_bound(R_i, n)$,
  - R_i is odd (even),
  - $R_i = R_j$.

Every infeasible point on/inside the convex hull
must be covered by at least one hypothesis.

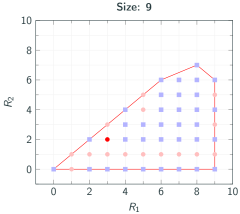
Mining Phase: Example



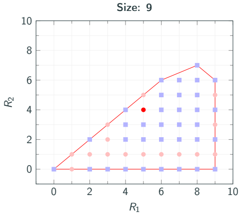
(Group ①) $R_1 = 1$



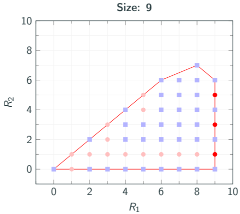
(Group ②) $R_2 = 1$



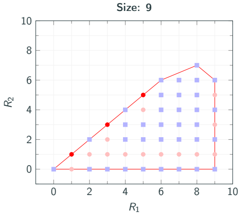
(Group ③) $R_1 = 3 \wedge R_2 = 2$



(Group ④) $R_1 = 5 \wedge R_2 = 4$



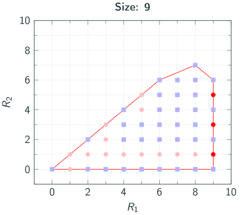
(Group ⑤) $R_1 = up(R_1, n) \wedge R_2 \bmod 2 = 1$



(Group ⑥) $R_1 = R_2 \wedge R_1 \bmod 2 = 1$

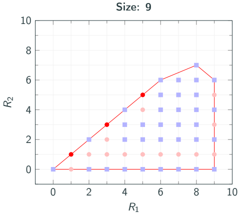
Classification of Groups of Points

1. **Independent Groups**: $H = C_1 \wedge C_2 \wedge \cdots \wedge C_p$, every C_k depends only on one R_i.
2. **Dependent Groups**: $H = C_1 \wedge C2 \wedge \cdots \wedge C_p$, at least one C_k depends on more than one R_i.



(Group ⑤) $R_1 = up(R_1, n) \wedge$
$R_2 \bmod 2 = 1$

Independent Group



(Group ⑥) $R_1 = R_2 \wedge$
$R_1 \bmod 2 = 1$

Dependent Group

The proof scheme depends on the group type!

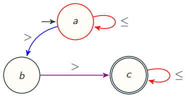
Proving Phase: Independent Groups

- For every hypothesis $C_1 \wedge C_2 \wedge \cdots \wedge C_p$,
  generate a **constant size automaton** for each C_i relation.
- Do the **intersection** of the automata for all $C_1, C_2, \ldots C_p$.
- The intersection is an automaton that recognises **all and only** sequences satisfying the conjunction $C_1 \wedge C_2 \wedge \cdots \wedge C_p$.
- If the intersection is empty,
  then $C_1 \wedge C_2 \wedge \cdots \wedge C_p$ is not feasible
  else generate a counter example to **refine the hypothesis**.

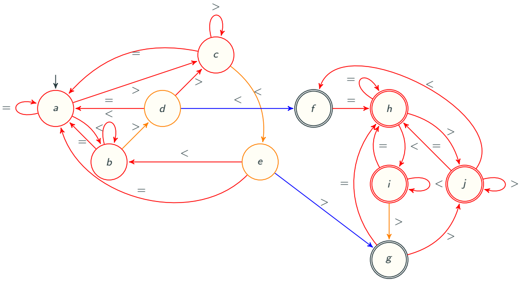
Proving Phase: Independent Group Example

sum_width_decreasing_sequence$(X, R_1) \wedge$ sum_width_zigzag(X, R_2)

An independent group is described by $R_1 = 3 \wedge R_2 = 2$



$R_1 = 3$



$R_2 = 2$

The intersection of two automata is **empty**!

The combination $R_1 = 3$ and $R_2 = 2$ is indeed **infeasible**.

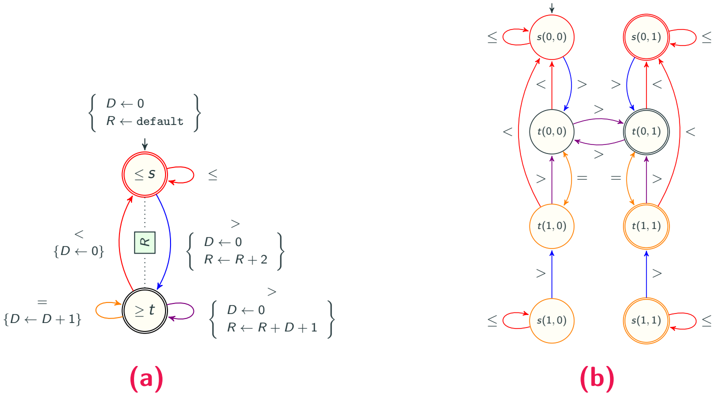
Systematic Generation of Automata for Proving Independent Groups

For two considered families of time-series constraints, we can generate systematically automata for:

- $R_i = c$, $c \in \mathbb{Z}$,
- $R_i = up(R_i, n) - c$, $c \geq 0 \in \mathbb{Z}$, and γ_i is nb_σ,
- $R_i = up(R_i, n)$, and γ_i is sum_width_σ,
- R_i is odd/even.

Example of Automaton for the ‘R is odd’ Rule

sum_width_decreasing_sequence(X, R)

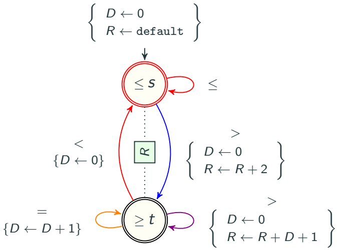


(a) Automaton for the sum_width_decreasing_sequence constraint;

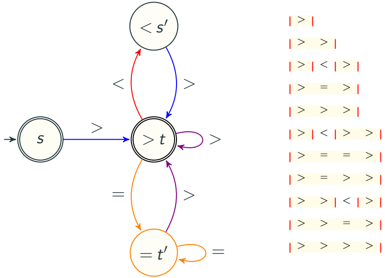
(b) Automaton for the ‘R is *odd*’ rule, constructed from (a)

Example of Automaton for the $R = up(R, n)$ Rule

sum_width_decreasing_sequence(X, R)



(a)



(b)

(a) Automaton for the sum_width_decreasing_sequence constraint;
(b) Automaton for the $R = up(R, n)$ rule, constructed from (a)

Proving Phase: Dependent Groups

- Proof of dependent groups requires case by case consideration.
- The proof consists of verifying a certain property using our cut-generation technique.
- Often, this property is only a sufficient, but not a necessary condition, for proving our hypothesis.

Conclusion

- **Convex Case:** A compositional way of generating cuts from register automata. Already evaluated in [**CP17implied**].
- **Non-Convex Case:** Data Mining + Proof (using automata characterising infeasible combinations of points for conjunction of constraints)
  Currently evaluated from two perspectives:
  - Use small sequences for learning, check on bigger sequences whether uncovered infeasible points appear or not.
  - Check how much it enhances propagation.
- Our method is offline and solver/system independent (build a data base of parameterised invariants)