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

Ekaterina Arafailova, Nicolas Beldiceanu, Helmut Simonis

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

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

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

Abstract. The Holy Grail of programming was described in [7] as the separation of modelling which problem we want to solve, and the application of a tool (a Constraint Solver) to provide a solution from the model. This requires a sophisticated system, which up to now needed to be hand-coded by an expert. We want to go beyond this paradigm, by automatically discovering gaps in existing propagators, deriving a hypothesis about missing propagation, proving or disproving the hypothesis, and finally automatically generating an implementation of the improved constraint propagator. As an example we use the area of time-series constraints.

We describe a method for discovering and proving generic invariants linking together several characteristics of an integer sequence; *generic invariants* are independent of the integer values used in the sequence, but are possibly parameterized by the sequence size. The method consists of three steps, namely (1) a *mining phase* where we systematically generate integer sequences up to a given size, and extract conjectures from the corresponding data sets, (2) a *proof phase* where we try to prove conjectures by checking whether the conjunction of multiple time-series constraints over this sequence is infeasible. We do this by exploiting the descriptions of time-series constraints as automata with accumulators. Finally we use proven conjectures in an (3) *exploitation phase* to improve the propagators for the constraints. Preliminary tests indicate that the discovered generic invariants can significantly 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

In the paper motivating this workshop [7], only the last paragraph ("Synthesis") talks about the automated generation of the Constraint Solver itself. As we

* E. Arafailova is supported by the EU H2020 programme under grant 640954 for the GRACeFUL project. N. Beldiceanu is partially supported by GRACeFUL and by the Gaspard-Monge programme. H. Simonis is supported by Science Foundation Ireland (SFI) under grant numbers SFI/12/RC/2289 and SFI/10/IN.1/I3032.

observed in [3] based on [14], the complexity of developing and maintaining specific propagators for particular constraints is one of the limiting factors in the wider adoption of Constraint Programming. It is perhaps time to go beyond the Holy Grail by automating the writing of (parts of) the Constraint Solver. We will give an example of how we can discover, prove correct and implement improved propagators for a family of time-series constraints by an automated process.

While artificial intelligence has considered from its very beginning the possibility to automate the process of scientific discovery [12], relatively little work has been carried out in this area [15]. 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 analyzed in the context of mathematics [11], most computer science work has focused on the first part, namely generating conjectures both for specific domains like graph theory [9], or for more general domains [6,13]. The main reason for this situation is that the proof part is a key bottleneck, as it is much more challenging to automate as already observed in [4], even if programs that could prove theorems in propositional or first order logic already exist since the fifties [18]. Nowadays there is a renewed interest in proof assistants like Isabelle [19] or Coq [5]; nevertheless such assistants still require describing and formalizing a proof based on human insight, which is typically demanding for proving or invalidating a large set of conjectures about discrete combinatorial objects. More recently, both in the context of circuit design and program verification, mining Boolean expressions, and Boolean combinations of numerical inequalities was respectively done in [8] and [10]. The later uses a theorem prover to verify the proposed invariants.

The main contribution of this paper is to provide for the domain of *time-series constraints on integer sequences* a methodology, which both proposes conjectures, and proves them in an automated way by deriving automata that describe properties of the conjecture and by intersecting them. The technical contributions are as follows:

1. To the best of our knowledge we show for the first time that deterministic automata without accumulators can be used to represent *in constant space all maximal solutions* attached to some bounds.
2. Unlike previous work which considers intersecting automata to get a simplified automata [17,16], we intersect automata to find out in a preprocessing phase conditions characterizing infeasible sets of points that can be used later on to enhance propagation.

We now put this contribution in the context of our recent work on time-series constraints.

In [2] we presented a compositional method for deriving linear invariants for a conjunction of global constraints that are each represented by an automaton, possibly, with accumulators. The experiments revealed that in some cases, despite imposing tight linear invariants linking the result variables of the constraints in the conjunction, the solver could still take a lot of time to find a

feasible solution. After some investigation it turned out that this happens because of some infeasible combinations of values of the result variables that were located within the convex hull of feasible solutions. Consequently, this work focuses on *automatically extracting and proving* invariants that characterize some subsets of infeasible points that are all located inside the convex hull of feasible points. The approach uses three sequential phases:

- Given a conjunction of two time-series constraints on the same sequence of variables, the *mining phase* generates in a systematic way all solutions for small sequence sizes; then it uses a simple bias based on Boolean combinations of arithmetic constraints, possibly parameterized by the sequence size, to derive multiple conjectures, which characterize subsets of infeasible points that are located within the convex hull of all feasible points, and that occur for all generated sequence sizes.
- The *proof phase* synthesizes, from each arithmetic constraint that occurs in a Boolean expression characterizing a subset of infeasible points, a finite automaton without accumulators that has a constant size; having a *constant size* is crucial for generating *sequence size independent invariants*. We sketch for one of the arithmetic constraints how to generate the corresponding constant size automata that do not use any accumulators. If the intersection of these automata is empty, we have proven the conjecture, otherwise we can use the intersection to provide a counter-example.
- For all proven conjectures, we automatically generate propagation rules that are used in the *exploitation phase* that not only prune infeasible assignments, but which also help to find feasible solutions more rapidly.

2 Example

As an illustrative example, we consider the conjunction of two time-series constraints [1] SUM_WIDTH_DECREASING_SEQUENCE($\langle X_1, X_2, \ldots, X_n \rangle, R_1$) and SUM_WIDTH_ZIGZAG($\langle X_1, X_2, \ldots, X_n \rangle, R_2$), as shown in Figure 1. It shows for sequences of length between 9 and 12 (only size 9 shown), in blue all combinations of values for the constraints that are achieved for at least one value assignment. The resulting convex hull is given by red lines, each segment corresponds to a linear inequality that characterises the conjunction as described in [2]. Points in red are infeasible value combinations inside (or on) the convex hull, these are not removed by the linear inequalities. Instead, we now try to learn rules to remove these points for any, arbitrary sequence length. We generate six groups of points that lead to six invariants, one shown in each of the plots. We now have to check each conjecture by deriving automata without accumulators that describe the constraints under the given conditions, and then check their conjunction.

We do this by constructing an automaton for each member of the bias from the original time-series constraint. Figure 2 shows an example for the SUM_WIDTH_DECREASING_SEQUENCE($\langle X_1, X_2, \ldots, X_n \rangle, R_1$) constraint and the bias element *odd*, which is used in Group 6 of Figure 1. The automaton with

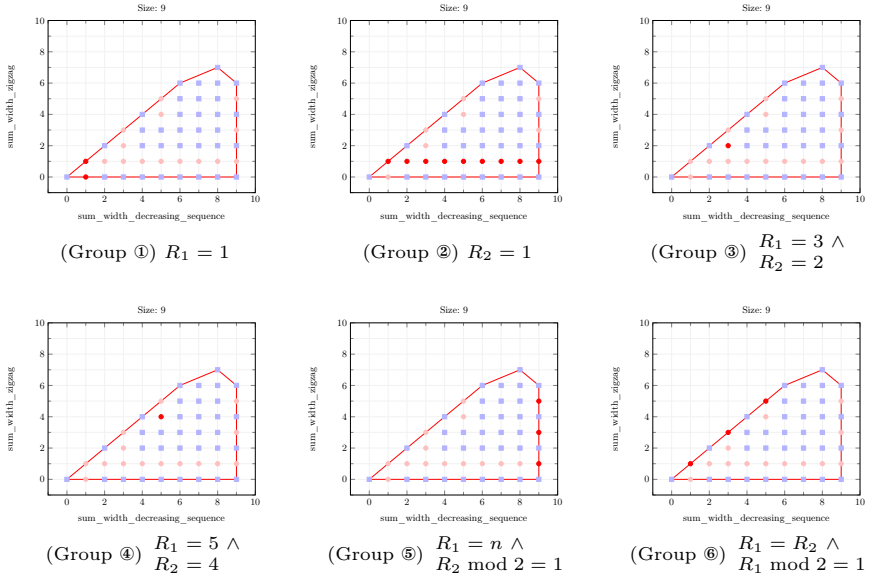

Fig. 1: The first six groups of infeasible points for sequence sizes $n \in \{9, 10, 11, 12\}$ illustrated for $n = 9$; note that within Group ⑤, the term n inside the condition $R_1 = n$ corresponds to the upper bound of R_1 of constraint SUM_WIDTH_DECREASING_SEQUENCE(X, R_1).

two accumulators D and R for the original constraint is on the left, the resulting automaton that checks that the value R of the constraint is odd is on the right. It is obtained by a systematic construction, the states s and t on the left are each replaced by four states $s(y_1, y_2)$ resp. $t(y_1, y_2)$ on the right, which encode whether the two accumulator values are odd or even. The first position represents the accumulator D, the second the result value R. Accepting states are those for which R is odd, i.e. the value of y_2 is 1. Edges in the automaton on the right correspond to edges in the original automaton, keeping track of changes in the parity of the accumulators. Note that the resulting automaton does not use accumulators, but encodes its properties in its states. Similar constructions are used for all other members of the bias.

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., Simonis, H.: Generating linear invariants for a conjunction of automata constraints. In: Beck, C. (ed.) Principles and Practice of Con-

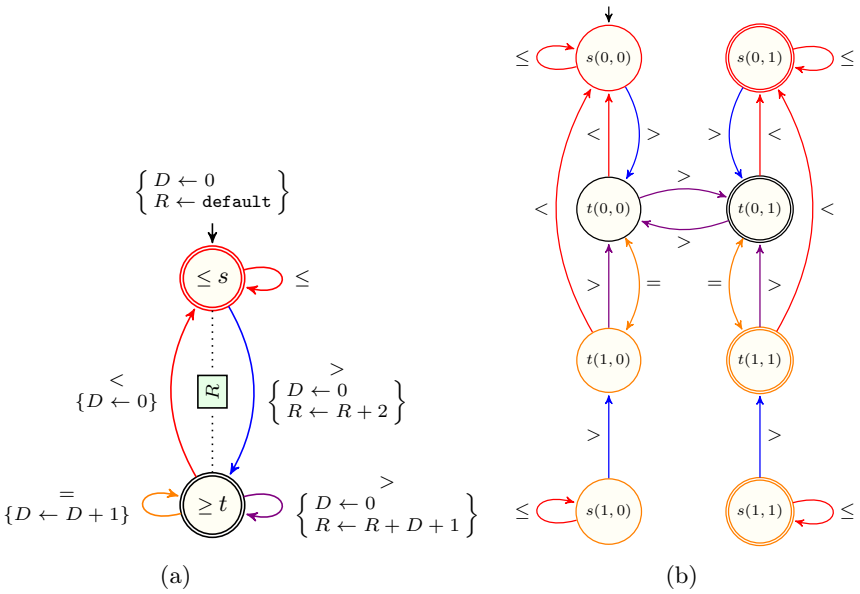

Fig. 2: (a) Automaton for the SUM_WIDTH_DECREASING_SEQUENCE time-series constraint; (b) Automata for property R is *odd*, constructed from (a)

straint Programming - CP 2017, 23rd International Conference, CP 2017. LNCS, Springer (forthcoming)
3. Beldiceanu, N., Flener, P., Monette, J., Pearson, J., Simonis, H.: Toward sustainable development in constraint programming. Constraints 19(2), 139–149 (2014), https://doi.org/10.1007/s10601-013-9152-4
4. 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)
5. 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)
6. Fajtlowicz, S.: On Conjectures of Graffiti. Annals of Discrete Mathematics 38, 113–118 (1988)
7. Freuder, E.C.: In pursuit of the holy grail. Constraints 2(1), 57–61 (1997), https://doi.org/10.1023/A:1009749006768
8. 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)
9. Hansen, P., Caporossi, G.: Autographix: An automated system for finding conjectures in graph theory. Electronic Notes in Discrete Mathematics 5, 158–161 (2000)
10. Krishna, S., Puhrsch, C., Wies, T.: Learning Invariants using Decision Trees. CoRR abs/1501.04725 (2015), http://arxiv.org/abs/1501.04725

11. Lakatos, I.: Proofs and Refutations. Cambridge University Press (1976)
12. Langley, P.W., Simon, H.A., Bradshaw, G., Zytkow, J.M.: Scientific Discovery – Computational Explorations of the Creative Process. MIT Press (1987)
13. Larson, C.E., Cleemput, N.V.: Automated conjecturing I: Fajtlowicz's Dalmatian heuristic revisited. Artif. Intell. 231, 17–38 (2016)
14. Laurière, J.L.: Constraint propagation or automatic programming. Tech. Rep. 19, IBP-Laforia (1996), in French
15. Lenat, D.B.: On automated scientific theory formation: a case study using the AM program. Machine intelligence 9, 251–286 (1979)
16. Menana, J.: Automata and Constraint Programming for Personnel Scheduling Problems. Theses, Université de Nantes (Oct 2011), https://tel.archives-ouvertes.fr/tel-00785838
17. Menana, J., Demassey, S.: Sequencing and counting with the MULTICOST-REGULAR constraint. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings. pp. 178–192 (2009)
18. Newell, A., Simon, H.A.: The logic theory machine – A complex information processing system. IRE Transactions on Information Theory 2(3), 61–79 (1956)
19. Paulson, L.C.: The foundation of a generic theorem prover. J. Autom. Reasoning 5(3), 363–397 (1989)