Functional description of sequence constraints and synthesis of combinatorial objects

Ekaterina Arafailova

▶ **To cite this version:**

Ekaterina Arafailova. Functional description of sequence constraints and synthesis of combinatorial objects. Discrete Mathematics [cs.DM]. Ecole nationale supérieure Mines-Télécom Atlantique, 2018. English. NNT : 2018IMTA0089 . tel-01962957

HAL Id: tel-01962957

https://theses.hal.science/tel-01962957

Submitted on 21 Dec 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THÈSE DE DOCTORAT DE

L'École Nationale Supérieure Mines-Télécom Atlantique
Bretagne Pays de la Loire – IMT Atlantique
COMUE **U**NIVERSITE **B**RETAGNE **L**OIRE

Ecole Doctorale N°601
Mathèmatique et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique et applications

Par

« Ekaterina ARAFAILOVA »

« Functional Description of Sequence Constraints and Synthesis of Combinatorial Objects »

Thèse présentée et soutenue à IMT ATLANTIQUE, NANTES, le 25 septembre 2018
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)
Thèse N° : 2018IMTA0089

Rapporteurs avant soutenance :

Mme Michela Milano, Professeure, Università di Bologna

M. Stanislav Živný, Associate Professor, University of Oxford

Composition du jury :

Président :	M. Claude Jard, Professeur, Université de Nantes
Examinateurs :	M. John Hooker, Professeur, Carnegie Mellon University
	Mme Michela Milano, Professeure, Università di Bologna
	M. Stanislav Živný, Associate Professor, University of Oxford
Dir. de thèse :	M. Nicolas Beldiceanu, Professeur, IMT Atlantique
Co-dir. de thèse :	M. Rémi Douence, Maître assistant, HDR, IMT Atlantique

Abstract

Contrary to the standard approach consisting in introducing ad hoc constraints and designing dedicated algorithms for handling their combinatorial aspect, this thesis takes another point of view. On the one hand, it focusses on describing a family of sequence constraints in a compositional way by multiple layers of functions. On the other hand, it addresses the combinatorial aspect of both a single constraint and a conjunction of such constraints by synthesising compositional combinatorial objects, namely bounds, linear inequalities, non-linear constraints and finite automata. These objects are obtained in a systematic way and are not instance-specific: they are parameterised by one or several constraints, by the number of variables in a considered sequence of variables, and by the initial domains of the variables. When synthesising such objects we draw full benefit both from the declarative view of such constraints, based on regular expressions, and from the operational view, based on finite transducers and register automata. There are many advantages of synthesising combinatorial objects rather than designing dedicated algorithms: 1) parameterised formulae can be applied in the context of several resolution techniques such as constraint programming or linear programming, whereas algorithms are typically tailored to a specific technique; 2) combinatorial objects can be combined together to provide better performance in practice; 3) finally, the quantities computed by some formulae can not just be used in an optimisation setting, but also in the context of data mining.

Key Words: constraint programming, automata, transducers, regular expressions, time series, parameterised combinatorial objects, linear and non-linear invariants

Résumé

À l'opposé de l'approche consistant à concevoir au cas par cas des contraintes et des algorithmes leur étant dédiés, l'objet de cette thèse concerne d'une part la description de familles de contraintes en termes de composition de fonctions, et d'autre part la synthèse d'objets combinatoires pour de telles contraintes. Les objets concernés sont des bornes précises, des coupes linéaires, des invariants non-linéaires et des automates finis; leur but principal est de prendre en compte l'aspect combinatoire d'une seule contrainte ou d'une conjonction de contraintes. Ces objets sont obtenus d'une façon systématique et sont paramétrés par une ou plusieurs contraintes, par le nombre de variables dans une séquence, et par les domaines initiaux de ces variables. Cela nous permet d'obtenir des objets indépendants d'une instance considérée. Afin de synthétiser des objets combinatoires nous tirons partie de la vue declarative de telles contraintes, basée sur les expressions régulières, ansi que la vue opérationnelle, basée sur les automates à registres et les transducteurs finis. Il y a plusieurs avantages à synthétiser des objets combinatoires par rapport à la conception d'algorithmes dédiés: 1) on peut utiliser ces formules paramétrées dans plusieurs contextes, y compris la programmation par contraintes et la programmation linéaire, ce qui est beaucoup plus difficile avec des algorithmes; 2) la synergie entre des objets combinatoires nous donne une meilleure performance en pratique; 3) les quantités calculées par certaines des formules peuvent être utilisées non seulement dans le contexte de l'optimisation mais aussi pour la fouille de données.

Mots clés : programmation par contraintes, automates, transducteurs, expressions régulières, séries temporelles, objets combinatoires paramétrés, invariants linéaires et non-linéaires

Acknowledgements

First of all, I would like to thank my thesis director Nicolas Beldiceanu. Thank you for your support during these three years, the knowledge and the experience that you have handed over to me, for your precious advice about many things. It was a great honour and a pleasure to work with you for three years.

I would like to thank my co-director Rémi Douence. Thank you for your feedback on my writing and presentations, for sharing your opinion about our work, and for bringing a different point of view in what we were doing.

I would like to thank my jury members: Michela Milano, Stanislav Živný, John Hooker and Claude Jard. Thank you very much for reviewing and examining my work and for your valuable feedback.

I would like to thank the EU H2020 programme under grant 640954 for the GRACeFUL project for funding my thesis and everyone with whom I had an opportunity to work with during the project.

I would like to thank Helmut Simonis with whom I had an opportunity to co-author all my papers. Thank you for sharing your knowledge about the practical side of our work, for showing Cork, for interesting discussions on various topics, and for the delicious biscuits.

I would like to thank my other co-authors, Mats Carlsson, Pierre Flener, María Andreína Francisco Rodríguez, and Justin Pearson. Thank you for all pleasure and fun I had when working with you, for the picnics and the barbecues, and for your warm welcome in Uppsala.

I would like to thank the members of the TASC research group: Amine Balafrej, Anicet Bart, Philippe David, Arthur Godet, Giovanni Lo Bianco, Xavier Lorca, Gilles Madi Wamba, Thierry Petit, Charles Prud'homme, Gilles Simonin, and Charlotte Truchet. Thank you for all the laughter and interesting discussions that we had during coffee breaks.

I would like to thank Catherine Fourny, Florence Rogues and Anne-Claire Binetruy. Thank you for solving administrative questions in a fast and efficient way, and for organising numerous travels that I had during my thesis.

I would like to thank Evgeny Gurevsky and Pavel Borisovsky for giving me the information about the ORO Master program, and for helping me to move to France.

I would like to thank my dear friends Abood Mourad, Viktoriia Ihnatova, Polina Kalchevskaya, and Natalia and Alexey Tichshenko. That would have been extremely complicated to get through the PhD story without you!

I would like to thank my mother Natalia and our cat Barsenka. Without your support and your trust in me I would not be where I am now doing what I am doing.

Last but not the last, I would like to thank my dear Matthieu. Your support and your love made me carry on and struggle at the moment of desperation. Without you I would not be able to finish my thesis. I cannot express how much I am grateful to you for everything.

Contents

1 Introduction **13**
- 1.1 Tradeoff Between the Expressiveness of a Modelling Language and the Efficiency of Solving for Combinatorial Problems . . . 13
- 1.2 Mathematical Programming and Constraint Programming for Modelling and Solving Combinatorial Problems . . . 13
- 1.3 Context of Our Work: Time-Series Constraints . . . 14
- 1.4 The Two Topics of this Thesis . . . 15
- 1.5 Differences with Existing Approaches . . . 17
- 1.6 A Guided Tour Through the Main Contributions of this Thesis . . . 17
- 1.7 The Reading Grid of this Thesis . . . 22

I Background **25**

2 Background on Regular Expressions **29**

3 Background on Automata, Register Automata and Transducers **31**
- 3.1 Defining Automata, Register Automata and Transducers . . . 31
- 3.2 Operations on Automata and Register Automata . . . 34
  - 3.2.1 Intersection . . . 34
  - 3.2.2 Union . . . 35
  - 3.2.3 Complement . . . 35

4 Background on Constraint Programming **37**
- 4.1 Constraints and Constraint Satisfaction Problems . . . 37
- 4.2 Solving a Constraint Satisfaction Problem . . . 38
- 4.3 Representation of a Constraint Satisfaction Problem . . . 39
- 4.4 Automata and Register Automata in Constraint Programming . . . 39
  - 4.4.1 REGULAR Global Constraint . . . 40
  - 4.4.2 COST-REGULAR and MULTI-COST-REGULAR Global Constraints . . . 40
  - 4.4.3 AUTOMATON Global Constraint . . . 41

5 Background on Time-Series Constraints **43**
- 5.1 Defining Time-Series Constraints . . . 43
- 5.2 Operational View of Time-Series Constraints . . . 46
  - 5.2.1 Seed Transducer for a Regular Expression . . . 47
  - 5.2.2 Synthesising and Simplifying Register Automata . . . 49
  - 5.2.3 Glue Constraints . . . 50
- 5.3 Related Approach: Quantitative Regular Expressions . . . 52

II Theoretical Contributions **55**

6 Overview of our Theoretical Contributions **59**
6.1 Contributions for Time-Series Constraints in Isolation . . . 59
6.1.1 First Key Idea: Regular-Expression Characteristics . . . 60
6.2 Contributions for a Conjunction of Time-Series Constraints . . . 61
6.2.1 Second Key Idea: Operational View of Time-Series Constraints . . . 62
6.3 Integrating Combinatorial Objects into the Global Constraint Catalogue . . . 63
6.4 Overview of the Extended Transducer-Based Model . . . 63

7 Synthesising Parameterised Bounds **65**
7.1 Regular-Expression Characteristics . . . 66
7.1.1 A Notation System for Regular-Expression Characteristics . . . 66
7.1.2 Size . . . 67
7.1.3 Height . . . 67
7.1.4 Range . . . 68
7.1.5 Set of Inducing Words . . . 68
7.1.6 Overlap . . . 69
7.1.7 Smallest Variation of Maxima . . . 71
7.1.8 Summary Example Illustrating All Regular-Expression Characteristics . . . 73
7.1.9 Necessary and Sufficient Condition for the Existence of an Occurrence of a Regular Expression . . . 74
7.2 Time-Series Constraints with Feature ONE . . . 76
7.2.1 A Sharp Lower Bound on the Number of Pattern Occurrences . . . 76
7.2.2 A First Not Necessarily Sharp Upper Bound . . . 77
7.2.3 Extending the Upper Bound to Get a Sharp Bound Under Some Hypothesis . . . 79
7.3 Time-Series Constraints with Feature WIDTH . . . 86
7.3.1 Properties of Regular Expressions . . . 86
7.3.2 Upper Bound for MAX_WIDTH_σ . . . 87
7.3.3 Upper Bound for SUM_WIDTH_σ . . . 88
7.3.4 Lower Bound for MIN_WIDTH_σ . . . 91
7.4 Synthesis . . . 92
7.5 Conclusion . . . 95

8 Synthesising Parameterised AMONG Implied Constraints **97**
8.1 Complexity of the SUM_SURF_PEAK Time-Series Constraint . . . 98
8.2 Deriving an AMONG Implied Constraint . . . 98
8.2.1 Regular-Expression Characteristics . . . 99
8.2.2 Deriving an AMONG Implied Constraint for the MAX_SURF_σ, MIN_SURF_σ and the SUM_SURF_σ Families . . . 101
8.3 Conclusion . . . 107

9 Synthesising Parameterised Linear Invariants **109**
9.1 Generating Linear Invariants . . . 109
9.1.1 Constructing the Invariant Digraph for a Conjunction of AUTOMATON Constraints wrt a Linear Function . . . 111
9.1.2 Finding the Relative Coefficients of the Linear Invariant . . . 114
9.1.3 Finding the Constant Term of the Linear Invariant . . . 115
9.2 Improving the Generated Linear Invariants . . . 116
9.2.1 Preprocessing Technique of the Intersection of Register Automata . . . 116
9.3 Generating Additional Invariants . . . 120

9.3.1 Generating Conditional Linear Invariants with the Non-Default Value Condition . 121
9.3.2 Generating Linear Guard Invariants 121
9.4 Infeasible Combinations of the Result Values not Eliminated by the Generated Linear Invariants 122
9.5 Conclusion 123

10 Synthesising Parameterised Non-Linear Invariants 125
10.1 Motivation and Running Example 126
10.2 Discovering and Proving Invariants 126
10.2.1 Mining Phase 128
10.2.2 Proof Phase 129
10.3 Infeasible Combinations not Eliminated by our Non-Linear Invariants 132
10.4 Conclusion 134

11 Synthesising Constant-Size Conditional Automata 135
11.1 Generation of Constant-Size Automata for Constant Atomic Relations 135
11.2 Generation of Constant-Size Automata for Modulo Atomic Relations 138
11.3 Generation of Constant-Size Automata for Gap Atomic Relations 139
11.3.1 Deriving a δ-gap Automaton for a Time-Series Constraint 140
11.3.2 Deriving the δ-gap Automaton for the NB_σ Family 145
11.3.3 Deriving the δ-gap Automaton for the SUM_WIDTH_σ Family 150
11.3.4 Conclusion 153
11.4 Generation of Constant-Size Automata for $\geq$ and $\leq$ Atomic Relations 153
11.5 Conclusion 154

12 Extended Transducer-Based Model 155
12.1 Defining Functions over Integer Sequences 155
12.2 Operational View of Functions Over Integer Sequences 159
12.2.1 Handling the Recognition Aspect: Seed Transducer 160
12.2.2 Handling the Computational Aspect: Reduced Instruction Set 162
12.3 Conclusion, Related Work, and Future Work 166

III Practical Evaluation of our Contributions 167

13 Evaluation of the Impact of Bounds 171

14 Evaluation of the Impact of AMONG Implied Constraints 175

15 Evaluation of the Impact of Linear Invariants 177

16 Evaluation of the Impact of Non-Linear Invariants 181

Conclusion 183
17.1 Summary of this Thesis 183
17.2 Future Work 184
17.2.1 Improving the Solving Aspect 184
17.2.2 Complexity Analysis 185
17.2.3 Formalisation and Generalisation Issues 185
17.2.4 Applications 185

French Summary **187**

Appendices **193**

A An Entry of the Global Constraint Catalogue **195**
A.1 Metadata 195
A.2 PDF Pages 203

B An Entry of the Database of Invariants of the Global Constraint Catalogue **209**
B.1 Metadata 209
B.2 PDF Pages 209

C Tables with Regular-Expression Characteristics **211**

Notation for Regular-Expression Characteristics **223**

Index **225**

Bibliography **229**

List of Figures

1.1 Synthesised combinatorial objects and the facets from which they were synthesised 16
1.2 The time series $\langle 3, 2, 4, 2, 4, 1, 3, 2, 3, 0\rangle$ with the maximum (five) number of decreasing sequences among all time series of length 10. 18
1.3 Feasible and infeasible combinations of the results values of two constraints imposed on the same sequence whose length is in $\{9, 10, 11, 12\}$. 20
1.4 (A) Automaton accepting the signatures of all, and only all, integer sequences with the maximum number of decreasing sequences. (B) All signatures of lengths 3 and 4 accepted by the automaton in (A). 21

3.1 (A) Automaton recognising the signatures of integer sequences whose elements are all in $\{-1, 0, 5\}$. (B) Register automaton recognising any signature and returning the number of elements in $\{-1, 0, 5\}$ in an integer sequence. (C) Register automaton recognising any signature and returning the sum of elements in $\{-1, 0, 5\}$ in an integer sequence. (D) Transducer with the input alphabet $\{\in, \notin\}$ and the output alphabet $\{\texttt{found}, \texttt{out}, \texttt{in}, \texttt{out}_\texttt{a}\}$ . . . 33
3.2 (A) Automaton recognising sequences of 0 and 1, where 0 are only located at odd positions; (B) automaton recognising sequences of 0 and 1, where 1 are only located at even positions; (C) intersection of (A) and (B); (D) complement of (A). 34
3.3 (A) and (B) register automata over alphabet $\{<, =, >\}$; (C) intersection of (A) and (B) . . 35

5.1 Time series $\langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$ with its two peaks of respective widths 3 and 1 46
5.2 Seed transducer for the PEAK regular expression. This figure is adapted from [10]. 48
5.3 Register automata for NB_PEAK and SUM_WIDTH_PEAK. These figures are adapted from [10]. 49
5.4 Simplified register automata for NB_PEAK (left) and SUM_WIDTH_PEAK (right). These figures are adapted from [10]. 50
5.5 (A) Features: the function used for computing the value of a feature. (B) Decoration table used for synthesising the register automaton for a time-series constraint 51
5.6 (A) The prefix $\langle 0, 1, 2\rangle$ of the time series $\langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$ without any peaks. (B) The suffix $\langle 2, 2, 0, 0, 4, 1\rangle$ of the time series $\langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$ with one peak 52

6.1 Regular-expression characteristics introduced in Chapters 7 and 8. 61
6.2 Seed transducer for $\sigma =$ DECREASING_TERRACE when b_σ is 1 (A) and 2 (B) 64

7.1 Illustration of the introduced regular-expression characteristics 74
7.2 Lemma 7.2.2 Case (1.1): Illustration of the word $z_1w_1w_2w_1w_2$ belonging to the language of '$v \mid z_1(w_1w_2)^*(w_1 \mid \varepsilon)$' 82

8.1 Time series illustrating the introduced regular-expression characteristics 99
8.2 Time series illustrating the intuition of the AMONG implied constraints 103

9.1 (A) Register automaton for NB_PEAK. (B) Register automaton for NB_VALLEY. (C) Intersection of (A) and (B). 111

9.2 Invariant digraph for NB_PEAK and NB_VALLEY wrt $e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V$ 112
9.3 (A) The invariant digraph of the register automata for two time-series constraints. (B) The set of feasible values of the result variables of two constraints 115
9.4 Intersection of the register automata for two time-series constraints, for which our method does not generate sharp linear invariants 117
9.5 Delayed intersection obtained from the intersection in Figure 9.4 118
9.6 Invariant digraph obtained from the delayed intersection in Figure 9.5 120
9.7 (A) General linear invariants, (B) linear invariants with non-default value conditions for a pair of time-series constraints. (C) Automaton with guard invariants 121
9.8 Illustration of infeasible combinations of the result values of time-series constraints not eliminated by the generated linear invariants 123

10.1 Feasible and infeasible combinations of the result values of two time-series constraints imposed on the same sequence whose length is in $\{9, 10, 11, 12\}$ 127
10.2 Seven groups of infeasible combinations of the result values of two time-series constraints 131
10.3 (A) Automaton for a gap atomic relation. (B) Automaton for a modulo atomic relation . . 133
10.4 Illustration of infeasible combinations of the result values not eliminated byt the generated non-linear invariants 133

11.1 (A) Register automaton for SUM_WIDTH_DECREASING_SEQUENCE(X, R). (B) Automaton for the $R = 3$ constant atomic relation. (C) All signatures of length 3 accepted by the automaton in Part (B) 136
11.2 (A) Register automaton for NB_DECREASING_SEQUENCE(X, R). (B) Automaton for the $R \bmod 2 = 1$ atomic relation. (C) All signatures of length 2 accepted by the automaton in Part (B) 138
11.3 (A) Automaton achieving the maximum number of peaks in a time series of length n. (B) All corresponding accepted words for $n - 1 \in \{4, 5\}$. (C) The signatures of time series with gap 1 and 2, respectively, and with loss 3 and 5, respectively 140
11.4 Illustration of gap and loss for six time series 143
11.5 Seed transducer (A) and separated seed transducer (B) for the PEAK regular expression. . . 149
11.6 Loss automaton for NB_PEAK. The initial value of the registers C, D, and R is zero. . . . 151

12.1 Well-formed output language 161
12.2 New seed transducers for 5 regular expressions 163
12.3 Trace for the MAX_WIDTH_GROUP constraint 165

13.1 Comparing backtrack count and runtime for *Automaton* and its variants 172
13.2 Scalability results comparing time for *Automaton* and *Combined* on problems of increasing length. 173
13.3 Comparing parts of the search tree for MAX_SURF_INCREASING_TERRACE, finding the first solution or proving infeasibility 174

14.1 Comparing backtrack count and runtime of the g_f_σ time-series constraint for previous best results and new method for finding the first solution or proving infeasibility 176

15.1 Comparing constraint variants, undecided instances percentage for size 18 as a function of time 178
15.2 Percentage of problems solved for 3 overlapping segments of lengths 22, 24, and 25. . . . 179

17.1 Synthesised combinatorial objects, grouped by the case they are synthesised for, i.e. characterising a single constraint or a conjunction of constraints 184
17.2 Les objets combinatoires synthétisés et les facettes à partir desquelles ils étaient synthétisés 189

List of Tables

5.1 Features and aggregators . . . 44
5.2 Regular-expression names σ, corresponding regular expressions, and values of the parameters a_σ and b_σ. This table is adapted from [14]. . . . 45
5.3 Glue matrix for the NB_PEAK constraint . . . 52
5.4 Comparison of time-series constraints and QREs . . . 53

7.1 Regular-expression names and corresponding size, height, range, set of inducing words, overlap and smallest variation of maxima . . . 75
7.2 A synthesis of all the bounds presented in Sections 7.2, 7.3, and in [8]. . . . 93
7.3 Classification of regular expressions: regular expression names σ, their properties and conditions on domain $[\ell, u]$ when they hold. . . . 94

8.1 Regular expressions and corresponding maximum value occurrence number and big width. 102
8.2 Regular expressions and corresponding interval of interest and the lower bound on the parameter of the AMONG implied constraints for the MAX_SURF_σ family . . . 105
8.3 Regular expressions and corresponding interval of interest and the lower bound on the parameter of the AMONG implied constraints for the SUM_SURF_σ family . . . 106

11.1 Decoration table for the loss automaton for NB_σ time-series constraints . . . 150

12.1 Features and aggregators of the extended transducer-based model . . . 157
12.2 Illustration of s-occurrences of a concrete pattern in a signature, and of their corresponding found indices, e-occurrences and i-occurrences . . . 158
12.3 Operational views of features and aggregators in the extended transducer-based model . . 159
12.4 Examples of functions over integer sequences . . . 159

16.1 Comparing the state-of-the-art baseline and the baseline with the generated invariants . . . 181

C.1 Table for the size regular-expression characteristic . . . 211
C.2 Table for the height regular-expression characteristic . . . 212
C.3 Table for the range regular-expression characteristic . . . 213
C.4 Table for the set of inducing words regular-expression characteristic . . . 214
C.5 Table for the overlap regular-expression characteristic . . . 215
C.6 Table for the smallest variation of maxima regular-expression characteristic . . . 216
C.7 Table for the interval of interest regular-expression characteristic . . . 218
C.8 Table for the maximum value occurrence number regular-expression characteristic . . . 220
C.9 Table for the big width regular-expression characteristic . . . 222

Chapter 1

Introduction

1.1 Tradeoff Between the Expressiveness of a Modelling Language and the Efficiency of Solving for Combinatorial Problems

Many real-life problems, e.g. staff scheduling in a call centre or production planning of a power plant, can be described as mathematical models. In such a context, we have two main aspects: 1) the modelling language aspect, i.e. our language should be rich enough to concisely express a large variety of problems; 2) the solving aspect, i.e. we should be able to find a solution to our model efficiently. Currently, we face one of the two following situations:

1. We have a powerful language allowing us to model easily and that can be further extended. However, the solving aspect is highly inefficient.
2. Our language is restricted and extending the language may require adding ad hoc elements specific to a considered problem, but not useful for any other problem.

Within the context of problems using integer sequences, the goal of this thesis is to obtain a tradeoff between the expressiveness of the modelling language and the efficiency of the solving aspect for combinatorial problems. We work towards a language that is powerful enough to describe a large variety of problems, and efficient enough from the solving point of view. Our approach is based on the following observation: any model for a combinatorial problem has two main components, namely 1) *variables* that represent quantities, e.g. produced amount of electricity for a given power plant, and take their values in given sets, called *domains*, and 2) *constraints*, which impose relations between these variables and represents business processes, technical restrictions, etc. Such models often use *discrete objects* such as

— permutations [117];
— trees [42], i.e. acyclic connected graphs;
— time series [47], i.e. integer sequences representing measurements taken over time.

Such discrete objects can be described by their characteristics, e.g. the number of cycles in a permutation [82], the diameter of a tree [104], and the number of peaks in a time series [22]. Characteristics are often used to represent the constraints of the problem. In the solving context, we typically need to find a discrete object simultaneously satisfying restrictions on its several characteristics, e.g. a time series with 3 peaks and 2 valleys. Restricting several characteristics may be more challenging than restricting a single characteristic since during the solving phase constraints have to communicate efficiently, which is not always the case.

1.2 Mathematical Programming and Constraint Programming for Modelling and Solving Combinatorial Problems

Mathematical Programming (MP) [126] and *Constraint Programming* (CP) [118] are two complementary approaches for modelling and solving combinatorial problems using discrete objects with a number of

successful applications in the domains of scheduling, packing, and routing [134, 135, 48, 51, 110, 95, 65].

The main difference between CP and MP are the types of constraints used for modelling. In the context of MP, constraints are usually linear or convex [16, 33, 115], and, for example, for problems with only linear constraints, solvers typically use the *simplex method* [58]. CP models use *global constraints*. The Global Constraint Catalogue [21] defines a global constraint as "an expressive and concise condition involving a non-fixed number of variables". For example, the ALLDIFFERENT($\langle X_1, X_2, \ldots, X_n \rangle$) [130] global constraint restricts a sequence of integer variables $\langle X_1, X_2, \ldots, X_n \rangle$ to take distinct values. Therefore, the sequence $\langle 1, 8, 7, -1, 3 \rangle$ satisfies an ALLDIFFERENT constraint, but $\langle 1, 8, 1, -1, 3 \rangle$ does not since X_1 is the same as X_3. In CP, a global constraint usually comes with a *filtering technique*, which is an algorithm or any kind of inference that allows one to reduce the domains of the variables by removing values that cannot be part of any solution to this constraint.

Despite different constraint types, and thus different solving techniques, CP and MP have some common drawbacks that motivate the work of this thesis:

- In both MP and CP, modelling can be challenging both from the point of view of problem description and from an inference point of view. In MP, this is due to the fact that constraints must be linear or convex. In CP, this is due to the fact that a required global constraint may not exist and needs to be introduced. Hence there is a common need to *define constraints in a compositional way* that can be then systematically reformulated as linear programs or for which one can obtain a filtering technique in a systematic way.

- When domains of variables are discrete, both MP and CP models may become hard to solve [106, 131]. Hence in order to solve a problem efficiently one tries to draw full benefit from the structure of the considered problem. In MP, this is done in the preprocessing step, where a solver verifies whether a considered problem has a well-known structure, e.g. network flow [63], and then applies a specific preprocessing technique for this subproblem and/or generates cuts [75, 96]. In CP, this is done by designing dedicated filtering techniques for global constraints of the problem. Hence there is a need to synthesise *combinatorial objects* characterising the structure of a considered combinatorial problem, e.g. bounds, linear cuts, implied constraints, which are redundant constraints that do not change the set of solutions of the problem, but their purpose is to remove infeasible values from the domains of the variables.

- The need to exploit the problem structure leads to a large number of *ad hoc* methods, e.g. specific bounds, algorithms, decompositions, filtering techniques, heuristics. These are methods that are efficient for solving the problem they were designed for, but either cannot be reused at all for any other problem or require a significant effort for adjusting them. Hence there is a need to develop *systematic methods* for synthesising combinatorial objects for constraints occurring in a considered problem.

1.3 Context of Our Work: Time-Series Constraints

This thesis studies a family of global constraints, called *time-series constraints*, defined in a compositional way by means of functions [22, 10]. A time-series constraint $\gamma(X, R)$ restricts R, called *the result value of* γ, to be the result of some computations over the sequence of integer variables $X = \langle X_1, X_2, \ldots, X_n \rangle$, called a *time series*, which represents measurements taken over time [22]. For example, R could be the number of consecutive pairs of variables $\langle X_i, X_{i+1} \rangle$ of X such that $X_i < X_{i+1}$ with i in $[1, n-1]$. The three main ingredients describing a time-series constraint are a *pattern*, a *feature*, and an *aggregator*. A pattern is some regular form of subsequences, which is from a formal point of view characterised by a regular expression over the alphabet of three letters $\{\text{'}<\text{'}, \text{'}=\text{'}, \text{'}>\text{'}\}$. For example, the DECREASING_SEQUENCE pattern, which corresponds to any maximal monotonously decreasing subsequence $\langle X_i, X_{i+1}, \ldots, X_j \rangle$ of a sequence of integers $\langle X_1, X_2, \ldots, X_n \rangle$ is characterised by the

'$(> (> \mid =)^*)^* >$' regular expression, which relates the variables of the subsequence $\langle X_i, X_{i+1}, \ldots, X_j \rangle$ as follows:

- $X_i > X_{i+1}$, i.e. this subsequence starts with a strict decrease;
- $X_{j-1} > X_j$, i.e. this subsequence also ends with a strict decrease;
- for any k in $[i+1, j-2]$, we have that $X_k \geq X_{k+1}$, i.e. in the middle this subsequence can either decrease or stay at the same level.

For example, in the $\langle 1, 2, 0, 0, -1, 3, 4, 2, 2 \rangle$ time series there are two decreasing sequences, namely $\langle 2, 0, 0, -1 \rangle$ and $\langle 4, 2 \rangle$. Note that although $\langle 2, 0 \rangle$ satisfies the conditions on the relations between its values, it is included in $\langle 2, 0, 0, -1 \rangle$, and thus is not maximal.

A feature and an aggregator are functions over integer sequences, e.g. the maximum of a sequence of integers, or the sum of elements in an integer sequence.

Time series are very common in many real-life applications. We now give a few examples of possible usage of time-series constraints:

- Analysis of the output of electric power stations over multiple days in the context of solving the unit commitment problem [28]. From known production curves of power plants one can extract a model using time-series constraints, and then generate similar production curves satisfying additional restrictions for a considered power plant.
- Modelling a problem of staff scheduling in a call centre [11]. The overall problem is to cover the given manpower demand over time, while minimising overall resource cost, and at the same time satisfying restrictions related to business processes, employment rules, and union contracts, which can be expressed as time-series constraints.
- Data mining in the context of power management for large-scale distributed systems [26].
- Trace analysis for Internet Service Provider to test the bandwidth of the user's Internet connexion [66].
- Anomaly detection and error correction in the temperature in a building [113].
- Real-time decision-making, for example, where one needs to analyse data streams in order to adjust the toll rate depending on the traffic [5].

1.4 The Two Topics of this Thesis

The **first topic of this thesis** is developing systematic methods for *synthesising compositional combinatorial objects* such as bounds, linear invariants, automata for time-series constraints. The main idea is to exploit the compositional nature of time-series constraints at the combinatorial level, i.e. the level related to the solution space associated with a constraint or a conjunction of constraints. Compositionality here means that we can combine such objects during the solving phase and also we can use them with different technologies and/or in different contexts, e.g. CP, MP, data mining.

A formula typically captures some combinatorial relation between different quantities. The idea put forward in this thesis is based on the bet that, provided that it is possible to synthesise them, the set of formulae and redundant constraints potentially has more impact than a set of dedicated algorithms. Indeed, from a compositional point of view, formulae can be used conjointly and applied in the context of several resolution techniques such as CP or MP, which is much more difficult in the context of algorithms. As we will see in the benchmarks of Part III, yet another advantage of combinatorial objects is *synergy* between them, i.e. we can compose them. Different combinatorial objects combined together provide us with better performance than when used separately. A vibrant example of such synergy is the interaction of bounds on the result value of a time-series constraint γ and *glue constraints* [8, 23]. For a sequence of variables $X = \langle X_1, X_2, \ldots, X_n \rangle$, a prefix $P = \langle X_1, X_2, \ldots, X_i \rangle$ and a reversed suffix $S = \langle X_n, X_{n-1}, \ldots, X_i \rangle$ of X, a glue constraint links the result values of three time-series constraints γ imposed on X, on P, and on S.

Synthesised combinatorial objects can be used for different purposes including, but not limited to:

- When solving a problem in the context of CP, the goal is, usually, to prune as many infeasible

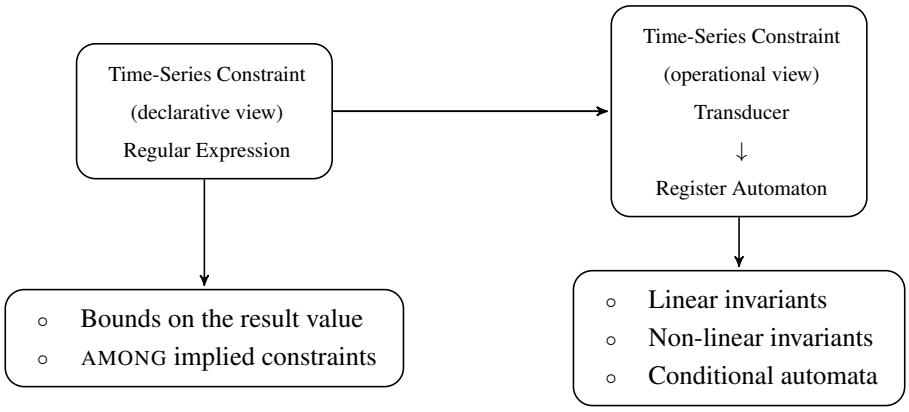


Figure 1.1 – Synthesised combinatorial objects and the facets from which they were synthesised, i.e. declarative with regular expressions or operational with transducers and/or register automata. An arrow from source to destination indicates that destination can be synthesised from source.

values of variables as possible since the smaller are the domains, the easier it is to find a solution. Synthesised combinatorial objects can be used for making the pruning of time-series constraints stronger.

- While time-series constraints can be reformulated as linear models [11] and integrated into existing linear models, the obtained linear reformulation is not *tight*, i.e. a linear programming solver such as CPLEX or Gurobi typically spends a lot of time to solve it. Our combinatorial objects can be used to fasten the solving aspect in the context of linear programming.
- Time-series constraints can be used in the context of *data mining*. For example, bounds on the result value of a time-series constraints are used for clustering time series representing the workload of a data centre [94]; bounds allow us to compare the maximum ranges of variation of the result values of different time-series constraints.

From the operational point of view, every time-series constraint γ has a representation by a *register automaton*, which is synthesised from the *seed transducer* for a regular expression associated with γ [22]. It was shown in [68] how to automatically generate a seed transducer from a regular expression. All combinatorial objects we obtain in this thesis will be either synthesised from the declarative view of time-series constraints, i.e. using regular expressions, or from their operational representation, i.e. using register automata and seed transducers. Figure 1.1 gives the classification of the combinatorial objects depending on the representation of time-series constraints, from which they were synthesised, i.e. declarative or operational. The combinatorial objects presented in Figure 1.1 will be further detailed in Section 1.6.

While using transducers and automata has a long-standing tradition in the context of synthesising reliable software components [133, 128], it is rarely used to synthesise combinatorial objects such as bounds, cuts or glue constraints. However one can point out the following correspondence between computer-aided verification [55] and constraint programming: first, both use sometimes high-level declarative specifications from which transducers and or automata are synthesised. Second, there is a correspondence between invariants that are typically extracted from these transducers and automata for proving some property of a program or a system, and the necessary conditions one would like to synthesise in the context of CP or MP to get stronger inferences: both are formulae that must always be true.

The **second topic of this thesis** is the *extension of the approach used for describing time-series constraints* to capture a larger number of sequence constraints such as [25, 105, 108]. The initial work [22] uses finite transducers to synthesise filtering techniques for time-series constraints. However, the same

transducer-based model can be extended for synthesising filtering techniques for other global constraints such as AMONG [25], SIMILARITY [105], and STRETCH [108].

1.5 Differences with Existing Approaches

Before giving an overview of our contributions, we state four reasons that distinguish our work from other approaches:

- First, in the literature there are approaches that either focus on the combinatorial aspect of specific constraints such as ALLDIFFERENT, REGULAR, NVALUE [116, 19, 39, 41] or propose generic approaches for describing constraints and synthesising filtering techniques [129, 100, 73]. Some of the approaches do not automatically handle the combinatorial aspect of a constraint: they rely on the user to describe a filtering technique by a set of formulae [129, 100]. In the others, the set of solutions to the constraint is represented by a multi-valued decision diagram (MDD) [35, 107] that can be exponential in size. Some works are devoted to synthesis of an approximation of MDDs of a smaller size [76]. However, MDDs do not focus on the relations between different characteristics of discrete objects. In our work, we go a step further and explore the topic of *automatically synthesising* propagators in the form of combinatorial objects for the large class of time-series constraints [22] involving more than 200 constraints.
- Second, the obtained combinatorial objects can be used, not only as propagators in the context of constraint programming, but also in the context of linear programming, data mining, local search. This implies that such objects represent essential information about the combinatorial aspect of a time-series constraint, and thus are independent of the context in which time-series constraints are used.
- Third, the obtained objects are *parameterised* by the description of a considered time-series constraint, the length of a time series, and the domains of the time-series variables, and are synthesised *once and for all*. This allows us to create a *database* of combinatorial objects for time-series constraints [10] and consult it in completely different contexts every time when required. There is no need to rerun our methods for synthesising these combinatorial objects for each problem instance. Note that, in order to obtain such combinatorial objects, we have to *automatically prove* that they are valid for any sequence length.
- Fourth, working towards uniform ways of representing families of global constraints and of handling their combinatorial aspect is not common within the CP community, but is still important since otherwise we would end up with a set of dedicated constraints for each problem that do not communicate.

1.6 A Guided Tour Through the Main Contributions of this Thesis

The main contributions presented in this thesis are the following:

- [**Parameterised upper and lower bounds** on the result value of every time-series constraint]
  A bound formula for a considered time-series constraint is parameterised by the time-series length n, and the domains of the time-series variables. Each bound formula is obtained from some generic formula, which is parameterised by a considered time-series constraint. Hence we only need to prove very few generic formulae, i.e. less than 10, rather than one formula per time-series constraint, i.e. more than 200. While the bound is always valid, its sharpness is only guaranteed when the domains of all time-series variables correspond to the same integer interval. For almost all time-series constraints, both upper and lower bounds are evaluated in constant time, except 12 time-series constraints, for which it takes $O(n)$ to evaluate the bound [8].

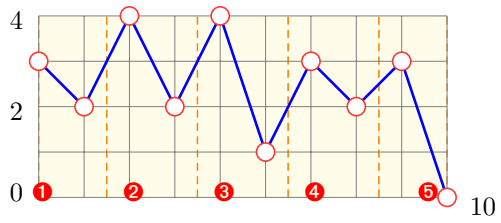


Figure 1.2 – The time series $\langle 3, 2, 4, 2, 4, 1, 3, 2, 3, 0\rangle$ with the maximum (five) number of decreasing sequences among any time series of length 10. The horizontal axis is for time-series elements, and the vertical axis is for the values. The dashed lines separate different decreasing sequences.

This work was published in the *Constraints* journal [14] and in the proceedings of the *CP'16* conference [8], and the bounds for all time-series constraints were integrated into the Volume II of the Global Constraint Catalogue [10].

Example 1.6.1 (sharp bounds)**.** Consider a sequence of integers $X = \langle X_1, X_2, \ldots, X_n\rangle$. A *decreasing sequence* in X is a maximal inclusion-wise monotonously decreasing subsequence of X. For example, the sequence $\langle 1, 2, 1, 0, 0, -1, -2, 2, 4, 2, 2\rangle$ has two decreasing sequences, namely $\langle 2, 1, 0, 0, -1, -2\rangle$ and $\langle 4, 2\rangle$. Since each decreasing sequence contains at least two elements and any two decreasing sequences never overlap, the maximum number of decreasing sequences in X is $\lfloor \frac{n}{2} \rfloor$. Hence for the NB_DECREASING_SEQUENCE$(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint, where R is constrained to be the number of decreasing sequences in $\langle X_1, X_2, \ldots, X_n\rangle$, a sharp upper bound on R is $\lfloor \frac{n}{2} \rfloor$. For example, Figure 1.2 gives a time series of length $n = 10$ with 5 decreasing sequences, which is the maximum possible number of decreasing sequences in any time series of length 10. The formula $\lfloor \frac{n}{2} \rfloor$ is a special case of a generic formula of Theorem 7.2.2, on 84, that gives the number of maximal inclusion-wise occurrences of a pattern in a sequence of integer numbers. In this example, the pattern is DECREASING_SEQUENCE. △

- [**Parameterised AMONG implied constraints** for three families of time-series constraints]
  An AMONG global constraint [25] restricts the number of variables of a sequence of variables to take their values in a particular finite set of integer values. Here, the word *implied* means that these constraints are redundant, i.e. they do not change the set of solutions of the problem, but their purpose is to remove infeasible values from the domains of the variables. Similar to bounds, there is one per family generic AMONG implied constraint that is parameterised by the pattern of a considered time-series constraint. Hence we only need to prove three AMONG implied constraints in order to further use them for 66 time-series constraints.
  This work was published in the proceedings of the *CP'17* conference [12], and the AMONG implied constraints for 66 time-series constraints were integrated in the Volume II of the Global Constraint Catalogue [10].

Example 1.6.2 (AMONG implied constraints, example adapted from [12])**.** Consider the MAX_SURF_DECREASING_SEQUENCE(X, R) time-series constraint, where X is a sequence of integer variables of length n, and R is constrained to be the maximum of the sums of the elements of the decreasing sequences of X. For example, the sequence $\langle 1, 2, 1, 0, 0, -1, -2, 2, 4, 2, 2\rangle$ has two decreasing sequences, namely $\langle 2, 1, 0, 0, -1, -2\rangle$ and $\langle 4, 2\rangle$, with a sum of elements 0 and 6, respectively. The maximum of these two values is 6, and thus R is fixed to 6.
Now assume that the value of R is known and is equal to, for example 18, but X is unknown, and our goal is to find a sequence X of 7 integers, which are all in $[1, 4]$, such that X yields 18 as the value of R. By enumerating all integer sequences satisfying these restrictions, we observe that any such integer sequence contains a single decreasing sequence with at least 4 its elements being 3 or 4. This allows us to the state the AMONG$(N, \langle X_1, X_2, \ldots, X_7\rangle, \langle 3, 4\rangle)$ constraint with $N \geq 4$, which

means that the number N of occurrences of the values 3 and 4 in the sequence $\langle X_1, X_2, \ldots, X_7 \rangle$ is at least 4.
The parameters of the AMONG implied constraint, i.e. $\langle 3, 4 \rangle$ in this example, and a lower bound on N are obtained from a generic formula, parameterised by the pattern associated with a time-series constraint. △

- [**Parameterised linear implied inequalities** linking the result values of a conjunction of time-series constraints imposed on the same time series of length n]
  We explore the relations between the result values of *several* time-series constraints imposed on the same time series. We call these inequalities *linear invariants*.
  This work was published in the proceedings of the *CP'17* conference [13], and the obtained linear inequalities were integrated in the database of invariants of the Volume II of the Global Constraint Catalogue [10].

  Example 1.6.3 (Linear invariants)**.** Consider the conjunction of NB_DECREASING_SEQUENCE(X, R_1) and NB_INCREASING_SEQUENCE(X, R_2) imposed on the same sequence of variables X of length n, where R_1 (respectively R_2) is constrained to be the number of decreasing (respectively increasing) sequences in X. An increasing sequence in X is a maximal inclusion-wise monotonously increasing subsequence of X. Since between any two consecutive increasing sequences there is exactly one decreasing sequence and vice versa, the linear inequalities $R_1 \leq R_2 + 1$ and $R_2 \leq R_1 + 1$ hold for any sequence X of integers. In addition, the total number of decreasing and increasing sequences in an integer sequence of length n cannot exceed n. Hence the linear inequality $R_1 + R_2 \leq n$ holds for any sequence X of any length n.
  We extract such linear invariants using *register automata* associated with the corresponding time-series constraints. △

- [**Parameterised non-linear invariants** linking the result values of a conjunction of time-series constraints imposed on the same time series of length n]
  Such invariants characterise sets of infeasible combinations of the result values of the time-series constraints in a conjunction that cannot be described as a linear combination of the result values of the conjunction of time-series constraints and n. In other words, these are sets of infeasible combinations that are located within the convex hull of feasible combinations.
  The obtained non-linear invariants were integrated in the database of invariants of the Volume II of the Global Constraint Catalogue [10].

  Example 1.6.4 (Non-linear invariants)**.** Consider the conjunction of SUM_WIDTH_DECREASING_SEQUENCE(X, R_1) and SUM_WIDTH_INCREASING_SEQUENCE(X, R_2) imposed on the same sequence of variables X of length n, where R_1 (respectively R_2) is constrained to be the sum of the number of elements of all decreasing (respectively increasing) sequences in X. For example, the integer sequence $\langle 1, 2, 1, 0, 0, -1, -2, 2, 4, 2, 2 \rangle$ has two decreasing sequences, namely $\langle 2, 1, 0, 0, -1, -2 \rangle$ and $\langle 4, 2 \rangle$, with 6 and 2 elements, respectively, i.e. of width 6 and 2; and two increasing sequences, namely $\langle 1, 2 \rangle$ and $\langle -2, 2, 4 \rangle$, with 2 and 3 elements, respectively, i.e. of width 2 and 3. Hence R_1 (respectively R_2) is constrained to be the sum of 6 and 2 (respectively 2 and 3), which is 8 (respectively 5).
  By generating all feasible combinations of R_1 and R_2 for sequences of length 9, 10, 11 and 12, we observe that there are quite a few infeasible pairs of R_1 and R_2 that are located within the convex hull of feasible pairs. The generated combinations are reported in Figure 1.3. Our goal is to synthesise and prove such generic non-linear invariants linking R_1, R_2 and parameterised by a function of n stating that the points depicted by red circles in Figure 1.3 are infeasible. For example, the five invariants that we obtain for the considered pair of constraints are
  - $R_1 \neq 1$,

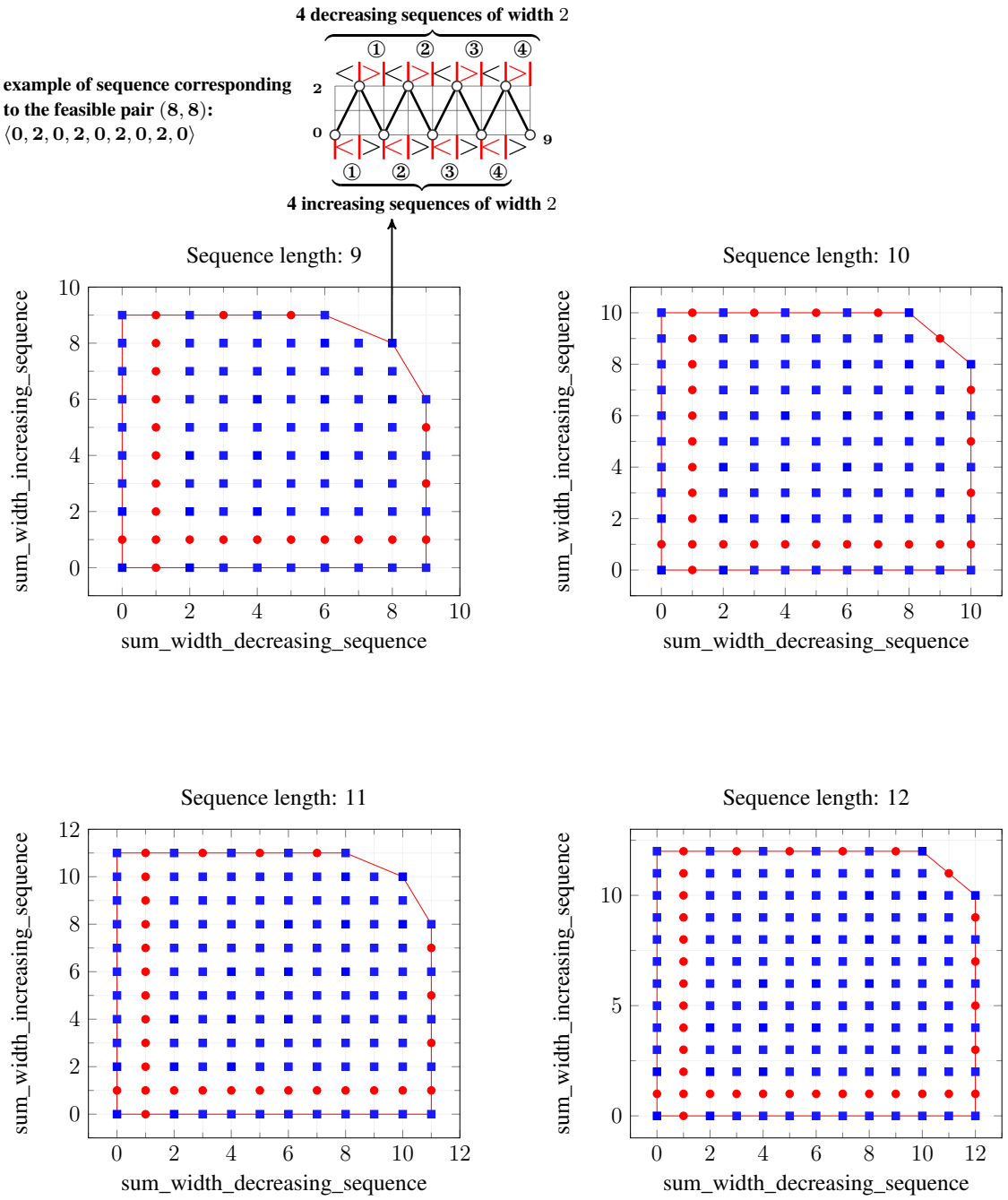


Figure 1.3 – Feasible (blue squares) and infeasible (red circles) combinations of the results values R_1 and R_2 of the constraints SUM_WIDTH_DECREASING_SEQUENCE(X, R_1) and SUM_WIDTH_INCREASING_SEQUENCE(X, R_2) imposed on the same sequence X whose length is in $\{9, 10, 11, 12\}$. We represent only infeasible combinations that are located within the convex hull of all feasible combinations.

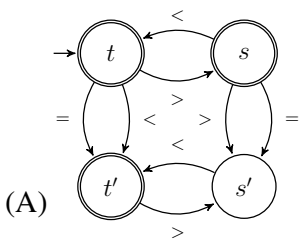
- $R_2 \neq 1$,
- $R_1 \neq n \ \vee \ R_2 \bmod 2 = 0$,
- $R_2 \neq n \ \vee \ R_1 \bmod 2 = 0$,
- $n \bmod 2 = 1 \ \vee \ R_1 \neq n-1 \ \vee \ R_2 \neq n-1$.

Note that some of these invariants are parameterised by functions of n, namely $n-1$ and $n \bmod 2$. Note also that these invariants hold for any integer sequence X of any length n. △

- [**Constant-size automata** representing the set of all integer sequences satisfying some condition, e.g. all integer sequences with the maximum possible number of decreasing sequences for a given sequence length]
On the one hand, finite automata are used since the beginning of computer science to model many aspects of computation [81]. On the other hand, bounds are ubiquitous in a number of optimisation problems [88, 18], where they allow one to speed up the search process. While bounds are typically expressed as parameterised formulae [30, 14], the question of a compact and explicit representation of the set of *all solutions* reaching a particular bound went unnoticed. Such automata are a crucial part of our method for synthesising and proving non-linear implied constraints, mentioned in the previous item.
The obtained automata were integrated in the Volume II of the Global Constraint Catalogue [10].

Example 1.6.5 (Constant-size automata)**.** Consider the NB_DECREASING_SEQUENCE(X, R) time-series constraint introduced in Example 1.6.1, where X is a sequence of variables of length n. Recall that the maximum number of decreasing sequences in a sequence of length n is $\lfloor \frac{n}{2} \rfloor$. With any integer sequence X we can associate a sequence of binary relations in $\{<, =, >\}$ between every pair of its consecutive variables. We name such sequence the *signature* of X. For example, the signature of the sequence $\langle 1, 2, 0, 2, 3, -1 \rangle$ is $\langle <, >, <, <, > \rangle$. The constant-size automaton $\mathcal{M}$ in Part (A) of Figure 1.4 accepts the signatures of all and only all integer sequences with the maximum number of decreasing sequences, i.e. sequences for which the constraint NB_DECREASING_SEQUENCE$(X, \lfloor \frac{n}{2} \rfloor)$ holds. The state s is the initial state of $\mathcal{M}$, and the states s, t, and t' are accepting states. A transition labelled with a binary operator $\circ$ in $\{<, =, >\}$ is triggered iff for the current consecutive pairs of values X_i and X_{i+1}, the corresponding binary relation $\circ$ holds. Part (B) gives all the signatures of lengths 3 and 4 accepted by this automaton. △

- [**An extended transducer-based computational model** describing functions over integer sequences arising in the context of constraint programming]
The extended model covers time-series constraints, but also most sequence constraints in the Volume I of the Global Constraint Catalogue [21].



(B)

|>|<|>|	|>|< =|>|
=|>|<|>|	|>|<|> >|
<|>|<|>|	|>|<|>|=
|>|= <|>|	|>|<|>|<
|> >|<|>|	|>|< <|>|

Figure 1.4 – (A) Automaton accepting the signatures of all, and only all, integer sequences with the maximum number of decreasing sequences. (B) All signatures of lengths 3 and 4 accepted by the automaton in (A).

Summary of our Contributions:

- Systematic methods for synthesising objects, namely automata and formulae, e.g. bounds, linear and non-linear invariants, AMONG implied constraints that
  1. capture the combinatorial flavour of a considered time-series constraint or a conjunction of time-series constraints,
  2. are parameterised by a considered instance, i.e. the domains of the variables and the sequence length, and a considered time-series constraint,
  3. can be used in different contexts.
- Extension of a transducer-based model used for describing time-series constraints, which provides us with a uniform way of representing sequence constraints.

1.7 The Reading Grid of this Thesis

We present the plan of this thesis as well as a reading grid, which consists of three main parts:

1. *Background*, containing all necessary information for understanding this thesis. This includes topics such as regular expressions in Chapter 2, register automata and transducers in Chapter 3, constraint programming in Chapter 4, and time-series constraints in Chapter 5. When presenting time-series constraints we will consider both their declarative view, given in Section 5.1, and their operational view, given in Section 5.2.
2. *Theoretical Contributions*, which gives first a detailed overview of our contributions for synthesising combinatorial objects for time-series constraints and extending a transducer-based computational model, and then presents the following contributions:
   - Parameterised bounds, presented in Chapter 7.
     Considered context: constraint in isolation.
     Required background: Chapter 2 (regular expressions), Chapter 4 (constraint programming), Section 5.1 (declarative view of time-series constraints).
   - Parameterised AMONG implied constraints, presented in Chapter 8.
     Considered context: constraint in isolation.
     Required background: Chapter 2 (regular expressions), Chapter 4 (constraint programming), Section 5.1 (declarative view of time-series constraints).
   - Parameterised linear invariants, presented in Chapter 9.
     Considered context: conjunction of constraints.
     Required background: Chapter 3 (automata and register automata), Chapter 4 (constraint programming), Section 5.2 (operational view of time-series constraints).
   - Parameterised non-linear invariants, presented in Chapter 10.
     Considered context: conjunction of constraints.
     Required background: Chapter 3 (automata and register automata), Chapter 4 (constraint programming), Section 5.2 (operational view of time-series constraints).
   - Conditional constant-size automata, presented in Chapter 11.
     Considered context: conjunction of constraints.
     Required background: Chapter 3 (automata and register automata), Chapter 4 (constraint programming), Section 5.2 (operational view of time-series constraints).

- Extended transducer-based model, presented in Chapter 12.

3. *Practical Evaluation*, contains the practical evaluation of the impact of the synthesised combinatorial objects.

Part I

Background

In this part, we give the necessary background for understanding this thesis. We now introduce the chapters of this part and explain their importance in the context of this work:

- As mentioned in Chapter 1, a pattern is one of the three main components of a time-series constraint. From a formal point of view, a pattern is a *regular expression* [57], which describes a *regular language*.
  Chapter 2 gives background on regular expressions and regular languages.
- From an operational point of view, regular languages can be represented by *finite automata* [80]. A finite automaton consists of a finite number of states and transitions between states, labelled with input symbols. It consumes an input sequence and either accepts this sequence or fails. With every automaton we can associate a regular expression whose regular language is accepted by this automaton and vice versa. *Finite transducers* [119] are automata that not only consume an input sequence but also produce an output sequence. *Register automata* [20] are automata augmented with a constant number of registers that are used to perform computations over an input sequence, e.g. count the number of decreasing sequences in an integer sequence.
  Chapter 3 gives background on automata, transducers, and register automata.
- An important notion of CP is a *constraint satisfaction problem* (CSP) [118], which consists of variables with finite domains and constraints. Typically in the CP context we are searching for a solution to a CSP using filtering techniques for constraints of the problem. Automata and register automata can be used to filter some global constraints. For a sequence of variables of a fixed length, an automaton or a register automaton can be decomposed as a conjunction of logical constraints [29]. The number of variables and constraints in such a conjunction depends linearly on the length of an input sequence.
  Chapter 4 gives a formal definition of constraint satisfaction problem, mentions the most common techniques for solving a CSP, and also gives examples of global constraints, for which automata and register automata are used for both describing these constraints and also for filtering them.
- Time-series constraints [22] are a central part of the work of this thesis. Due to their compositional nature, a single definition is used for obtaining more than 200 constraints for 22 patterns. Register automata can be used to filter time-series constraints. Because of the large number of time-series constraints we have to synthesise register automata in a systematic way as follows:

  1. Generate a finite transducer whose output sequence identifies all maximal occurrences of the pattern [68].
  2. Replace in the transducer every output symbol with a set of register updates corresponding to the feature and the aggregator [22].

  Chapter 5 gives a formalisation of time-series constraints.

Chapter 2

Background on Regular Expressions

In this chapter, which is adapted from [14], we give the background on *regular expressions*, which are one of the key ingredients of time-series constraints.

An *alphabet* $\mathcal{A}$ is a finite set of symbols, and a symbol of $\mathcal{A}$ is called a *letter*. A *word* on $\mathcal{A}$ is a finite sequence of symbols belonging to $\mathcal{A}$. The empty word is denoted by ε. The *length* of a word w is the number of letters in w and is denoted by $|w|$. For $i \in [1, |w|]$, $w[i]$ denotes the i^{th} letter of a word w. The concatenation of two words is denoted by putting them side by side, with an implicit infix operator between them. A word w is a *factor* of a word x if there exist two words v and z such that $x = vwz$; when $v = \varepsilon$, w is a *prefix* of x, when $z = \varepsilon$, w is a *suffix* of x. If both w is not empty and different from x, then it is a *proper* factor of x. Given a word w and a positive integer $k > 0$, w^k denotes the *concatenation of k occurrences* of w. Given an integer k and a language $\mathcal{L}$, $\mathcal{L}^k$ is defined by $\mathcal{L}^0 = \{\varepsilon\}$, $\mathcal{L}^1 = \mathcal{L}$ and $\mathcal{L}^k = \mathcal{L} \cdot \mathcal{L}^{k-1}$ where '$\cdot$' is the concatenation operator. Then the Kleene closure of $\mathcal{L}$ is defined by $\cup_{n \geq 0} \mathcal{L}^n$ and denoted by $\mathcal{L}^*$.

Definition 2.0.1 (Regular expression [57])**.** A regular expression r on an alphabet $\mathcal{A}$ and the language $\mathcal{L}_r$ it describes, the regular language, are recursively defined as follows:

(1) 0 and 1 are regular expressions that respectively describe $\emptyset$ (the empty set) and $\{\varepsilon\}$.

(2) For every letter ℓ of $\mathcal{A}$, ℓ is a regular expression that describes the singleton $\{\ell\}$.

(3) If r_1 and r_2 are regular expressions, respectively describing the regular languages $\mathcal{L}_{r_1}$ and $\mathcal{L}_{r_2}$, then $r_1 + r_2$, $r_1 \cdot r_2$ and r_1^* are regular expressions that respectively describe the regular languages $\mathcal{L}_{r_1} \cup \mathcal{L}_{r_2}$, $\mathcal{L}_{r_1} \cdot \mathcal{L}_{r_2}$, and $\mathcal{L}_{r_1}^*$.

Example 2.0.1 (Regular expressions over the alphabet associated with time-series constraints)**.** Consider the alphabet $\Sigma = \{\text{'}<\text{'}, \text{'}=\text{'}, \text{'}>\text{'}\}$.

- DECREASING $=$ '$>$' is a regular expression on Σ. The word $v =$ '$>$' is a word of length 1 on Σ that belongs to $\mathcal{L}_{\text{DECREASING}}$, and it does not have any proper factors. The word '$>>$' is a word of length 2 on Σ, which does not belong to $\mathcal{L}_{\text{DECREASING}}$.
- INFLEXION $=$ '$< (< \mid =)^* > \mid > (> \mid =)^* <$' is a regular expression on Σ. The word $v =$ '$>=<$' is a word of length 3 on Σ that belongs to $\mathcal{L}_{\text{INFLEXION}}$. The word v has multiple proper factors, e.g. '$>$', '$<$'. The word '$>=<<$' does not belong to $\mathcal{L}_{\text{INFLEXION}}$ since it finishes with the suffix '$<<$'. △

Definition 2.0.2 (Non-fixed length regular expression)**.** A regular expression r is a *non-fixed length regular expression* if not all words of $\mathcal{L}_r$ have the same length.

Example 2.0.2 (Fixed and non-fixed length regular expressions)**.** We give two examples of regular expressions, a first one with a fixed length and a second one with a non-fixed length.

- The DECREASING $=$ '$>$' regular expression has a fixed length since $\mathcal{L}_{\text{DECREASING}}$ contains a single word.
- The INFLEXION $=$ '$< (< \mid =)^* > \mid > (> \mid =)^* <$' regular expression does not have a fixed length since $\mathcal{L}_{\text{INFLEXION}}$ contains words of different lengths. △

Definition 2.0.3 (Disjunction-capsuled regular expression)**.** A regular expression over an alphabet $\mathcal{A}$ is *disjunction-capsuled* if it is in the form of '$r_1 r_2 \dots r_p$', where every r_i (with $i \in [1, p]$) is, either a letter of the alphabet $\mathcal{A}$, or a regular expression whose regular language contains the empty word.

Note that Definition 2.0.3 is a slight extension of a similar notion introduced in [83]. If a regular expression σ over an alphabet Σ is disjuntion-capsuled, then there is a single shortest word $a_1 a_2 \dots a_k$ in the language of σ with every a_i being a letter in Σ, and every word v in $\mathcal{L}_\sigma$ can be decomposed as $v = v_1 a_1 v_2 a_2 v_3 \dots v_k a_k v_{k+1}$ with all v_i being words in Σ^*. This is an important property that we will use when deriving a lower bound on the result values of time-series constraints in Chapter 7.

Example 2.0.3 (Disjunction-capsuled regular expression)**.** Table 5.2 on page 45 recalls the 22 regular expressions used for describing time-series constraints in [10, 22]. Every regular expression σ in column 2 of Table 5.2 is in the form of $\sigma = \sigma_1 | \sigma_2 | \dots | \sigma_t$ with $t \geq 1$, and every σ_i (with $i \in [1, t]$) is a disjunction-capsuled regular expression. Then $\mathcal{L}_\sigma$ is the union of the $\mathcal{L}_{\sigma_i}$ (with $i \in [1, t]$).

The '$(> \mid > (> \mid =)^* >)(< \mid < (< \mid =)^* <)$' regular expression has the same regular language as GORGE, but is not disjunction-capsuled. △

Chapter 3

Background on Automata, Register Automata and Transducers

In this section, we give the background on automata, register automata and transducers:

- In Section 3.1, we recall the notions of *deterministic finite automaton (DFA)*, *register automaton*, and *finite transducer*.
- In Section 3.2, we recall operations on automata and register automata such as intersection, complement, and union.

3.1 Defining Automata, Register Automata and Transducers

In this section, we recall in Definition 3.1.1 the notion of *deterministic finite automaton (DFA)* or simply automaton, in Definition 3.1.2 the notion of *register automaton*, and in Definition 3.1.3 the notion of *finite transducer*.

Definition 3.1.1 (DFA [80])**.** A *deterministic finite automaton (DFA)* or just *automaton* $\mathcal{M}$ is a tuple $\langle Q, \Sigma, \delta, q_0, A\rangle$, where

- Q is a finite set of *states*.
- Σ is a finite *input alphabet*.
- $\delta\colon Q \times \Sigma \to Q$ is the *transition function* defining the set of *transitions* of $\mathcal{M}$. Note that δ is not necessarily a total function. There is a transition in $\mathcal{M}$ from a state $q_1 \in Q$ to a state $q_2 \in Q$ labelled with $s \in \Sigma$ iff $\delta(q_1, s) = q_2$.
- $q_0 \in Q$ is the *initial state*.
- $A \subseteq Q$ is the *set of accepting states*.

An input word $w = w_1 w_2 \dots w_k \in \Sigma^*$ is *accepted* or *recognised* by $\mathcal{M}$ iff, upon the right-to-left consumption of the letters of w, $\mathcal{M}$ triggers the following sequence of transitions:

$$q_0 \xrightarrow{\delta(q_0,w_1)} q_1 \xrightarrow{\delta(q_1,w_2)} q_2 \dots q_{k-1} \xrightarrow{\delta(q_{k-1},w_k)} q_k,\ q_k \in A$$

If upon consuming the letters of w, the automaton $\mathcal{M}$ either visits a state q_i such that $\delta(q_i, w_{i+1})$ is undefined, or the last visited state q_k is not an accepting state, then we say that $\mathcal{M}$ *fails* on w.

Definition 3.1.2 (register automaton [20])**.** A *register automaton* $\mathcal{M}$ with $p > 0$ registers $\langle R_1, R_2, \dots, R_p\rangle$ is a tuple $\left\langle Q, \Sigma, q_0, R^0, \hat{\delta}, A, \alpha \right\rangle$, where

- Q is the finite set of *states*.
- Σ is the finite *input alphabet*.
- $q_0 \in Q$ is the *initial state*.
- $R^0 = \langle R_1^0, R_2^0, \dots, R_p^0\rangle$ is the vector of the initial values of the registers $\langle R_1, R_2, \dots, R_p\rangle$.

- $\hat{\delta}\colon (Q \times \mathbb{Z}^p) \times \Sigma \to Q \times \mathbb{Z}^p$ is the *transition function*, which defines the transitions of $\mathcal{M}$, and also the register updates upon these transitions. There is a transition in $\mathcal{M}$ from a state $q_1 \in Q$ to a state $q_2 \in Q$ labelled with $s \in \Sigma$ and $\langle R'_1, R'_2, \ldots, R'_p \rangle$ are the new values of the registers iff $\hat{\delta}(q_1, \langle R_1, R_2, \ldots, R_p \rangle, s) = (q_2, \langle R'_1, R'_2, \ldots, R'_p \rangle)$.
- $A \subseteq Q$ is the *set of accepting states*.
- $\alpha\colon Q \times \mathbb{Z}^p \to \mathbb{Z}^h$ is a function, called *acceptance function*, which maps the final state and the last values of the registers $\langle R_1, R_2, \ldots, R_p \rangle$ into an integer vector of length h. If h is 1 then we will treat this vector as an integer.

An input word $w = w_1 w_2 \ldots w_k \in \Sigma^*$ is *accepted* or *recognised* by $\mathcal{M}$ and it returns the resulting vector H iff, upon the right-to-left consumption of the letters of w, $\mathcal{M}$ triggers the following sequence of transitions:

$$q_0 \xrightarrow{\hat{\delta}(q_0, \langle R_1^0, R_2^0, \ldots, R_p^0 \rangle, w_1)} q_1 \xrightarrow{\hat{\delta}(q_1, \langle R_1^1, R_2^1, \ldots, R_p^1 \rangle, w_2)} q_2 \ldots q_{k-1} \xrightarrow{\hat{\delta}(q_{k-1}, \langle R_1^{k-1}, R_2^{k-1}, \ldots, R_p^{k-1} \rangle, w_k)} q_k,$$

where every $(q_i, \langle R_1^i, R_2^i, \ldots, R_p^i \rangle)$ (with i in $[1, k]$) is the result of $\hat{\delta}(q_{i-1}, \langle R_1^{i-1}, R_2^{i-1}, \ldots, R_p^{i-1} \rangle, w_i)$, q_k is an accepting state of $\mathcal{M}$, and $\alpha(q_k, \langle R_1^k, R_2^k, \ldots, R_p^k \rangle)$ is equal to H.

If upon consuming the letters of w, $\mathcal{M}$ either visits a state q_i such that $\hat{\delta}(q_i, \langle R_1, R_2, \ldots, R_p \rangle, w_{i+1})$ is undefined, or the last visited state q_k is not an accepting state, then we say that $\mathcal{M}$ *fails* on w.

Definition 3.1.3 (finite transducer [119])**.** A *finite transducer* $\mathcal{S}$ is a tuple $\langle Q, \Sigma, \Omega, \delta', A \rangle$, where

- Q is the finite set of *states*.
- Σ is the finite *input alphabet*.
- Ω is the finite *output alphabet*.
- $\delta'\colon Q \times \Sigma \to Q \times \Omega$ is the *transition function*, which defines the transition of $\mathcal{S}$. There is a transition in $\mathcal{S}$ from a state $q_1 \in Q$ to a state $q_2 \in Q$ labelled with an input symbol $s \in \Sigma$ and a finite sequence of output symbols $t \in \Omega^*$ iff $\delta'(q_1, s) = (q_2, t)$.
- $q_0 \in Q$ is the *initial state*.
- $A \subseteq Q$ is the *set of accepting states*.

An input word $w = w_1 w_2 \ldots w_k \in \Sigma^*$ is *accepted* or *recognised* by $\mathcal{S}$ and $\mathcal{S}$ produces the output sequence $\langle t_1, t_2, \ldots, t_k \rangle$ on w iff upon the right-to-left consumption of the letters of w, $\mathcal{S}$ triggers the following sequence of transitions:

$$q_0 \xrightarrow[t_1]{\delta'(q_0, w_1)} q_1 \xrightarrow[t_2]{\delta'(q_1, w_2)} q_2 \ldots q_{k-1} \xrightarrow[t_k]{\delta'(q_{k-1}, w_k)} q_k,$$

where every (q_i, t_i) (with i in $[1, k]$) is the result of $\delta'(q_{i-1}, w_i)$, and q_k is an accepting state of $\mathcal{S}$.

If upon consuming the letters of w, $\mathcal{S}$ either visits a state q_i such that $\delta'(q_i, w_{i+1})$ is undefined, or the last visited state q_k is not an accepting state, then we say that $\mathcal{S}$ *fails* on w.

Picturing automata, register automata and transducers. In all figures of this thesis, states of automata, register automata and transducers are pictured as circles, and accepting states are denoted by double circles. The initial state is denoted by an arrow coming from nowhere. A transition is denoted by a line or curved arrow. For register automata, the acceptance function is depicted by a box connected by dotted lines to each accepting state. If a register is left unchanged while triggering a given transition, then we do not mention this register update on the corresponding transition. For transducers, every transition is labelled with a symbol of the input alphabet followed by a colon and a word whose letters belong to the output alphabet. When the input alphabet is $\{\text{`}<\text{'}, \text{`}=\text{'}, \text{`}>\text{'}\}$, a transition labelled with the '$\geq$' (respectively '$\leq$') input symbol is a shorthand for two parallel transitions labelled with '$>$' (respectively '$<$') and '$=$', respectively.

Automata and register automata are often used for checking properties of integer sequences or computing quantities from integer sequences, e.g. an automaton accepting only monotonously decreasing

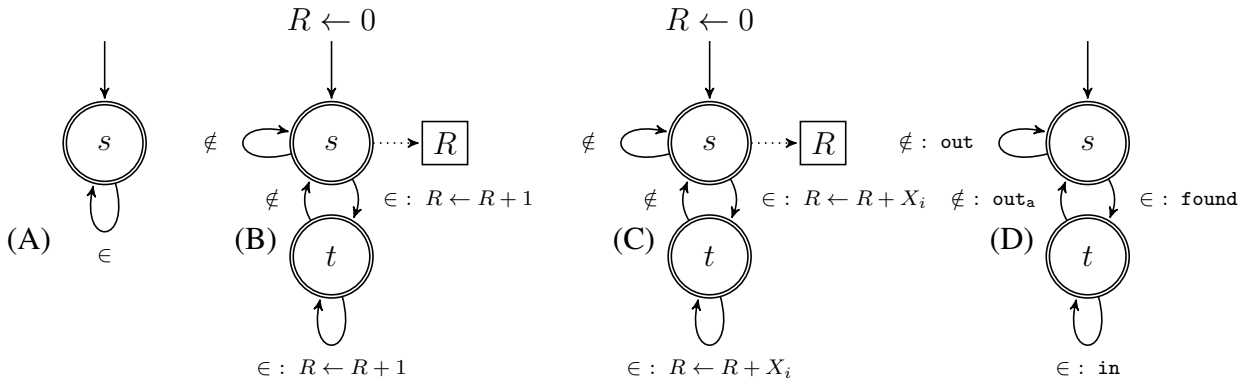


Figure 3.1 – In the three figures, the signature of an integer sequence $\langle X_1, X_2, \ldots, X_n\rangle$ is defined by $S_i = \text{`}\notin\text{'} \Leftrightarrow X_i \notin \{-1, 0, 5\} \wedge S_i = \text{`}\in\text{'} \Leftrightarrow X_i \in \{-1, 0, 5\}$. (A) Automaton recognising the signatures of integer sequences whose elements are in $\{-1, 0, 5\}$. (B) Register automaton recognising any signature and returning the number of elements in $\{-1, 0, 5\}$ in an integer sequence. (C) Register automaton recognising any signature and returning the sum of elements in $\{-1, 0, 5\}$ in an integer sequence. (D) Transducer with the input alphabet $\{\in, \notin\}$ and the output alphabet $\{\texttt{found}, \texttt{out}, \texttt{in}, \texttt{out}_\texttt{a}\}$.

sequences of integers, a register automaton computing the number of monotonously decreasing subsequences in an integer sequence. Usually in this case every element of a considered integer sequence $\langle X_1, X_2, \ldots, X_n\rangle$ is mapped into a letter of the input alphabet Σ of the considered automaton or register automaton. In order to generalise the implicit condition $X_i = s$, with s being in Σ, we use the notion of the *signature* of an integer sequence.

Definition 3.1.4 (signature, arity)**.** Consider a sequence of integer numbers $X = \langle X_1, X_2, \ldots, X_n\rangle$, and a function $S\colon \mathbb{Z}^p \to \Sigma$, where Σ is a finite set denoting an alphabet. Then, the *signature* of X is a sequence $\langle S_1, S_2, \ldots, S_{n-p+1}\rangle$, where every S_i equals $S(X_i, X_{i+1}, \ldots, X_{i+p-1})$. The constant p is called the *arity* of the signature. A signature of arity 1 (respectively 2) is called *unary* (respectively *binary*).

The next definition introduces the notion of *accepting sequence* wrt an automaton or a register automaton.

Definition 3.1.5 (accepting sequence wrt an automaton/a register automaton)**.** Consider an automaton or a register automaton $\mathcal{M}$ over an input alphabet Σ and an integer sequence X, whose signature is in Σ^*. The sequence X is called *accepting* wrt $\mathcal{M}$ iff $\mathcal{M}$ could consume the entire signature of X, and if $\mathcal{M}$ finishes in an accepting state.

Note that if a register automaton $\mathcal{M}$ is used for computing a quantity from an integer sequence X then the register updates defined by the transition function of $\mathcal{M}$ may depend on the values of X. The following example illustrates this point.

Example 3.1.1 (signature, arity, automaton, register automaton, finite transducer, accepting sequence)**.** Consider a unary signature S over the alphabet $\{\in, \notin\}$ such that for any integer sequence $\langle X_1, X_2, \ldots, X_n\rangle$, $S_i = \text{`}\notin\text{'} \Leftrightarrow X_i \notin \{-1, 0, 5\} \wedge S_i = \text{`}\in\text{'} \Leftrightarrow X_i \in \{-1, 0, 5\}$. For example, for the sequence $t = \langle 1, 2, -1, 3, 5, 0\rangle$, its signature is $\langle \notin, \notin, \in, \notin, \in, \in\rangle$.

The automaton in Part (A) of Figure 3.1 recognises signatures consisting only of '$\in$'. This automaton has a single state, which is its initial and accepting state, and a single transition. The integer sequence $\langle -1, -1, 0, -1\rangle$ is accepting wrt this automaton, but t is not.

The register automaton in Part (B) of Figure 3.1 recognises any signature, and it returns the number of occurrences of '$\in$' in this signature. After having consumed the signature of t, it returns 3. The register updates of this register automaton do not depend on the values in an integer sequence, and hence for any

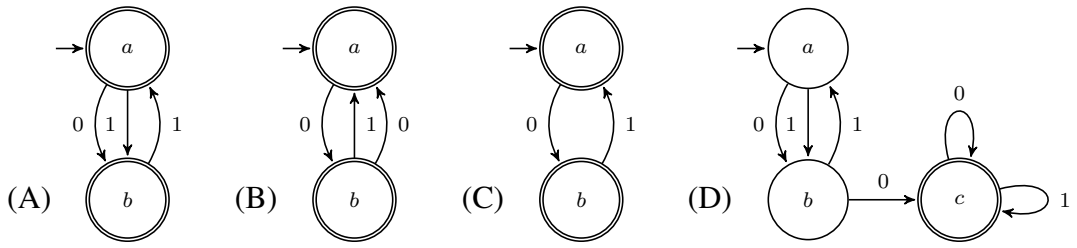


Figure 3.2 – (A) Automaton recognising sequences of 0 and 1, where 0 are only located at odd positions; (B) automaton recognising sequences of 0 and 1, where 1 are only located at even positions; (C) intersection of (A) and (B); (D) complement of (A).

integer sequence with the same signature, this register automaton will return the same value. Any integer sequence is accepting wrt this register automaton.

The register automaton in Part (C) of Figure 3.1 recognises any signature over the alphabet $\{\in, \notin\}$, and it returns the sum of elements in $\{-1, 0, 5\}$ of an input integer sequence. For example, after having consumed the signature of t, it returns 4. The register updates of this register automaton depend on the values in an input integer sequence, and for two integer sequences with the same signature, the register automaton does not necessarily return the same result.

Part (D) of Figure 3.1 gives a finite transducer, which recognises any signature over the alphabet $\{\in, \notin\}$ and returns a sequence of elements of $\{\texttt{found}, \texttt{out}, \texttt{in}, \texttt{out}_\texttt{a}\}$. For example, after having consumed the signature of t, it returns $\langle \texttt{out}, \texttt{out}, \texttt{found}, \texttt{out}_\texttt{a}, \texttt{found}, \texttt{in} \rangle$. Note that `found` indicates the start of a subsequence whose elements are in $\{-1, 0, 5\}$, and `in` indicates a continuation of such a subsequence. △

3.2 Operations on Automata and Register Automata

We shortly recall in Sections 3.2.1, 3.2.2, 3.2.3, three operations on automata, namely intersection, union, and complement, respectively. We will use the intersection of register automata when deriving linear invariants in Chapter 9, and all the three operations on automata when deriving non-linear invariants in Chapter 10. The notion of *language* of an automaton or *language* of a register automaton, given in Definition 3.2.1, will be used all through these definitions.

Definition 3.2.1 (automaton language)**.** The *language* of an automaton (respectively register automaton) is a set of signatures recognised by this automaton (respectively register automaton).

3.2.1 Intersection

Definition 3.2.2 (intersection of automata)**.** The *intersection* of k automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ is an automaton, denoted by $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$, whose language is the intersection of the languages of all $\mathcal{M}_i$.

Example 3.2.1 (intersection of automata)**.** Let us consider two automata $\mathcal{M}_1$ and $\mathcal{M}_2$ whose input alphabet is $\{0, 1\}$:

- A_1, given in Part (A) of Figure 3.2, recognises signatures in which '0' appears only in odd positions, e.g. $\langle 0, 1, 0, 1, 1 \rangle$. The language of A_1 is the regular language of '$((0|1)1)^*(0|1|\varepsilon)$'.
- A_2, given in Part (B) of Figure 3.2, recognises signatures in which '1' appears only in even positions, e.g. $\langle 0, 1, 0, 1, 0 \rangle$. The language of A_2 is the regular language of '$(0(0|1))^*(0|\varepsilon)$'.

The intersection $\mathcal{I}$ of $\mathcal{M}_1$ and $\mathcal{M}_2$ is an automaton recognising sequences of alternating '0' and '1' starting with '0', and is given in Part (C) of Figure 3.2. The language of $\mathcal{I}$ is the regular language of the '$(01)^*(0|\varepsilon)$' regular expression. △

Definition 3.2.3 (intersection of register automata [98])**.** The *intersection* of k register automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ is a register automaton, denoted by $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$, such that the following conditions all hold:

1. The language of $\mathcal{I}$ is the intersection of the languages of all $\mathcal{M}_i$.
2. The number of registers of $\mathcal{I}$ is equal to $\sum_{i=1}^{k} p_i$, where every p_i is the number of registers of $\mathcal{M}_i$.
3. When consuming any input signature S, for every register $A_{i,j}$ of $\mathcal{I}$, at every transition its value is equal to the value of the register j of $\mathcal{M}_i$ when consuming S.
4. For every input signature S, the register automaton $\mathcal{I}$ returns a tuple $\langle R_1, R_2, \ldots, R_k \rangle$, where R_i is the value returned by $\mathcal{M}_i$ after consuming S.

Example 3.2.2 (intersection of register automata)**.** Part (C) of Figure 3.3 contains the intersection $\mathcal{I}$ of the register automata $\mathcal{M}_1$ and $\mathcal{M}_2$ in Parts (A) and (B), respectively. Both $\mathcal{M}_1$ and $\mathcal{M}_2$ have one register, and thus $\mathcal{I}$ has two registers and returns a pair of values. △

3.2.2 Union

Definition 3.2.4 (union of automata)**.** The *union* of k automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ is an automaton, denoted by $U = \mathcal{M}_1 \cup \mathcal{M}_2 \cup \cdots \cup \mathcal{M}_k$, whose language is the union of the languages of all $\mathcal{M}_i$.

Example 3.2.3 (union of automata)**.** The union of the automata in Parts (A) and (B) of Figure 3.2 is an automaton recognising any sequence of '0' and '1'. △

3.2.3 Complement

Definition 3.2.5 (complement of an automaton)**.** The *complement* of an automaton $\mathcal{M}$ over an alphabet Σ is an automaton, denoted by $\mathcal{M}'$, whose language is the complement of the language of $\mathcal{M}$ wrt Σ^*.

Example 3.2.4 (complement of an automaton)**.** The automaton in Part (D) of Figure 3.2 is the complement of the automaton in Part (A). △

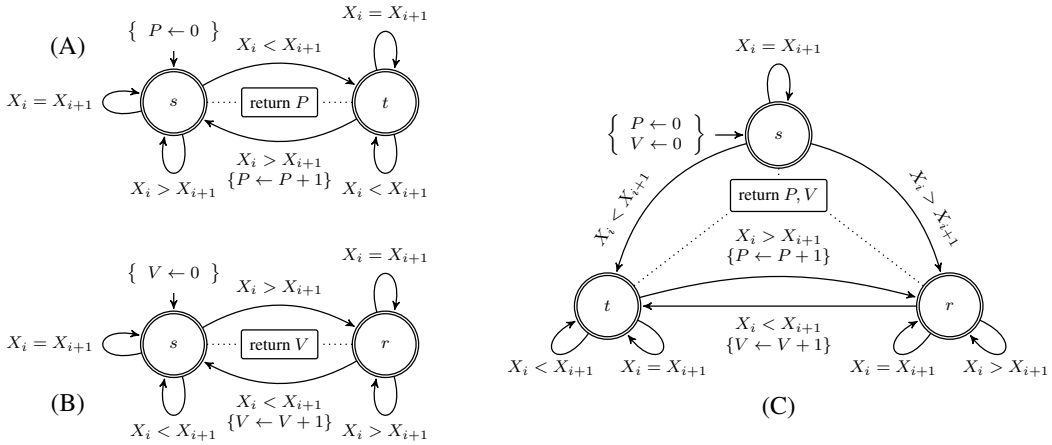


Figure 3.3 – This figure is adapted from [13]. (A) and (B) register automata over alphabet $\{<, =, >\}$, returning the number of maximal occurrences of PEAK and VALLEY (see Table 5.2 on page 45), respectively, in the signature $\langle S_1, S_2, \ldots, S_{n-1} \rangle$ of an integer sequence $\langle X_1, X_2, \ldots, X_n \rangle$, where every S_i is defined by the following constraints: $S_i =$ '$<$' $\Leftrightarrow X_i < X_{i=1} \wedge S_i =$ '$=$' $\Leftrightarrow X_i = X_{i=1} \wedge S_i =$ '$>$' $\Leftrightarrow X_i > X_{i=1}$. (C) Intersection of (A) and (B), both returning the number of maximal occurrences of PEAK and VALLEY, respectively, in the signature of $\langle X_1, X_2, \ldots, X_n \rangle$.

Chapter 4

Background on Constraint Programming

In this section, we give the background on constraint programming. We first define a *constraint satisfaction problem (CSP)* in Section 4.1. Then, we present the main components used for solving a CSP in Section 4.2, and a standard representation of a CSP in Section 4.3. Finally, in Section 4.4, we discuss the role of automata, and register automata in CP.

4.1 Constraints and Constraint Satisfaction Problems

In this section, we recall the key definitions of CP, namely *constraint* in Definition 4.1.1 and *constraint satisfaction problem* in Definition 4.1.2.

Definition 4.1.1 (constraint, parameters of a constraint, solution to a constraint [118])**.** Consider a set of integer variables $X_1, X_2, \ldots, X_m$, where variable X_i can take its values in a finite subset D_i of integer numbers, called the *domain* of X_i.

- A *constraint* γ is a restriction imposed on the variables $X_1, X_2, \ldots, X_m$, called *the scope* of γ.
- A constraint may depend on some integer numbers, called *parameters*.
- An assignment $c_i \in D_i$ of all the X_i *satisfies* the constraint γ iff the restriction imposed by γ is satisfied. Such an assignment of variables is also called a *solution* to γ.

In [21], a constraint is called *global* if it can be imposed on a non-fixed number of variables. The Volume I of the Global Constraint Catalogue [21] contains 443 global constraints. The Volume II of the Global Constraint Catalogue [10] contains 626 global constraints and is devoted to *time-series constraints*, which are the central part of this work and will be given in Chapter 5.

The following example illustrates the notions given in Definition 4.1.1 for the AMONG global constraint [25], which we will actively use in Chapter 8.

Example 4.1.1 (AMONG constraint)**.** Let us consider an $\text{AMONG}(N, \langle X_1, X_2, \ldots, X_n\rangle, \langle p_1, p_2, \ldots, p_k\rangle)$ global constraint [25], where N is an integer variable, $\langle X_1, X_2, \ldots, X_n\rangle$ is a sequence of n integer variables, and $\langle p_1, p_2, \ldots, p_k\rangle$ is a list of k parameters, i.e. integer numbers. The AMONG constraint restricts N to be the number of variables in $\langle X_1, X_2, \ldots, X_n\rangle$ whose values are in the list $\langle p_1, p_2, \ldots, p_k\rangle$. The scope of this constraint is the variables $N, X_1, X_2, \ldots, X_n$.

Consider the following instance $\text{AMONG}(N, \langle X_1, X_2, X_3, X_4, X_5\rangle, \langle 0, -1, 5\rangle)$ with $N \in \{0, 2, 3\}$, $X_1 \in \{0, 1, 2\}$, $X_2 \in \{6, 7, 8\}$, $X_3 \in \{-1\}$, $X_4 \in \{-3, 4, 8, 9\}$, and $X_5 \in \{3, 4, 5, 6\}$. No assignment of the variables with N being 0 is a solution to the constraint since the only possible value of X_3 is -1, and in any solution to the considered AMONG constraint, the value of N is at least 1. The assignment of N to 2, and of the $X = \langle X_1, X_2, X_3, X_4, X_5\rangle$ to $\langle 0, 7, -1, 4, 6\rangle$ is a solution to the constraint, since exactly two variables of X have their values in $\langle 0, -1, 5\rangle$, namely X_1 with the value 0, and X_3 with the value -1. △

Definition 4.1.2 (CSP, feasible/infeasible CSP [118])**.** A *constraint satisfaction problem* (CSP) consists of a set of variables V, their domains, and a set of constraints whose scopes are subsets of V.

- A *solution* to a CSP is an assignment of all the variables in V that simultaneously satisfies all the constraints.
- A CSP is *feasible* if it has at least one solution, and is *infeasible* otherwise.

Example 4.1.2 (CSP)**.** Consider a CSP defined by the variables with their domains $N \in \{0, 2, 3\}$, $X_1 \in \{0, 1, 2\}$, $X_2 \in \{6, 7, 8\}$, $X_3 \in \{-1\}$, $X_4 \in \{-3, 4, 8, 9\}$, $X_5 \in \{3, 4, 5, 6\}$, $M \in \{3, 4\}$, and the conjunction of two constraints imposed on subsets of these variables $\text{AMONG}(N, \langle X_1, X_2, X_3, X_4, X_5\rangle, \langle 0, -1, 5\rangle)$ and $\text{MAXIMUM}(\langle X_1, X_3\rangle, M)$, which restricts M to be the maximum of X_1 and X_3. Let us find a solution to this CSP. In Example 4.1.1, we showed that the assignment of N to 2 and the assignment of $\langle X_1, X_2, X_3, X_4, X_5\rangle$ to $\langle 0, 7, -1, 4, 6\rangle$ was a solution to the AMONG constraint. However, it is not a solution to the considered CSP, which has the two constraints AMONG and MAXIMUM, since the maximum of the values X_1 and X_3 is 0, and this value does not belong to the domain of M. It can be proved by using a constraint solver such as Choco [114], for example, that the considered CSP is infeasible. △

4.2 Solving a Constraint Satisfaction Problem

When we say "to solve a CSP" we mean to find a solution to this CSP or to prove its infeasibility. In general, saying whether a CSP has a solution or is infeasible is an *NP-complete* problem [131]. Hence, there does not exist a general *fast* algorithm which could be used for finding a solution to *any* CSP. Enumerating all possible assignments of the variables of a CSP would lead to a combinatorial explosion, and thus we need a better approach for finding a solution or proving the infeasibility of a CSP.

There are three common techniques for solving a CSP, namely *constraint propagation*, *search*, and *filtering algorithms*, which we further present.

Constraint Propagation. Consider a CSP over finite domain variables $X_1, X_2, \ldots, X_m$. *Constraint Propagation* can be defined as a mechanism of reduction of the domains of the variables $X_1, X_2, \ldots, X_m$ by removing values that cannot be a part of any solution. Note that constraint propagation does not perform any search.

As an example, consider a CSP with one constraint $X_1 \geq X_2$ over two variables $X_1 \in \{1, 2\}$ and $X_2 \in \{2, 3\}$. The constraint propagation will remove the value 1 from the domain of X_1 and 3 from the domain of X_2 since these values are not part of any solution.

The more values constraint propagation removes, the easier it is to further find a solution to a CSP since the domains are smaller and the solver needs to enumerate fewer combinations of variable-value assignments. There are various theoretical measures, called *consistencies*, of the quality of the domain reduction achieved by constraint propagation. One of the most used consistencies is the *generalised arc consistency* [118] or *domain consistency*, described in Definition 4.2.2.

Definition 4.2.1 (arc-consistent CSP [118])**.** Consider a CSP $\mathcal{C}$ over variables $X_1, X_2, \ldots, X_m$ whose domains are, respectively, $D_1, D_2, \ldots, D_m$. The CSP $\mathcal{C}$ is *arc-consistent* iff for every variable X_i (with i in $[1, m]$), for every d in D_i, there is an assignment $d_1, d_2, \ldots, d_{i-1}, d_{i+1}, \ldots, d_m$ of the variables $X_1, X_2, \ldots, X_{i-1}, X_{i+1}, \ldots, X_m$ such that $d_1, d_2, \ldots, d_{i-1}, d, d_{i+1}, \ldots, d_m$ is a solution to $\mathcal{C}$.

Definition 4.2.2 (achieving GAC)**.** Consider a CSP $\mathcal{C}$ over variables $X_1, X_2, \ldots, X_m$ whose domains are, respectively, $D_1, D_2, \ldots, D_m$. *Achieve generalised arc consistency (GAC)* for $\mathcal{C}$, or *maintain domain consistency* for $\mathcal{C}$, means to obtain a CSP $\mathcal{C}'$ such that the following three conditions hold:

1. The set of constraints and variables of $\mathcal{C}$ and $\mathcal{C}'$ coincides.
2. The set of solutions to $\mathcal{C}$ and $\mathcal{C}'$ coincides.
3. $\mathcal{C}'$ is arc-consistent.

In other words, when a CSP is arc-consistent it means that there are no values in the domains of the variables that are not part of at least one solution of this CSP. If a CSP is arc-consistent then such a CSP is feasible. Hence, in the context of solving a CSP, we aim to achieve GAC for each sub-problem corresponding to one of the constraints of the CSP.

For CSPs with a single global constraint, e.g. ALLDIFFERENT, achieving GAC is polynomial [116], whereas for CSPs with CUMULATIVE [2, 21] constraints, it is NP-hard.

Filtering algorithm. A *filtering algorithm* for a constraint is an algorithm that removes infeasible values from the domains of variables of a considered constraint. The goal of a filtering algorithm is to bring the considered constraint to GAC. In this case, the filtering is called *complete*. Designing dedicated filtering algorithms for a global constraint or a conjunction of global constraints is one of the most common way to solve a CSP. For example, the *sweep* technique is a generic approach for filtering a conjunction of constraints sharing some variables [19, 92].

Search. Although, constraint propagation can be used for removing the infeasible values from the domains of variables, and, in the best case, for stating whether a considered CSP is feasible or not, very often for finding a solution it is still required to perform *search*, i.e. to try out different assignments of variables until a solution to the CSP is found or infeasibility is proven.

Backtrack search is one of the most used forms of search in the context of CP. When performing backtrack search, a constraint solver gradually assigns values from the domains to the variables of a CSP. If a partial assignment at some point does not for sure satisfy at least one of the constraints of the CSP, then it is not part of any solution, and the solver does a *backtrack*, i.e. it goes to the closest previous partial assignment and assigns a different value to the last assigned variable, and then the process repeats. If at some point, all variables are assigned values from their domains, then the solver found a solution to the CSP.

The *number of backtracks* along with the *elapsed time* is the most used empirical criteria for assessing the quality of constraint propagation when solving a CSP. Usually, constraint propagation is triggered every time when a value is removed from the domain of a variable. Hence, the smaller is the number of backtracks, the more robust is the constraint propagation.

To accelerate the search different strategies can be used, for example, variables can be assigned values in different orders, or metaheuristics, e.g. limited discrepancy search [78], last-conflict-based reasoning [89].

4.3 Representation of a Constraint Satisfaction Problem

In order to develop methods for solving a given CSP one usually represents this CSP as a *hypergraph*. In the hypergraph of a CSP, every vertex corresponds to a variable of this CSP, and every hyperedge corresponds to a constraint. Using the hypergraph of a CSP one can develop dedicated methods for filtering this CSP that draws full benefit of the structure of the hypergraph. There were several works exploiting the structure of a hypergraph, for which the corresponding CSP can be solved in a polynomial time. For example, a CSP whose hypergraph has a bounded treewidth can be solved in a polynomial time [70, 59]. Another condition comes from database theory and states that if the hypergraph is Berge-acyclic, then the corresponding CSP can be solved in a polynomial time [34].

The hypergraph representation focusses more on a flat description of a network of constraints, while this thesis studies a class of constraints defined in a compositional way by a cascade of functions. Exploring the structure of reformulation of this class of constraints may be one of the directions for a future work.

4.4 Automata and Register Automata in Constraint Programming

In the context of constraint programming, often a checker for a constraint can be represented as an automaton or as a register automaton. This is the case when for a constraint we can construct an automaton

such that every word accepted by such automaton corresponds to a set of solutions to the constraint, and every solution to the constraint corresponds to a word accepted by the automaton.

Example 4.4.1 (register automaton as a checker)**.** Consider the AMONG$(N, \langle X_1, X_2, \ldots, X_n\rangle, \langle 0, -1, 5\rangle)$ global constraint, introduced in Example 4.1.1. A register automaton $\mathcal{M}$ for this constraint is given in Part (B) of Figure 3.1. For an integer sequence $X = \langle X_1, X_2, \ldots, X_n\rangle$, and an integer value R the constraint holds iff after consuming the signature of X, the last value of the register R of $\mathcal{M}$ is equal to N. △

Automata with or without registers of constant size, i.e. the number of states and the input alphabet do not depend on the length of an input sequence, allow a compact and homogenous representation for many constraints, e.g. AMONG [25], PATTERN [43], and all time-series constraints [22]. Such a representation of constraints can be used for propagating them. For some global constraints automata and register automata are also used for defining them.

We now present the REGULAR global constraint, which is defined by an automaton, and its three generalisations, namely COST-REGULAR and MULTI-COST-REGULAR, which are defined by an automaton and a cost matrix, and AUTOMATON, which is defined by a register automaton. Generalisation of constraints using automata was motivated first by representing different grammar types of the Chomsky hierarchy [54] in the context of constraint programming, and second by practical needs, e.g. cost matrices are often required in the context of optimisation, register automata have a greater expressive power compared to automata.

4.4.1 REGULAR Global Constraint

We first consider the REGULAR global constraint, which was introduced in [109], and, for example, is common in nurse rostering problems [127]. This constraint is available in many constraint solvers, e.g. Choco, SICStus, Gecode, OR-tools. It is also implemented in the MiniZinc constraint modelling language [102].

Definition 4.4.1 (REGULAR [109])**.** Let $\mathcal{M} = (Q, \Sigma, \delta, q_0, A)$ denote an automaton, and $\langle X_1, X_2, \ldots, X_n\rangle$ be a sequence of integer variables with every X_i ranging over a finite integer domain $D_i \subseteq \Sigma$. The REGULAR$(\langle X_1, X_2, \ldots, X_n\rangle, \mathcal{M})$ constraint holds iff $\langle X_1, X_2, \ldots, X_n\rangle$ is accepted by $\mathcal{M}$.

Pesant presented in [109] a filtering algorithm for the REGULAR$(\langle X_1, X_2, \ldots, X_n\rangle, \mathcal{M})$ constraint, which achieves GAC. The algorithm creates a *layered directed graph* G, where each node (i, j) corresponds to a domain value j of a variable X_i. By doing two passes through the list of arcs of G, the authors removes some arcs and nodes from G in a way so that for the obtained graph G', an assignment $\langle j_1, j_2, \ldots, j_n\rangle$ of a considered sequence of variables $\langle X_1, X_2, \ldots, X_n\rangle$ is feasible iff there is a path in G' formed by nodes $(1, j_1), (2, j_2), \ldots, (n, j_n)$. Hence the graph G' represents the set of all and only all solutions to the constraints.

4.4.2 COST-REGULAR and MULTI-COST-REGULAR Global Constraints

The COST-REGULAR and MULTI-COST-REGULAR global constraints are extensions of REGULAR, introduced in [60] and [99], respectively.

Definition 4.4.2 (COST-REGULAR [60])**.** Let $\mathcal{M} = (Q, \Sigma, \delta, q_0, A)$ denote an automaton, let $\langle X_1, X_2, \ldots, X_n\rangle$ be a sequence of integer variables with every X_i ranging over a finite integer domain $D_i \subseteq \Sigma$, and let C be a cost matrix associating an integer with every transition of $\mathcal{M}$. The COST-REGULAR$(\langle X_1, X_2, \ldots, X_n\rangle, \mathcal{M}, z, C)$ global constraint holds iff $\langle X_1, X_2, \ldots, X_n\rangle$ is accepted by $\mathcal{M}$, z is equal to the sum of costs of every transition triggered by $\mathcal{M}$ upon consuming X.

The filtering algorithm for COST-REGULAR [60] is also based on the idea of a layered directed graph as in [109], but this time every arc of this graph is augmented with an integer weight. Note that giving a cost

matrix is equivalent as extending the automaton with one register that, on every transition, is incremented by a constant depending on this transition.

The MULTI-COST-REGULAR global constraint [99] is an extension of COST-REGULAR, where each transition of an automaton is associated with several integers.

4.4.3 AUTOMATON Global Constraint

The AUTOMATON global constraint is defined by a register automaton [29].

Definition 4.4.3 (AUTOMATON [29])**.** Let $\mathcal{M} = (Q, \Sigma, q_o, I, \hat{\delta}, A, \alpha)$ denote a register automaton, $\mathcal{S}\colon \mathbb{Z}^p \to \Sigma$ be a signature function, $X = \langle X_1, X_2, \ldots, X_n \rangle$ be a sequence of integer variables with every X_i ranging over a finite integer domain $D_i \subseteq \mathbb{Z}$, and R be an integer variable. The AUTOMATON$(X, \mathcal{M}, \mathcal{S}, R)$ global constraint holds iff after having consumed the signature of X the register automaton $\mathcal{M}$ returns R.

In order to propagate an AUTOMATON constraint, in [29], the authors encode the corresponding register automaton as a conjunction of logical constraints. The NP-completeness of an AUTOMATON constraint was proved in [27] by reduction to the *Subset Sum* problem [72]. Hence, there is no, in the general case, known efficient algorithm for achieving GAC for the AUTOMATON constraint. However, when there are no registers and the signature is unary, the reformulation of [29] is Berge-acyclic and can be solved in a polynomial time [34].

The AUTOMATON constraint is not as common as, for example, the REGULAR constraint. It is available in SICStus Prolog and SWI Prolog.

Chapter 5

Background on Time-Series Constraints

We are now ready to give a notion used throughout this work, namely a *time-series constraint*, which is a functional constraint imposed on a *time series*, and on an integer variable, called the *result variable*. In the context of our work, a *time series* is a sequence of integers corresponding to measurements taken over time, as it was defined in [22]. Time series appear in many real-life applications, e.g. trace analysis for Internet Service Provider, anomaly detection and error correction in building data, analysis of output of electric power stations over multiple days [28], power management for large-scale distributed systems [26], staff scheduling at a call centre [11].

A notion related to a time series is its *signature*, which is a special case of signature introduced in Definition 3.1.4.

Definition 5.0.4 (signature of a time series [22])**.** Consider a time series $X = \langle X_1, X_2, \ldots, X_n \rangle$, the signature of X is the sequence $S = \langle S_1, S_2, \ldots, S_{n-1} \rangle$, where every S_i is defined by the following constraint:

$$(X_i < X_{i+1} \Leftrightarrow S_i = \text{`<'}) \wedge\ (X_i = X_{i+1} \Leftrightarrow S_i = \text{`='}) \wedge\ (X_i > X_{i+1} \Leftrightarrow S_i = \text{`>'})$$

Example 5.0.2. The signature of the time series $\langle 0, 1, 2, 2, 0, 0, 4, 1 \rangle$ is $\langle <, <, =, >, =, <, > \rangle$. △

This section is organised as follows:

- First, in Section 5.1, we present the definition of time-series constraints.
- Second, in Section 5.2, we present the operational view of time-series constraints: the transducers used to synthesise an implementation of time-series constraints in the form of register automata. In this section, we also recall the notion of *glue constraint*, which is indispensable for filtering time-series constraints.
- Finally, in Section 5.3, we discuss the notion of *quantitative regular expression*, which is a computational model that is similar to time-series constraints, but is originally motivated by a different context, namely the analysis of data streams.

5.1 Defining Time-Series Constraints

A time-series constraint is imposed on a time series $X = \langle X_1, X_2, \ldots, X_n \rangle$, and an integer variable R, and restricts R to be the result of some computations over X. These computations are characterised by three main parameters, namely two functions, called the *feature* and the *aggregator*, and a regular expression over the alphabet $\{\text{`<'}, \text{`='}, \text{`>'}\}$.

We first recall in Definitions 5.1.1 and 5.1.2 the notions of *occurrence of a regular expression* in the signature of a time series, and in a time series itself, respectively. Then, in Definition 5.1.3, we give the notion of time-series constraint.

With every considered regular expression σ we associate two integer non-negative constants b_σ and a_σ used for trimming the left and the right borders of occurrences of σ. The 'b' of b_σ stands for 'before', while

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

(A)

g	value	$\mathtt{id}_{g,f}$
`sum`	$\sum_{k=1}^{m} f_k$	0
`max`	$\max_{k\in[1,m]} f_k$	$\min_f$
`min`	$\min_{k\in[1,m]} f_k$	$\max_f$

(B)

Table 5.1 – Consider a sequence $X = \langle X_1, X_2, \ldots, X_n\rangle$. (A) features f, their values computed from a subsequence $\langle X_i, X_{i+1}, \ldots, X_j\rangle$, their identity, minimum and maximum values, where n/a stands for not available; (B) aggregators g, their values computed from a sequence of feature values $\langle f_1, f_2, \ldots, f_m\rangle$, and their identity values. These tables are adapted from [22].

the 'a' of a_σ stands for 'after'. Table 5.2 gives examples of regular expressions of time-series constraints together with their parameters b_σ and a_σ.

Definition 5.1.1 (s-occurrence, i-occurrence, e-occurrence [22])**.** Consider a regular expression σ over the alphabet $\{\text{'<'}, \text{'='}, \text{'>'}\}$, the signature $S = \langle S_1, S_2, \ldots, S_{n-1}\rangle$ of some time series, and a subsignature $\langle S_i, S_{i+1}, \ldots, S_j\rangle$ with $1 \leq i \leq j \leq n-1$, forming an inclusion-wise maximal word in S matching σ. The s-occurrence (i, j) of σ is the index sequence $i, i+1, \ldots, j$; the i-occurrence $[i + b_\sigma, j]$ of σ is the index sequence $i + b_\sigma, i + b_\sigma + 1, \ldots, j$; and the e-occurrence $[i + b_\sigma, j + 1 - a_\sigma]$ of σ is the index sequence $i + b_\sigma, i + b_\sigma + 1, \ldots, j + 1 - a_\sigma$.

For a regular expression σ, an s-occurrence indicates the beginning and the end of a maximal occurrence of σ in the signature of a time series. An e-occurrence trims the borders of such maximal occurrence accordingly to the values of the parameters b_σ and a_σ. An i-occurrence is used for defining the *footprint* of σ in a time series, which shows where occurrences of σ are located in this time series. It is important that any i-occurrence is never empty, and two different i-occurrences never overlap otherwise the footprints of two occurrences of σ would overlap and we could not distinguish these occurrences. To guarantee the disjointness of i-occurrences we choose an appropriate value of b_σ.

The notion of e-occurrence (respectively s-occurrence) is used in Definition 5.1.2 for defining the notion of σ-pattern (respectively extended σ-pattern).

Definition 5.1.2 (σ-pattern, extended σ-pattern [8, 14])**.** Consider a regular expression σ over the alphabet $\{\text{'<'}, \text{'='}, \text{'>'}\}$, and a time series $X = \langle X_1, X_2, \ldots, X_n\rangle$. A *$\sigma$-pattern* in X is any subsequence $\langle X_{i+b_\sigma}, X_{i+b_\sigma+1}, \ldots, X_{j+1-a_\sigma}\rangle$ of X such that the index sequence $i + b_\sigma, i + b_\sigma + 1, \ldots, j + 1 - a_\sigma$ is an e-occurrence of σ in the signature of X. An *extended σ-pattern* in X is any subsequence $\langle X_i, X_{i+1}, \ldots, X_{j+1}\rangle$ of X such that the index sequence $i, i+1, \ldots, j$ is an s-occurrence of σ in the signature of X.

Definition 5.1.3 (time-series constraint [22])**.** Consider a sequence of integer variables $X = \langle X_1, X_2, \ldots, X_n\rangle$, and an integer variable R. A *time-series constraint* $g_f_\sigma(X, R)$ is parameterised by $\langle \sigma, f, g\rangle$, where

- σ is one of the 22 regular expressions from Table 5.2.
- f is one of the functions from Part (A) of Table 5.1, called the *feature*.
- g is one of the functions from Part (B) of Table 5.1, called the *aggregator*.

The value of R is constrained to be the result of applying the aggregator g to the list of the values of the feature f from every σ-pattern of X. If there are no σ-patterns in X, then R is constrained to be the identity value of g, defined as $\mathtt{id}_{g,f}$ in Part (B) of Table 5.1.

name σ	regular expression	a_σ	b_σ
BUMP_ON_DECREASING_SEQUENCE	'>><>>'	1	2
DECREASING	'>'	0	0
DECREASING_SEQUENCE	'(> (> | =)*)* >'	0	0
DECREASING_TERRACE	'>=$^+$>'	1	1
DIP_ON_INCREASING_SEQUENCE	'<<><<'	1	2
GORGE	'(> (> | =)*)* >< ((< | =)* <)*'	1	1
INCREASING	'<'	0	0
INCREASING_SEQUENCE	'(< (< | =)*)* <'	0	0
INCREASING_TERRACE	'<=$^+$<'	1	1
INFLEXION	'< (< | =)* > | > (> | =)* <'	1	1
PEAK	'< (< | =)*(> | =)* >'	1	1
PLAIN	'>=*<'	1	1
PLATEAU	'<=*>'	1	1
PROPER_PLAIN	'>=$^+$<'	1	1
PROPER_PLATEAU	'<=$^+$>'	1	1
STEADY	'='	0	0
STEADY_SEQUENCE	'=$^+$'	0	0
STRICTLY_DECREASING_SEQUENCE	'>$^+$'	0	0
STRICTLY_INCREASING_SEQUENCE	'<$^+$'	0	0
SUMMIT	'(< (< | =)*)* <> ((> | =)* >)*'	1	1
VALLEY	'> (> | =)*(< | =)* <'	1	1
ZIGZAG	'(<>)$^+$ < (> |ε) | (><)$^+$ > (< |ε)'	1	1

Table 5.2 – Regular-expression names σ, corresponding regular expressions, and values of the parameters a_σ and b_σ. This table is adapted from [14].

If the feature is `one`, and the aggregator is `sum`, then we name the corresponding time-series constraint as NB_σ instead of SUM_ONE_σ.

Note that Definition 5.1.3 follows [22], where time-series constraints were defined only for the 22 regular expressions in Table 5.2. However, in this thesis we will not limit ourselves to the regular expressions in Table 5.2 but will characterise a class of regular expressions, for which our methods apply.

In the literature, there is a number of works on extracting patterns from time series [123, 121, 136, 1]. However, most of the works focus on detecting peaks or valleys in time series, since these patterns indicate significant changes in the behaviour of a time series. Our approach can handle a large variety of patterns including all patterns of Table 5.2.

As we mentioned in [14], most of the regular expressions in Table 5.2 capture topological patterns that one wants to control when generating time series, while some of them, such as ZIGZAG or BUMP_ON_DECREASING_SEQUENCE, correspond to anomalies one wants to detect in existing time series. The two integer constants b_σ and a_σ are used for trimming the left and right borders of an extended σ-pattern in order to obtain a σ-pattern from which the feature value is computed. This is useful in the case when we need to perform the computations from only a part of occurrence of a regular expression. For example, for INCREASING_TERRACE = '<=$^+$<', since $b_{\text{INCREASING_TERRACE}} = a_{\text{INCREASING_TERRACE}} = 1$, when computing the feature value only the variables involved into an equality contribute to the feature value, i.e. the variables of the flat part of the terrace.

Example 5.1.1 (time-series constraint)**.** Consider the PEAK = '< (< | =)*(> | =)* >' regular expression with the values b_{PEAK} and a_{PEAK} both being 1. As we showed in Example 5.0.2, the signature of the time series $X = \langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$, shown in Figure 5.1, is $S = \langle <, <, =, >, =, <, >\rangle$. There are two maximal occurrences of the PEAK regular expression in S, namely '<<=>' and '<>', corresponding to the

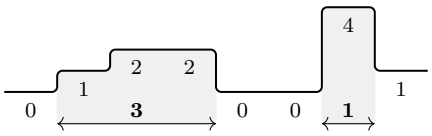


Figure 5.1 – Time series $\langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$ with its two peaks of respective widths 3 and 1

$\langle 1, 2, 2\rangle$ and $\langle 4\rangle$ PEAK-patterns, called *peaks*. Hence, the constraint NB_PEAK$(X, 2)$ holds. The *width* of both peaks is the number of their elements, which is, respectively, 3 and 1. Then, all three constraints MIN_WIDTH_PEAK$(X, 1)$, MAX_WIDTH_PEAK$(X, 3)$, and SUM_WIDTH_PEAK$(X, 4)$ hold. △

If in a time series all variables have fixed values, i.e. all variables are assigned an integer value, then such a time series is called *ground*. We now introduce the notion of *maximal time series*, which we will use when deriving sharp upper bounds on the result value of a time-series constraint in Chapter 7, and also when deriving the AMONG implied constraint in Chapter 8.

Definition 5.1.4 (maximal time series)**.** Consider a time-series constraint $g_f_\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ with every X_i ranging over the same integer interval domain $[\ell, u]$. A ground time series is *maximal* for $g_f_\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ if it contains at least one σ-pattern and yields the maximum value of R among al time series of length n ranging over $[\ell, u]$.

Example 5.1.2 (maximal time series)**.** Consider the NB_DECREASING_SEQUENCE time-series constraint imposed on a time series of length 10 with every time-series variable ranging over the integer interval domain $[0, 4]$. The time series $\langle 3, 2, 4, 2, 4, 1, 3, 2, 3, 0\rangle$, visually presented in Figure 1.2 on page 18, is maximal for NB_DECREASING_SEQUENCE since any other time series of length 10 over $[0, 4]$ has at most 5 decreasing sequences. Note that this time series is not the unique maximal time series for NB_DECREASING_SEQUENCE when n is 10. △

5.2 Operational View of Time-Series Constraints

In order to synthesise a representation of a time-series constraint in the form of a register automaton, the notion of *seed transducer* was introduced in [22]. It was shown in [68] how to generate a seed transducer from a regular expression that belongs to the class of *recognisable patterns*. We recall in Definition 5.2.4 the notion of recognisable pattern [68]. Before that, we recall in Definitions 5.2.1 and 5.2.3 the notions of *regular-expression overlap* and *mismatch overlap*, originally introduced in [68].

Definition 5.2.1 (regular-expression overlap [68])**.** Given a regular expression σ and three words w, x, y such that $xw \in \mathcal{L}_\sigma$, $wy \in \mathcal{L}_\sigma$, and $xwy \notin \mathcal{L}_\sigma$, the length of w is called the *word overlap* of xw and wy. The maximum word overlap between all pairs of words xw and wy in $\mathcal{L}_\sigma$ such that $xwy \notin \mathcal{L}_\sigma$ is called the *σ-overlap* and denoted by o_σ. If for any pair of words $xw \in \mathcal{L}_\sigma$ and $wy \in \mathcal{L}_\sigma$, the word xwy belongs to $\mathcal{L}_\sigma$, then the regular-expression overlap of σ is 0.

Example 5.2.1 (regular-expression overlap)**.** For the $\sigma_1 =$ '$>>^*$' regular expression, the value of o_{σ_1} is 0 since the maximum word overlap is never defined for any pair of words in $\mathcal{L}_{\sigma_1}$. But for the $\sigma_2 =$ '$>=^+>$' regular expression, the value of o_{σ_2} is 1. △

Definition 5.2.2 (prefix language [68])**.** Given a regular expression σ over an alphabet Σ, the *prefix language* of σ is the set of all prefixes of all words in $\mathcal{L}_\sigma$ and is denoted by $\overrightarrow{\sigma}$.

Definition 5.2.3 (mismatch overlap [68])**.** Given a regular expression σ over an alphabet Σ, a word $w \in \overrightarrow{\sigma} \setminus \mathcal{L}_\sigma$, and a symbol $z \in \Sigma$, if $wz \notin \overrightarrow{\sigma}$, then the length of the longest suffix in $\overrightarrow{\sigma}$ of the word wz is called the *mismatch overlap* of w and z. The maximum mismatch overlap of all words in $\overrightarrow{\sigma} \setminus \mathcal{L}_\sigma$ and all symbols in Σ is called the *mismatch overlap* of σ and denoted by μ_σ.

The mismatch overlap of a regular expression σ over Σ reached for a word w in $\overrightarrow{\sigma}$ and a letter $z \in \Sigma$ is equal to one plus the length of the longest suffix v of w such that wz is not a prefix of any word in $\mathcal{L}_\sigma$, but vz is.

Example 5.2.2 (mismatch overlap)**.** For the $\sigma_1 =$ '$>=^+>$' regular expression, the value of μ_{σ_1} is 1 and is reached, for instance, when the word w is '$>$' and the symbol z is '$>$'. For the $\sigma_2 =$ '$<^+=<^+>$' regular expression, the value of μ_{σ_2} is infinite since for any word w in the language of '$<^+=<^+$', which is a subset of $\overrightarrow{\sigma_2}$, and for the symbol $z =$ '$=$', the mismatch overlap is one plus the number of '$<$' in the longest suffix of w that is in $\mathcal{L}_{<^+}$, a subset of the prefix language of σ_2. This value is not bounded. △

Definition 5.2.4 (recognisable pattern [68])**.** Given a regular expression σ over an alphabet Σ and the associated value of the parameter b_σ, the pair $\langle \sigma, b_\sigma \rangle$ is called *a recognisable pattern* if b_σ is at least $\min(\mu_\sigma, \mathrm{o}_\sigma)$.

Note that Definition 5.2.4 does not depend on the parameter a_σ. The intuition behind the notion of recognisable pattern is that b_σ should be large enough so that 1) i-occurrences of σ do not overlap; and 2) computations are not performed on the mismatched part. All regular expressions of Table 5.2 with the associated values of the parameter b_σ are recognisable patterns. We further assume that every considered regular expression with the associated value of b_σ form a recognisable pattern.

We recall the notion of seed transducer in Section 5.2.1, and further in Section 5.2.2 we show how to compile a register automaton for a time-series constraint using the seed transducer for its regular expression.

5.2.1 Seed Transducer for a Regular Expression

Consider a regular expression σ and an integer constant b_σ such that $\langle \sigma, b_\sigma \rangle$ is a recognisable pattern. A *seed transducer* [22] for a regular expression σ is a deterministic finite transducer where each transition is labelled with two letters: a letter in the input alphabet $\Sigma = \{'<', '=', '>'\}$, called the *input symbol*, and a letter in the output alphabet $\Omega = \{\texttt{maybe}_\texttt{b}, \texttt{out}, \texttt{out}_\texttt{r}, \texttt{out}_\texttt{a}, \texttt{found}, \texttt{found}_\texttt{e}, \texttt{in}, \texttt{maybe}_\texttt{a}\}$, called the *output symbol*. Hence, a seed transducer consumes the signature S of a time series 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 phases of recognition of σ and detection of σ-patterns in X. Consider different possibilities of the produced output symbol T_i when consuming an input symbol S_i of S:

- T_i is $\texttt{out}$. A transition labelled by this output symbol implies that the variable X_i does not belong to any σ-pattern. However, X_i may belong to an extended σ-pattern, i.e. X_i plays an important role for the recognition of σ in S, but never contributes to the feature computation.

- T_i is $\texttt{out}_\texttt{r}$. A transition labelled by this output symbol implies that the variable X_i does not belong to any σ-pattern, but there is a suffix of $\langle X_1, X_2, \ldots, X_{i-1} \rangle$ that could have belonged to a σ-pattern if, instead of S_i, a different symbol had been read. As in the case of $\texttt{out}$, X_i may belong to an extended σ-pattern. The 'r' of $\texttt{out}_\texttt{r}$ is a shorthand for 'reset'.

- T_i is $\texttt{out}_\texttt{a}$. A transition labelled by this output symbol indicates the end of the current σ-pattern. The variable X_i belongs to the current σ-pattern iff a_σ is 0, and it may also belong to the next extended σ-pattern. The 'a' of $\texttt{out}_\texttt{a}$ is a shorthand for 'after'.

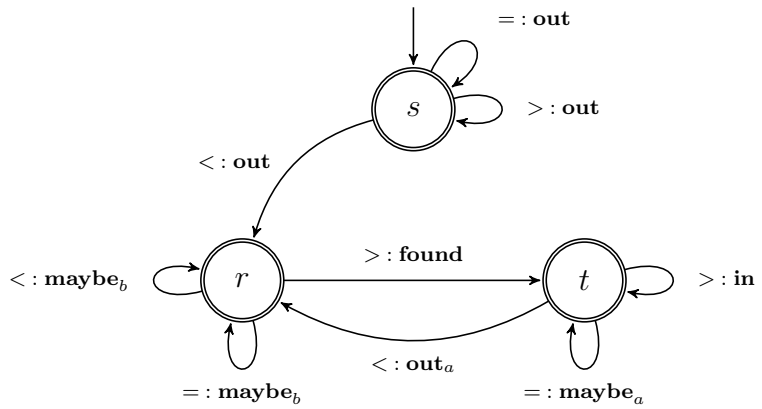


Figure 5.2 – Seed transducer for the PEAK regular expression. This figure is adapted from [10].

- T_i is $\texttt{maybe}_\texttt{b}$. A transition labelled by this output symbol indicates a potential σ-pattern. We are not yet able to determine whether X_i belongs to a σ-pattern or not since it depends on the unseen part of the signature $\langle S_{i+1}, S_{i+2}, \ldots, S_{n-1}\rangle$. The variable X_i belongs to a σ-pattern iff there exist indices $i_1 \in [1, i]$ and $i_2 \in [i+1, n-1]$ such that the word $\langle S_{i_1}, S_{i_1+1}, \ldots, S_{i_2}\rangle$ belongs to the language of σ. Hence the beloning of X_i to a σ-pattern depends on an unseen part of the signature. The 'b' of $\texttt{maybe}_\texttt{b}$ is a shorthand for 'before'.

- T_i is `found`. A transition labelled by this output symbol corresponds to the discovery of a new σ-pattern in X that may potentially be extended. The variable X_i belongs to the current σ-pattern.

- T_i is $\texttt{found}_\texttt{e}$. A transition labelled by this output symbol corresponds to the discovery of a new σ-pattern in X that cannot be extended further. The variable X_i belongs to the current σ-pattern. The 'e' of $\texttt{found}_\texttt{e}$ is a shorthand for 'end'.

- T_i is `in`. A transition labelled by this output symbol corresponds to an extension of the current σ-pattern. The variable X_i belongs to the current σ-pattern.

- T_i is $\texttt{maybe}_\texttt{a}$. A transition labelled by this output symbol corresponds to a potential extension of the current σ-pattern. As in the case of $\texttt{maybe}_\texttt{b}$, the fact that X_i belongs or not to a σ-pattern depends on the unseen part of the signature $\langle S_{i+1}, S_{i+2}, \ldots, S_{n-1}\rangle$. The variable X_i belongs to a σ-pattern iff there exist indices $i_1 \in [1, i-1]$ and $i_2 \in [i+1, n-1]$ such that the word $\langle S_{i_1}, S_{i_1+1}, \ldots, S_{i_2}\rangle$ belongs to the language of σ. The 'a' of $\texttt{maybe}_\texttt{a}$ is a shorthand for 'after'.

The following definition formalises the notion of occurrence of a regular expression in the output sequence of a seed transducer.

Definition 5.2.5 (t-occurrence [22])**.** Given a seed transducer $\mathcal{S}$ and the signature S of some time series, the *t-occurrence* of $\mathcal{S}$ for S consists of the indices of the phase letters of a maximal word within the transduction of S that matches one of the regular expressions '$\texttt{maybe}_\texttt{b}{}^*\texttt{found}_\texttt{e}$' or '$\texttt{maybe}_\texttt{b}{}^*\texttt{found}(\texttt{maybe}_\texttt{a}{}^*\texttt{in}^+)^*$'.

Example 5.2.3 (seed transducer for a regular expression)**.** Consider the PEAK regular expression introduced in Example 5.1.1. The seed transducer $\mathcal{T}$ for PEAK is given in Figure 5.2. While consuming the signature $S = \langle <, <, =, >, =, <, >\rangle$, $\mathcal{T}$ produces $T = \langle \texttt{out}, \texttt{maybe}_\texttt{b}, \texttt{maybe}_\texttt{b}, \texttt{found}, \texttt{maybe}_\texttt{a}, \texttt{out}_\texttt{a}, \texttt{found}\rangle$. As shown in Example 5.1.1, S contains two maximal occurrences of PEAK, complying with the two t-occurrences of PEAK in T. △

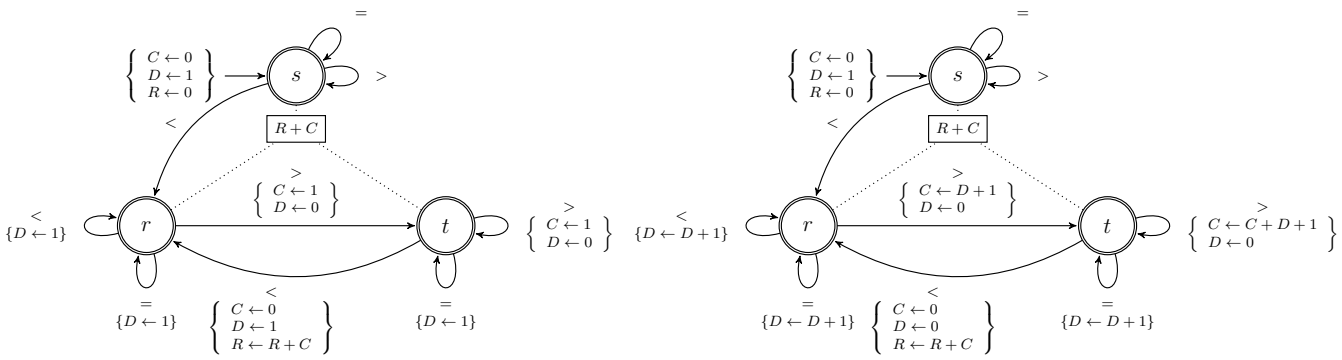


Figure 5.3 – Register automata for NB_PEAK and SUM_WIDTH_PEAK. These figures are adapted from [10].

5.2.2 Synthesising and Simplifying Register Automata

Synthesis of register automata. Consider a time-series constraint g_f_σ. It was shown in [22] that three registers, named C, D and R, are enough to represent any g_f_σ by a register automaton. The meaning of these three registers is the following:

- The register C contains the feature value of the current σ-pattern read so far. Hence, this register contains information about *the present*.
- The register D contains the feature value of the part that *may* belong to the current σ-pattern; it will be validated or invalidated later on. Hence, this register contains information about the *future*.
- The register R contains the aggregated value of the features of σ-patterns completely finished so far. Hence, this register contains information about the *past*.

In order to obtain a register automaton with registers C, D, and R for g_f_σ, [22] uses two objects 1) the seed transducer for σ, recalled in Section 5.2.1; and 2) the *decoration table*, which associates with every phase letter of the seed transducer a set of register updates. Hence, a register automaton for g_f_σ is obtained from the seed transducer for σ by replacing every phase letter with register updates wrt the decoration table, which also defines the initial values of the registers and the acceptance function. Table in Part (B) of Figure 5.5 is the decoration table used for any constraint g_f_σ such that a_σ is 1.

Example 5.2.4 (synthesised register automata)**.** Consider the PEAK regular expression, introduced in Example 5.1.1. The seed transducer for PEAK is given in Figure 5.2. The register automata obtained from this seed transducer for the NB_PEAK and SUM_WIDTH_PEAK, using the decoration table in Part (B) of Figure 5.5, are given in Figure 5.3. △

Simplifications of register automata. One can notice that sometimes having three registers in a register automaton is redundant. For example, a register automaton counting the number of peaks in a time series needs a single register, which is incremented every time when the transition labelled with `found` is triggered. For some other regular expressions, their seed transducers do not have transitions labelled with $\texttt{maybe}_\texttt{b}$ and $\texttt{maybe}_\texttt{a}$, and thus the register D is never updated and can be removed.

In order to simplify register automata we specialised decoration tables in [11], i.e. instead of one general decoration given in Part (B) of Figure 5.5, we created 12 decoration tables. Each of the obtained tables applies for a subset of time-series constraints. We do not give details about how to design specialised decoration tables here, but only list a few properties of the triples $\langle \sigma, f, g \rangle$ that allow us to obtain more specific decoration tables:

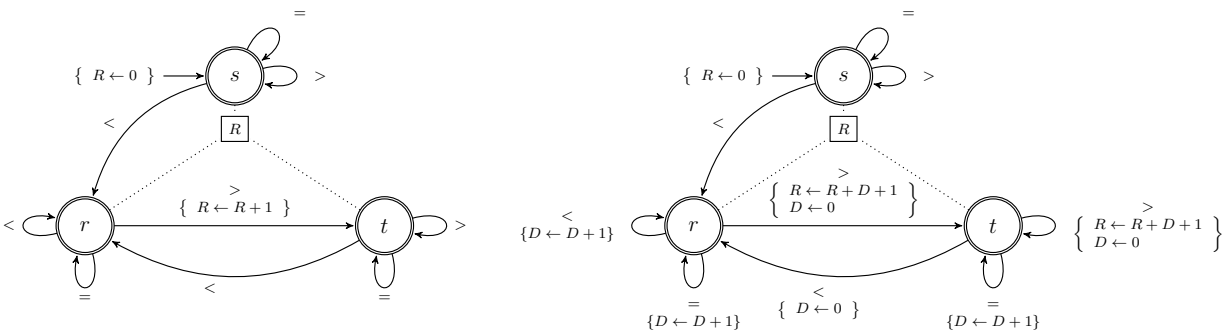


Figure 5.4 – Simplified register automata for NB_PEAK (left) and SUM_WIDTH_PEAK (right). These figures are adapted from [10].

1. Consider a time series $X = \langle X_1, X_2, \ldots, X_n \rangle$ such that the seed transducer for σ produced the output sequence $\langle T_1, T_2, \ldots, T_{n-1} \rangle$ after having consumed the signature of X. The property is the following: for any σ-pattern $X_{i,j} = \langle X_i, X_{i+1}, \ldots, X_j \rangle$ of X, the value of $f(X_{i,j})$ is equal to δ_f^k, where δ_f^k is defined by the table in Part (A) of Figure 5.5, and $k \in [i, j]$ is the index such that the produced output symbol T_k is either `found` or $\texttt{found}_\texttt{e}$. This property allows us to use a single register R, which is updated as $g(R, \delta_f^k)$ every time a transition labelled with `found` or $\texttt{found}_\texttt{e}$ is triggered.
   Examples of time-series constraints having this property are all NB_σ constraints, all g_MAX_PEAK constraints, and all g_MIN_VALLEY constraints.

2. The seed transducer for σ does not have any transition labelled with $\texttt{maybe}_\texttt{b}$ or $\texttt{maybe}_\texttt{a}$. This property allows us to remove the register D since it is never updated.
   Examples of time-series constraints having this property are all g_f_STEADY_SEQUENCE constraints, and all g_f_DECREASING constraints.

3. The seed transducer for σ does not have any transition labelled with `found`, `in` or $\texttt{maybe}_\texttt{a}$. This property allows us to remove the register C since it is never updated.
   Examples of time-series constraints having this property are all g_f_DECREASING_TERRACE constraints, and all g_f_INFLEXION constraints.

4. For any time series $X = \langle X_1, X_2, \ldots, X_n \rangle$, and for any σ-pattern $X_{i,j} = \langle X_i, X_{i+1}, \ldots, X_j \rangle$ of this time series, and for an arbitrary sequence of integers $k_1, k_2, \ldots, k_p$ such that $i \leq k_1 < k_2 < \cdots < k_p \leq j$, we have $g\left(f\left(X_{i,j}\right)\right) = g\left(f\left(\langle X_i, X_{i+1}, \ldots, X_{k_1} \rangle\right), f\left(\langle X_{k_1+1}, X_{k_1+2}, \ldots, X_{k_2} \rangle\right), \ldots, f\left(\left\langle X_{k_{p-1}+1}, X_{k_{p-1}+2}, \ldots, X_j \right\rangle\right)\right)$. This property does not allow us to remove registers, but it allows to perform the aggregation without waiting for the end of the σ-pattern, i.e. waiting for the $\texttt{out}_\texttt{a}$ symbol. This is important in the context of CP since it allows to propagate earlier.
   Examples of time-series constraints having this property are all SUM_WIDTH_σ constraints and all SUM_SURF_σ constraints.

Example 5.2.5 (Simplified register automata)**.** The NB_PEAK (respectively SUM_WIDTH_PEAK) time-series constraint satisfies Property 1 (respectively Property 4) and thus the corresponding register automaton can be simplified. The simplified register automata for the NB_PEAK and the SUM_WIDTH_PEAK time-series constraints are given in Figure 5.4. △

5.2.3 Glue Constraints

In this section, we recall *glue constraints* [23, 8], which are an important element of the propagation of time-series constraints. We will use glue constraints when evaluating the impact of our contributions in Part III.

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

(A)

initial values **acceptance function**	$R \leftarrow \mathtt{id}_{g,f}$ $g(R, C)$	$C \leftarrow \mathtt{id}_{g,f}$	$D \leftarrow \mathtt{id}_f$
phase letters	**register updates** *update of R*	*update of C*	*update of D*
`out`			
$\mathtt{out}_\mathtt{r}$			$D \leftarrow \mathtt{id}_f$
$\mathtt{out}_\mathtt{a}$	$R \leftarrow g(R, C)$	$C \leftarrow \mathtt{id}_{g,f}$	$D \leftarrow \mathtt{id}_f$
$\mathtt{maybe}_\mathtt{b}$			$D \leftarrow \phi_f\left(D, \delta_f^i\right)$
$\mathtt{maybe}_\mathtt{a}$			$D \leftarrow \phi_f\left(D, \delta_f^i\right)$
`found`		$C \leftarrow \phi_f\left(D, \delta_f^i\right)$	$D \leftarrow \mathtt{id}_f$
$\mathtt{found}_\mathtt{e}$	$R \leftarrow g\left(R, \phi_f\left(D, \delta_f^i\right)\right)$		$D \leftarrow \mathtt{id}_f$
`in`		$C \leftarrow \phi_f\left(C, \phi_f\left(D, \delta_f^i\right)\right)$	$D \leftarrow \mathtt{id}_f$

(B)

Figure 5.5 – (A) Features: the function used for computing the feature value of the feature f. (B) Decoration table used for synthesising the register automaton for a g_f_σ time-series constraint from the seed transducer for σ, a feature f, and an aggregator g when the value of the parameter a_σ is 1. This table is adapted from [22].

For a given time-series constraint imposed on a time series X, and an index i of a time-series variable in X, a glue constraint links the result value from X with the result values from the prefix of X ending at the variable X_i, and the suffix of X starting at the variable X_i. Before recalling the concept of glue constraint in Definition 5.2.8, we recall the notion of *reverse of a signature* in Definition 5.2.6, and the notion of *reverse of a register automaton* in Definition 5.2.7.

Definition 5.2.6 (reverse of a signature [8])**.** Consider the signature $S = \langle S_1, S_2, \ldots, S_{n-1}\rangle$ of some time series. The reverse of S is the sequence $\left\langle S'_1, S'_2, \ldots, S'_{n-1}\right\rangle$, where every S'_i (with i in $[1, n-1]$) is defined as

$$S'_i = \begin{cases} \text{`}<\text{'} & \text{if } S_{n-i} = \text{`}>\text{'}, \\ \text{`}=\text{'} & \text{if } S_{n-i} = \text{`}=\text{'}, \\ \text{`}>\text{'} & \text{if } S_{n-i} = \text{`}<\text{'}. \end{cases}$$

Example 5.2.6 (reverse of a signature)**.** The signature $\langle =, >, =, <, <, =, >, <, =\rangle$ is the reverse of the $\langle =, >, <, =, >, >, =, <, =\rangle$ signature and vice versa. △

Definition 5.2.7 (reverse of a register automaton [8])**.** Consider a register automaton $\mathcal{M}$ whose input alphabet is $\langle <, =, >\rangle$. The reverse of $\mathcal{M}$ is a register automaton $\mathcal{M}'$ such that, for any input signature S recognised by $\mathcal{M}$, upon consuming S, $\mathcal{M}$ returns the same value as $\mathcal{M}'$ upon consuming the reverse of S.

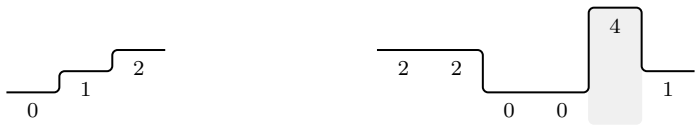
Example 5.2.7 (reverse of a register automaton)**.** Both automata in Figure 5.4 are their own reverse. △

For any time-series constraint g_f_σ such that σ is one of the regular expression in Table 5.2 except BUMP_ON_DECREASING_SEQUENCE, DIP_ON_INCREASING_SEQUENCE, and INFLEXION, the reverse of its register automaton is the register automaton for a time-series constraint g_f_σ', where σ' is a regular expression in Table 5.2.

Definition 5.2.8 (glue constraint [8])**.** Consider a time-series constraint $g_f_\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$, its register automaton $\mathcal{M}$, the reverse $\mathcal{M}'$ of $\mathcal{M}$, and an index i in $[1, n]$. Let $\overrightarrow{R}_i$ (respectively $\overleftarrow{R}_i$) be constrained by $g_f_\sigma(\langle X_1, X_2, \ldots, X_i\rangle, \overrightarrow{R}_i)$ (respectively $g_f_\sigma(\langle X_n, X_{n-1}, \ldots, X_i\rangle, \overleftarrow{R}_i)$), and $\overrightarrow{Q}_i$ (respectively $\overleftarrow{Q}_i$) be the final state of $\mathcal{M}$ (respectively of $\mathcal{M}'$) after having consumed the signature of $\langle X_1, X_2, \ldots, X_i\rangle$ (respectively of $\langle X_n, X_{n-1}, \ldots, X_i\rangle$). Then, the constraint $R = g(\overleftarrow{R}_i, \overrightarrow{R}_i, \Delta(\overrightarrow{Q}_i, \overleftarrow{Q}_i))$ is called a *glue constraint*, where the function $\Delta(\overrightarrow{Q}_i, \overleftarrow{Q}_i)$ is called the *glue expression*, and depends on $\mathcal{M}$ and $\mathcal{M}'$.

	s	r	t
s	0	0	0
r	0	1	1
t	0	1	0

Table 5.3 – Glue matrix for the NB_PEAK constraint



(A) (B)

Figure 5.6 – (A) The prefix $\langle 0, 1, 2\rangle$ of the time series $\langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$ without any peaks. (B) The suffix $\langle 2, 2, 0, 0, 4, 1\rangle$ of the time series $\langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$ with one peak, highlighted in grey.

It was shown in [8] that for a given g_f_σ time-series constraint, the glue expression can be compositionally obtained from the seed transducer for σ and the decoration table. We give the glue expression in a form of matrix, called a *glue matrix*, where every row (respectively column) corresponds to some state q_1 (respectively q_2) of $\mathcal{M}$ (respectively $\mathcal{M}'$), and the cell on the intersection of the row for q_1 and the column for q_2 gives the glue expression $\Delta(q_1, q_2)$ for q_1 and q_2.

Example 5.2.8 (glue matrix)**.** Consider the NB_PEAK time-series constraint and its simplified register automaton from Figure 5.4, which is its own reverse. The corresponding glue matrix is given in Table 5.3. Figure 5.6 gives the prefix P of length 3 and the suffix S of length 6 of the time series $T = \langle 0, 1, 2, 2, 0, 0, 4, 1\rangle$. After consuming the signature of P (respectively reverse of S) by the automaton in Figure 5.4, it returns 0 (respectively 1) since P does not have any peaks (respectively S has one peak), and it finishes in the state r. Then, by the glue constraint the number of peaks in T is the sum of three quantities, namely the number of peaks in P, the number of peaks in the reverse of S, and the glue expression from the (r,r) cell of Table 5.3. Hence, T has two peaks, which complies with Example 5.1.1. △

5.3 Related Approach: Quantitative Regular Expressions

In this section, we discuss *quantitative regular expressions (QREs)*, which have the same logic as time-series constraints, i.e. multi-level computations from occurrences of some pattern in an integer sequence, but which are used in a completely different context.

Quantitative regular expressions (QREs) were introduced in [5] and are mainly used for data stream analysis [5, 6, 15, 93]. Often, in the context of data stream analysis one needs to map parts of streams into numerical values that are further used for real-time decision-making. QREs provide a declarative language for describing queries over data streams [5].

A QRE is parameterised by a regular expression σ over an alphabet given by a considered application, and a set of functions $f_1, f_2, \ldots, f_t$. The result of a QRE is a numerical value computed by applying functions $f_1, f_2, \ldots, f_t$ in a cascading way to occurrences of σ in an input stream of data. The baseline implementation of QREs is a streaming algorithm [5]. While time-series constraints use only three level for computing a given result, this provides a framework of restricted family of constraints for which the combinatorial aspect can be studied in a systematic way, as we will show in the next part of this thesis. Getting similar results for QREs would be much more challenging.

criterion	time-series constraints	QREs
signature	binary	unary
implementation	automata with at most 3 registers	streaming algorithms
number of levels	3 (regular expression, feature, aggregator)	unlimited
typical applications	extracting time-series constraints, generating time series satisfying a conjunction of time-series constraints	analysis of data streams
need to handle combinatorics	yes	no

Table 5.4 – Comparison of time-series constraints and QREs

Table 5.4 shows a few differences between time-series constraints and QREs. The main difference is in applications: time-series constraints can be used for modelling and then solving a problem, hence there is a need to handle the combinatorial aspect of time-series constraints, and not the one of QREs, which focus on stream analysis.

Part II

Theoretical Contributions

In this part, we present our theoretical contributions related to synthesis of combinatorial objects for time-series constraints and to the extension of the transducer-based model for describing sequence constraints.

First, Chapter 6 gives a detailed overview of our contributions and also states the key ideas of our methods for synthesising combinatorial objects for time-series constraints.

Second, Chapters 7, 8, 9, 10 and 11 present our methods for synthesising combinatorial objects for time-series constraints, which are defined in a compositional way by multiple layers of cascading functions: regular expression occurrences, feature and aggregator, and have a baseline implementation by register automata. Despite a rich and powerful modelling language provided by time-series constraints, it is unknown, in general, how to efficiently maintain domain consistency for a conjunction of time-series constraints, which makes it unpredictable how these constraints potentially interact, and the baseline propagation provided by the decomposition of a register automaton [20] is weak. Hence there is a need for adding implied constraints that do not change the solution space but allow us to find a solution faster. We consider two cases of using time-series constraints:

1. For a **time-series constraint in isolation**, we synthesise sharp bounds on its result value, presented in Chapter 7, and also AMONG implied constraints, presented in Chapter 8. Both methods use the regular-expression representation of time-series constraints. We now give two examples:

   1. A time series of length 10 cannot have 8 peaks, but the solver does not realise that this value is infeasible, and typically spends a large amount of time trying to prove the infeasibility. By imposing a sharp upper bound on the result value of a time-series constraint we reduce the effort to find a solution or to prove infeasibility.

   2. When the surface of a peak is close to its maximum possible value among all peaks of the same width and over the same domain, then the values in this peak should also be large. By imposing a lower bound on the number of variables in a peak that should take large values, we reduce the effort to find a solution or to prove infeasibility.

2. For a **conjunction of time-series constraints** imposed on the same sequence, we synthesise linear, Chapter 9, and non-linear invariants, Chapters 10 and 11, linking the result values of the constraints in the conjunction and parameterised by the sequence length. Both methods use register automata and/or seed transducers. We now give two examples:

   1. No time series can simultaneously have 5 peaks and 1 valley since peaks and valleys alternate and between any two peaks there is always exactly one valley. Very often the solver cannot capture such implicit relations between constraints, which leads to a weak filtering. Imposing a linear invariant stating that the number of peaks in a time series is less than or equal to the number valleys plus one would eliminate the infeasible combination of 5 peaks and 1 valley and reduce effort for finding a solution or proving infeasibility.

   2. When infeasible combinations of the result values are located within the convex hull of feasible values linear invariants cannot eliminate them. Hence, there is a need for non-linear invariants eliminating such combinations.

Third, Chapter 12 presents an extended transducer-based model that does not have a mix of qualitative and quantitative aspects as it was the case in [22] and allows us to describe a larger class of sequence constraints.

Chapter 6

Overview of our Theoretical Contributions

Before presenting our theoretical contributions, we give a synthetic overview of these contributions.

1. In Section 6.1, we give an overview of our contributions for synthesising combinatorial objects for a single time-series constraint, and discuss the main idea of these contributions, namely using the *declarative view* of time-series constraints, i.e. regular expressions and *regular-expression characteristics*.
2. In Section 6.2, we give an overview of our contributions for synthesising combinatorial objects for conjunctions of time-series constraints, and discuss the main idea of these contributions, namely using the *operational view* of time-series constraints, i.e. register automata and transducers.
3. In Section 6.3, we demonstrate that the synthesised combinatorial objects were integrated into the Volume II of the Global Constraint Catalogue [10].
4. In Section 6.4, we discuss the extended transducer-based model, which allows to capture more sequence constraints using transducers, and highlight the main differences with the initial model of [22].

6.1 Overview of our Contribution for Time-Series Constraints in Isolation

Consider a time-series constraint $g_f_\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$. To characterise this constraint we developed methods for synthesising the following combinatorial objects:

- Generic formulae for sharp bounds on the result variable R, provided every time-series variable X_i is ranging over the same integer interval domain, and the regular expression σ satisfies certain properties. When the domains of X_i are not uniform and/or are not intervals, the bounds are still valid, but their sharpness is no longer guaranteed. For a given pair $\langle f, g\rangle$ of feature and aggregator, we have a small number of formulae, usually a single one, that is parameterised by a regular expression σ. This allows us to prove a very few formulae that apply for all time-series constraints whose feature is f and aggregator is g, which supports the idea of the *compositionality* of the obtained combinatorial object.
  In Chapter 7, we systematically derive and prove formulae for the cases when the feature f is either `one` or `width`.

- Generic AMONG implied constraint for time-series constraints whose feature is `surf`. We impose the conjunction $\text{AMONG}(N, \langle X_1, X_2, \ldots, X_n\rangle, L) \ \wedge \ N \geq B$, where L is a list of integer values depending on the regular expression σ and the aggregator g, and B is the result of evaluation of a function also depending on σ, g, and R. For each aggregator, we have one formula for B, which is parameterised by the regular expression σ. The main idea of these implied constraints is that, when the value of R is close to its maximum value, there should be large enough values in

$\langle X_1, X_2, \ldots, X_n \rangle$. The crucial element of this part is the bound on the result value R of Chapter 7, and the structure of time series reaching this bound.
In Chapter 8, we derive and prove three generic formulae, one for each aggregator in $\{\texttt{max}, \texttt{min}, \texttt{sum}\}$.

6.1.1 First Key Idea: Regular-Expression Characteristics

When we say that a formula is parameterised by a regular expression σ, it means that it is parameterised by some quantities computed from σ, called *regular-expression characteristics*. For example, the length of a shortest word in the language of σ is one characteristic of a regular expression, and we will actively use it when deriving the bounds on the result value of time-series constraints.

Regular-expression characteristics are similar to graph characteristics, e.g.

- diameter, the largest of the smallest distances between any two vertices of a graph;
- clique number, the number of vertices in a subgraph of a graph, where every two different vertices are adjacent;
- number of connected components, the number of subgraphs of a graph where there is a path between any pair of vertices.

Graph characteristics have been already used in constraint programming for obtaining parameterised formula [31]. However, for regular expressions, to the best of our knowledge no such work has been done before, and our work for deriving formulae for sharp bounds and AMONG implied constraints opens new perspectives that will be described in Chapters 7 and 8.

Consider a $g_f_\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[\ell, u]$. Currently, there are three use cases of regular-expression characteristics:

1. Necessary and sufficient condition for having at least one σ-pattern in $\langle X_1, X_2, \ldots, X_n \rangle$, given in Section 7.1.9. We call it, for short, necessary and sufficient condition. This condition restricts the domain $[\ell, u]$, i.e. the domain should be large enough, and the sequence length n, i.e. the sequence should be long enough.
2. Sharp bounds on R, given in Sections 7.2 and 7.3.
3. AMONG implied constraint for g_f_σ, given in Section 8.2.

We now give a list of regular-expression characteristics introduced in Chapters 7 and 8 together with their intuitions and uses cases:

- The *size* of σ, denoted by ω_σ. It is the length of a shortest word in the language of σ. It is used both for synthesising bounds and in the necessary and sufficient condition.

- The *height* of σ, denoted by η_σ. It is the minimum difference between the maximum and the minimum values in an extended σ-pattern. It is used for synthesising bounds, in the necessary and sufficient condition, and also in the AMONG implied constraints for defining the list of parameters of AMONG.

- The *set of inducing words* of σ, denoted by Θ_σ. It is a subset of $\mathcal{L}_\sigma$ such that for every word v in $\mathcal{L}_\sigma$, there exists a word $w = w_1 w_2 \ldots w_k$ in Θ_σ such that every $w_i \neq \varepsilon$ and every $v \in \mathcal{L}_\sigma$ can be represented as $v_1 w_1 v_2 w_2 \ldots v_k w_k v_{k+1}$ with every v_i being a word in $\{\text{`}<\text{'}, \text{`}=\text{'}, \text{`}>\text{'}\}^*$. It is used for synthesising lower bounds.

- The *range* of σ wrt $\langle n \rangle$, denoted by $\phi_\sigma(n)$. It is the minimum difference between the maximum and the minimum values in an extended σ-pattern of width n. It is used for synthesising bounds.

- The *overlap* of σ wrt $\langle \ell, u \rangle$, denoted by $o_\sigma^{\langle \ell, u \rangle}$. It is the maximum number of common time-series variables of two consecutive extended σ-patterns. It is used for synthesising upper bounds, and also in AMONG implied constraints for deriving a lower bound on the value of the parameter N of AMONG.

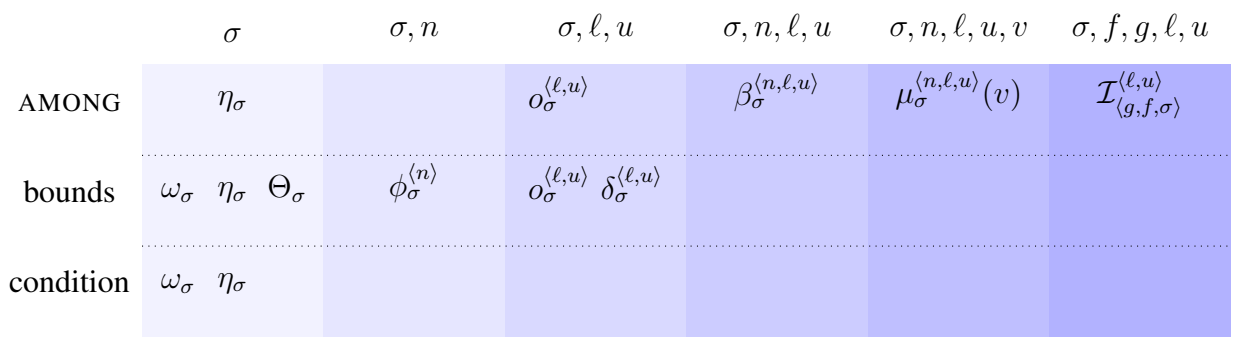


Figure 6.1 – Regular-expression characteristics introduced in Chapters 7 and 8. The horizontal (respectively vertical) axis is for arguments (respectively use cases) of the characteristics. Here, σ is a regular expression, f is a feature, g is an aggregator, ℓ and u are, respectively, the domain minimum and maximum values, v is an integer number, n is a time-series length, AMONG stands for an AMONG implied constraint, "bounds" stands for sharp bounds on the result value of a time-series constraint, and "condition" stands for the necessary and sufficient condition. The colour intensity designates the argument-wise complexity of a characteristic: the darker is the colour the more complex the characteristic.

- The *smallest variation of maxima* of σ wrt $\langle \ell, u \rangle$, denoted by $\delta_\sigma^{\langle \ell,u \rangle}$. It is the smallest difference between the maximum values of two consecutive extended σ-patterns that have at least one common time-series variable. It is used for synthesising upper bounds.

- The *big width* of σ wrt $\langle n, \ell, u \rangle$, denoted by $\beta_\sigma^{\langle \ell,u \rangle}$. It is the largest width of a σ-pattern in a time series $\langle X_1, X_2, \ldots, X_n \rangle$. It is used in AMONG implied constraints for deriving a lower bound on the value of the parameter N of AMONG.

- The *maximum value number of an integer value* v in a σ-pattern wrt $\langle n, \ell, u \rangle$, denoted by $\mu_\sigma^{\langle n,\ell,u \rangle}(v)$. It is the largest number of occurrences of v in a σ-pattern of $\langle X_1, X_2, \ldots, X_n \rangle$. It is used in the AMONG implied constraints for deriving a lower bound on the value of the parameter N of AMONG.

- The *interval of interest* of $\langle g, f, \sigma \rangle$ wrt $\langle \ell, u \rangle$, denoted by $\mathcal{I}_{\langle g,f,\sigma \rangle}^{\langle \ell,u \rangle}$. It is the interval of values that often appears in σ-patterns of the maximal time series for g_f_σ. The values of this interval correspond to the list of value parameters passed to AMONG.

Figure 6.1 summarises the presented information about regular-expression characteristics in a synthetic way.

6.2 Overview of our Contribution for a Conjunction of Time-Series Constraints

Consider a $\gamma_1(X, R_1) \wedge \gamma_2(X, R_2) \wedge \cdots \wedge \gamma_k(X, R_k)$ conjunction of time-series constraints with every γ_i being either NB_σ_i or SUM_WIDTH_σ_i. We focus on these families of time-series constraints since every result value R_i depends only on the signature of X. For example, for every γ_i, both time series $\langle 1, 2, 3 \rangle$ and $\langle -1, 6, 8 \rangle$ yield the same value of R_i since they have the same signature $\langle <, < \rangle$. We give an overview of our two contributions for characterising the relations between the result variables of time-series constraints in a conjunction:

- Linear inequalities of the form $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i \geq 0$, where $e, e_0, \ldots, e_k$ are integer numbers that we extract from the intersection of register automata for $\gamma_1, \gamma_2, \ldots, \gamma_k$. Such invariants capture implicit relations between the result values of different time-series constraints in a considered

conjunction, and are valid for any instance, i.e. independently of n and from the domains of the time-series variables. In addition, for conjunctions of two different time-series constraints and for a given sequence length, we observe that in many cases the extracted linear inequalities are facets of the convex hull of feasible combinations $(R_1, R_2, \ldots, R_k)$, i.e. they are sharp.
Chapter 9 describes our method for extracting linear invariants from the intersection of register automata for time-series constraints in a considered conjunction. Although, we initially did this work for time-series constraints, the method applies for any sequence constraint whose register automaton have the *incremental-automaton property*, described in Property 9.1.1.

- Non-linear invariants parameterised by a function of n, and that *characterise infeasible combinations* of $(R_1, R_2, \ldots, R_k)$ within the convex hull of feasible combinations. Even having all facet-defining inequalities may not be enough to achieve a good propagation for a conjunction of time-series constraints. One of the reasons is the presence of infeasible points within the convex hull of feasible values of $(R_1, R_2, \ldots, R_k)$. The values corresponding to these points cannot be removed from the domains of $R_1, R_2, \ldots, R_k$ by just using linear inequalities.
  In order to synthesise invariants characterising infeasible points within the convex hull we use the following three-phase method:

  1. generate a dataset from which we will extract non-linear invariants in the next phase;
  2. use a data mining technique to extract hypotheses describing sets of infeasible points;
  3. *automatically prove or invalidate* the extracted hypotheses. Proofs are independent of the sequence length and from the domains of the time-series variables.

  The crucial component of the proof phase is the automatic generation of automata without registers, called *conditional automata*, representing a set of signatures satisfying some condition, e.g. all signatures with the maximum number of occurrences of the PEAK regular expression. Although, we use conditional automata only for proving non-linear invariants, this piece of work may be interesting on its own, since to the best of our knowledge, there were no works on representing the set of solutions achieving some bound by a constant-size automaton.
  Chapter 10 describes our method for extracting non-linear invariants for time-series constraints in a considered conjunction. Chapter 11 explains how to automatically generate conditional automata that are required for the proof phase from the transducer for the corresponding regular expression or from the associated register automaton.

6.2.1 Second Key Idea: Operational View of Time-Series Constraints

For a conjunction of two time-series constraints $g_1_f_1_\sigma_1(X, R_1)$ and $g_2_f_2_\sigma_2(X, R_2)$ such that σ_1 and σ_2 are different, the relations between occurrences of σ_1 and σ_2 in the signature of X can be complicated, for example,

- an occurrence of σ_1 is always included in an occurrence of σ_2,
- the overlap between an occurrence of σ_1 and an occurrence of σ_2 is not bounded by any constant,
- there is always one occurrence of σ_1 between any two occurrences of σ_2.

Hence the approach for synthesising invariants linking R_1, R_2 and n using regular-expression characteristics would force us to analyse the relations between 242 pairs of regular expressions. Moreover, the more constraints in a considered conjunction, the harder it becomes.

Rather than manually analysing the interaction of each possible pair of regular expressions we use the following facts:

1. it is possible to automatically generate a seed transducer for a regular expression [68],
2. it is possible to synthesise a register automaton for a time-series constraint g_f_σ from the seed transducer for σ [22],

3. it allows to use the compositional aspect of register automata, i.e. to intersect them and extract from this intersection some invariants capturing the interaction of a conjunction of constraints.

6.3 Integrating Combinatorial Objects into the Global Constraint Catalogue

As mentioned in the introduction, our synthesised combinatorial objects were integrated into the Volume II of the Global Constraint Catalogue. In the catalogue, each time-series constraint has an entry, consisting of the following slots:

- origin of the constraint and the related regular expression;
- the signature of the constraint;
- the arguments of the constraint;
- restrictions, which include *sharp bounds on the result value of the constraint*, presented in Chapter 7, and the AMONG *implied constraints*, presented in Chapter 8;
- the purpose of the constraint;
- an example with a figure when the constraint holds;
- typical restrictions on the number of variables in a time series, and the domains of time-series variables;
- symmetries, if any;
- argument properties, e.g. the result value of the constraint is functionally determined by a time series;
- the register automata (original and simplified) for the constraint;
- the glue matrices for the constraint;
- and *conditional automata* generated by the methods of Chapter 11.

For each catalogue entry, we have metadata in the form of Prolog file; from this metadata we generate the LaTeX code, from which the PDF file for the catalogue is compiled. Appendix A.1 gives the metadata for the NB_PEAK time-series constraint, and Appendix A.2 gives the pages of the PDF file generated from these metadata.

Invariants for a conjunction of time-series constraints presented in Chapters 9 and 10 were integrated into the database of invariants of the catalogue, which contains 917 entries: 176 entries corresponding to a single constraint and 741 entries corresponding to pairs of constraints. The total number of invariants is 2422: 763 linear invariants, 974 conditional linear invariants, 662 non-linear invariants, and 23 manually derived invariants. The total number of conditional automata generated for proving invariants is 2430. Appendix B.1 gives a piece of metadata for some invariants, and Appendix B.2 gives the pages of the PDF file generated from these metadata.

6.4 Overview of the Extended Transducer-Based Model

The initial model of [22] used transducers for regular expressions to synthesise register automata for time-series constraints that can be used as propagators. The two main drawbacks of this model are

- A mix of qualitative and quantitative aspects. Seed transducers representing the qualitative aspect of constraints are dependent on the values of the purely quantitative parameter b_σ, used for trimming the left extremity of occurrences of a regular expression (see Definition 5.1.1). Hence changing the value of b_σ would lead to a new transducer. For example, for the $\sigma =$ DECREASING_TERRACE regular expression, Figure 6.2 gives two seed transducers for the cases when b_σ is 1 and 2, respectively. They differ by the output symbol of the transition from r to t.
  The extended transducer-based model presented in Chapter 12 does no longer have this problem. A single seed transducer can be used for an interval of values of b_σ.

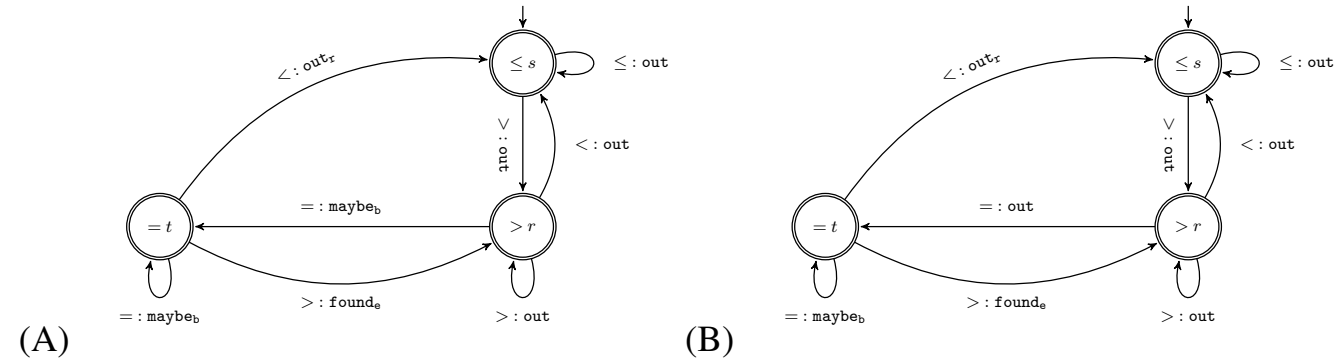


Figure 6.2 – Seed transducer for $\sigma =$ DECREASING_TERRACE when b_σ is 1 (A) and 2 (B)

- The model of [22] is dedicated to time-series constraints. However, a number of existing global constraints such as AMONG [25], and STRETCH [108] could be represented using the same approach. The extended transducer-based model allows any signature of any arity, and is no longer limited to three letters.

- Due to the new phase letter $\mathbf{maybe}_\text{r}$ the extended transducer-based model can also handle cases where we need to resynchronise the computation of a feature. For example, after triggering a sequence of transitions corresponding to $\texttt{maybe}_\texttt{b}$ we need to 'forget' a computed part corresponding to a few first transitions.

In addition, in the extended transducer-based model we introduce new features, and new aggregators, which enlarges the class of global constraints that can be described using our approach.

Chapter 7

Synthesising Parameterised Bounds

This chapter is an extended version of an article published in the *Constraints* journal [14]. The final authenticated version of this article is available online at: `http://dx.doi.org/10.1007/s10601-017-9276-z`.

It is currently unknown in general, how to maintain efficiently domain consistency for time-series constraints. Computing bounds on the result variable R of a $g_f_\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint is a way to potentially handle the combinatorial aspect and thus improve propagation. Since we have too many time-series constraints deriving such bounds needs to be done in a systematic way. Motivated by this, we sketched in [8] a methodology to obtain such bounds and illustrated it only for time-series constraints when $g = \texttt{max}$ and $f = \texttt{min}$.

The contribution of this chapter, which makes explicit the approach sketched in [8], is to introduce the notion of *regular-expression characteristic* that provides a unified way to concisely express bounds on the result variable R of a time-series constraint. Six regular-expression characteristics are introduced, which allows coming up in a compositional way with bounds when $\langle g, f\rangle \in \{\langle \texttt{sum}, \texttt{one}\rangle, \langle \texttt{max}, \texttt{width}\rangle, \langle \texttt{min}, \texttt{width}\rangle, \langle \texttt{sum}, \texttt{width}\rangle\}$: five main theorems (see Theorems 7.2.1, 7.2.2, 7.3.1, 7.3.2, and 7.3.3) allow obtaining 95 bounds implemented in Volume II of the Global Constraint Catalogue [10]. When the time-series variables $\langle X_1, X_2, \ldots, X_n\rangle$ have the same interval integer domain, these bounds are sharp for all the 22 regular expressions of Table 5.2. We now put in perspective with existing work the contribution of this chapter.

Going back to the work of Schützenberger [124], *regular cost functions* are quantitative extensions of regular languages that correspond to a function mapping a word to an integer value or infinity (QRE for short). Recently there was a rise of interest in this area, both from a theoretical perspective with max-plus automata [56], and from a practical point of view with the synthesis of cost register automata [4] for data streams [5]. Within constraint programming constraints using automata and register automata were introduced in [109] and in [20, 60], respectively, the latter also computing an integer value from a word. More recently, the work on synthesising register automata from transducers [22] in the context of time-series constraints is part of the QRE line of research. While most previous mentioned works give quantitative extensions of regular languages as their general motivation, to the best of our knowledge none of them introduced the concept of regular-expression characteristic, which is the key abstraction proposed here. The chapter is structured in the following way:

- ◇ In Section 7.1, we first introduce a notation system for denoting regular-expression characteristics. Then we present six regular-expression characteristics, which characterise different quantitative aspects of regular expressions useful for capturing some of their combinatorial flavour. Finally, based on two of these characteristics, we provide a necessary condition for the occurrence of a word of the language of a regular expression in the signature of a time series.
- ◇ In Section 7.2, we show how to obtain generic bounds for time-series constraints whose result variables are constrained to be the number of occurrences of a regular expression in the signature a time series, i.e. time-series constraints where $g = \texttt{sum}$ and $f = \texttt{one}$.

- ◇ In Section 7.3, we show how to obtain generic bounds for the result variables of time-series constraints for which the feature f is `width`, and the aggregator g is in $\{\texttt{max}, \texttt{min}, \texttt{sum}\}$.
- ◇ In Section 7.4, we synthesise all the results on bounds we have so far from the CP paper [8], and from Sections 7.2 and 7.3 of this thesis: for each bound we recall (1) the regular-expression characteristics it uses, (2) the generic theorem it comes from, and (3) the properties on regular expressions under which the bound is sharp.

7.1 Regular-Expression Characteristics

To obtain parameterised bounds, this section introduces regular-expression characteristics used for deriving sharp lower and upper bounds on the result value of a time-series constraint when the feature is either `one` or `width`. For all regular-expression characteristics, we use the notation system described in Section 7.1.1. We introduce the following characteristics:

- ○ The *size* of a regular expression in Section 7.1.2.
- ○ The *height* of a regular expression in Section 7.1.3.
- ○ The *range* of a regular expression wrt a time-series length in Section 7.1.4.
- ○ The *set of inducing words* of a regular expression in Section 7.1.5.
- ○ The *overlap* of a regular expression wrt an integer interval domain in Section 7.1.6.
- ○ The *smallest variation of maxima* of a regular expression wrt an integer interval domain in Section 7.1.7.

For instance, given a regular expression σ, the size characteristic is the length of a shortest word in $\mathcal{L}_\sigma$, while the overlap corresponds to the maximal overlap of two words in $\mathcal{L}_\sigma$ provided it is bounded by a constant. These two regular-expression characteristics are, for example, used to express the maximum number of occurrences of σ one can pack within the signature of a time series of length n.

Section 7.1.8 presents a summary example combining all the introduced regular-expression characteristics for the DECREASING_TERRACE regular expression. Section 7.1.9 introduces a necessary and sufficient condition for the existence of at least one occurrence of a regular expression within the signature of a time series under some hypothesis on the domain of the time-series variables. Table 7.1 provides for each of the 22 regular expressions in Table 5.2 the corresponding value of each regular-expression characteristic. Within Appendix C we give a table for each characteristic and also provide an illustrative example for each regular expression in Table 5.2.

7.1.1 A Notation System for Regular-Expression Characteristics

We introduce a notation system for naming the regular-expression characteristics. A regular-expression characteristic C is a function, denoted by $C_E^P(V)$, whose arguments are E, P, and V explained below:

- ○ E is either a regular expression over $\Sigma = \{<, =, >\}$ or a triple $\langle g, f, \sigma\rangle$, where g is an aggregator, f is a feature and σ is a regular expression over $\Sigma = \{<, =, >\}$. In the latter case, σ is considered in the context of the g_f_σ time-series constraint.
- ○ P is a subset, possible empty, drawn from $\{\ell, u, n\}$, where $[\ell, u]$ is the domain of the variables of a time series, and n is the length of a given time series.
- ○ V is a vector of additional arguments, which are different from E, ℓ, u, and n. If V is empty, then we simply write C_E^P. Quite often these additional arguments correspond to words in $\mathcal{L}_E$ since a characteristic C_E^P will be defined in terms of another characteristic $C_E^P(V)$: for instance the height of a regular expression E will be defined in terms of the heights of words in $\mathcal{L}_E$.

The domain of the function $C_E^P(V)$ is the Cartesian product of the following elements in the given order:

- ○ The domain of E, which is the set of all regular expressions over Σ, denoted by $\mathcal{R}_\Sigma$, if E is a regular expression, and is the set of all time-series constraints, denoted by $\mathcal{T}$, if E is a triple of an aggregator, a feature, and a regular expression.
- ○ The Cartesian product of the domains of the elements of P, if P is not empty.

- The Cartesian product of the domains of the arguments of V, if V is not empty.

The alphabet (Latin or Greek) from which comes the symbol 'C' depends on the type of values returned by $C_E^P(V)$:

- If the codomain of $C_E^P(V)$ is $\mathbb{Z}$, then 'C' is a lower-case Greek letter, e.g., δ.
- If the codomain of $C_E^P(V)$ is the power set of some set, then 'C' is an upper-case Greek letter, e.g., Δ.
- If $C_E^P(V)$ returns an interval, then 'C' is an upper-case Latin letter in calligraphy, e.g., $\mathcal{D}$.

Some regular-expression characteristics are associated with, either the lower or the upper bound on the value of the result variable of a time-series constraint.

In this case, the ones associated with the upper (respectively lower) bound are denoted by $\overline{C}_E^P(V)$ (respectively $\underline{C}_E^P(V)$).

7.1.2 Size

This section introduces the *size* regular-expression characteristic; it will be used in Theorem 7.2.2 for computing the sharp upper bound on the number of occurrences of a regular expression within the signature of a time series. The size of a regular expression σ is the minimum number of letters in a word among all words in $\mathcal{L}_\sigma$. Intuitively, to maximise the number of occurrences of σ within the signature of a time series, every word in this signature should be as short as possible, i.e. its length should be the size of σ. This characteristic is also used for defining a necessary and sufficient condition, see Property 7.1.1, for the existence of at least one occurrence of a regular expression within the signature of a time series over an integer interval domain.

Definition 7.1.1 (Size)**.** Consider a regular expression σ. The *size* of σ, denoted by ω_σ, is a function that maps an element of $\mathcal{R}_\Sigma$ to $\mathbb{N}$. It is defined by $\omega_\sigma = \min_{v \in \mathcal{L}_\sigma} |v|$.

Example 7.1.1 (Size of a regular expression)**.** We illustrate the size characteristic for two regular expressions.

- Consider the $\sigma =$ ZIGZAG regular expression. There are two shortest words in $\mathcal{L}_\sigma$, namely '<><' and '><>'. Both have length 3, thus the size of σ is 3. Hence, any extended σ-pattern has at least $3 + 1$ time-series variables.
- Consider the $\sigma =$ DECREASING_TERRACE regular expression. There is a single shortest word in $\mathcal{L}_\sigma$, namely '>=>'. Thus the size of σ is 3. Hence, any extended σ-pattern has at least $3 + 1$ time-series variables. △

7.1.3 Height

We introduce the notion of height of a regular expression, which is used for defining a necessary and sufficient condition, see Property 7.1.1, for the existence of at least one occurrence of a regular expression within the signature of a time series. This regular-expression characteristic is also used in Theorem 7.2.2 of Section 7.2 for computing a sharp upper bound on the number of occurrences of a regular expression within the signature of a time series. Definitions 7.1.2 and 7.1.3 are used for introducing Definition 7.1.4.

Definition 7.1.2 (Set of supporting time series)**.** Consider a regular expression σ and an integer interval domain $[\ell, u]$. The *set of supporting time series* of *a word* v in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$, denoted by $\Omega_\sigma^{\langle \ell, u \rangle}(v)$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{Z} \times \mathbb{Z} \times \Sigma^*$ to $\mathcal{P}(\mathbb{Z}^*)$, where $\mathcal{P}(\mathbb{Z}^*)$ is the power set of $\mathbb{Z}^*$. Each element of $\Omega_\sigma^{\langle \ell, u \rangle}(v)$ is a time series over $[\ell, u]$ whose signature is v, and is called a *supporting time series of* v *wrt* $\langle \ell, u \rangle$.

Definition 7.1.3 (Height of a word)**.** Consider a regular expression σ. The *height* of a word v in $\mathcal{L}_\sigma$, denoted by $\eta_\sigma(v)$, is a function that maps an element of $\mathcal{R}_\Sigma \times \Sigma^*$ to $\mathbb{N}$. It is defined by $\eta_\sigma(v) = \min_{\Omega_\sigma^{\langle \ell,u \rangle}(v) \neq \emptyset} (u - \ell)$, where $[\ell, u]$ is an integer interval domain.

Definition 7.1.4 (Height)**.** Consider a regular expression σ. The *height* of σ, denoted by η_σ, is a function that maps an element of $\mathcal{R}_\Sigma$ to $\mathbb{N}$. It is defined by $\eta_\sigma = \min_{v \in \mathcal{L}_\sigma} \eta_\sigma(v)$.

Example 7.1.2 (Height)**.** We illustrate the notion of height for two regular expressions.

- Consider the $\sigma =$ DECREASING regular expression and an integer interval domain $[\ell, u]$. When $u - \ell = 0$, there does not exist a time series over $[\ell, u]$ whose signature is a word of $\mathcal{L}_\sigma$; but when $u - \ell = 1$, there exists a time series $\langle u, u-1 \rangle$ over $[\ell, u]$ whose signature is the single word '$>$' of $\mathcal{L}_\sigma$. Hence, the height of σ equals 1.
- Consider the $\sigma =$ DECREASING_TERRACE regular expression and an integer interval domain $[\ell, u]$. When $u - \ell \leq 1$, there does not exist a time series over $[\ell, u]$ whose signature is a word in $\mathcal{L}_\sigma$; but when $u - \ell = 2$, there exists a time series over $[\ell, u]$ whose signature is a word, for example '$>=>$', in $\mathcal{L}_\sigma$. Hence, the height of σ equals 2. △

7.1.4 Range

This section introduces a regular-expression characteristic needed by Theorems 7.3.1, 7.3.2, and 7.3.3 for characterising sharp bounds on the result value of a time-series constraint when the feature is `width`. This characteristic, described in Definition 7.1.5, is called the *range* of a regular expression σ, and shows the variation of the minimum height of words of $\mathcal{L}_\sigma$ for words of increasing length.

Definition 7.1.5 (Range)**.** Consider a regular expression σ and a time series length n. The *range* of σ wrt $\langle n \rangle$, denoted by $\phi_\sigma^{\langle n \rangle}$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{N}$ to $\mathbb{N}$. It is defined by $\phi_\sigma^{\langle n \rangle} = \min_{v \in \mathcal{L}_\sigma,\ |v| = n-1} \eta_\sigma(v)$, where $\eta_\sigma(v)$ is the height of the word v. If $\mathcal{L}_\sigma$ does not contain any word of length $n-1$, then the value of $\phi_\sigma^{\langle n \rangle}$ is undefined.

Example 7.1.3 (Range)**.** Consider the $\sigma =$ STEADY_SEQUENCE regular expression. For every integer $n > \omega_\sigma$, the language $\mathcal{L}_\sigma$ contains a word with $n - 1$ equalities. Any word of this type has a height of 0. Hence, the range of σ is a constant function of n, which equals 0.

7.1.5 Set of Inducing Words

Given a disjunction-capsuled regular expression σ, we first introduce the notion of *inducing word* of $\mathcal{L}_\sigma$, which is a maximum sequence of letters that appears in every word of $\mathcal{L}_\sigma$ in a fixed order. Then we generalise this notion to any disjunction of disjunction-capsuled regular expressions. This notion will be further used in Property 7.2.1 and Theorem 7.2.1 for proving the lower bound on the number of σ-patterns in a time series, which very often is 0.

Definition 7.1.6 (Inducing word)**.** Consider a disjunction-capsuled regular expression σ. The (unique) non-empty shortest word of $\mathcal{L}_\sigma$ is the *inducing word* of $\mathcal{L}_\sigma$.

Definition 7.1.7 (Set of inducing words)**.** Consider a regular expression σ that is in the form of $\sigma = \sigma_1 \,|\, \sigma_2 \,|\, \ldots \,|\, \sigma_t$ with $t \geq 1$, where every σ_i (with $i \in [1, t]$) is a disjunction-capsuled regular expression. The *set of inducing words* of σ, denoted by Θ_σ, is a function that maps an element of $\mathcal{R}_\Sigma$ to $\mathcal{P}(\Sigma^*)$, where $\mathcal{P}(\Sigma^*)$ is the power set of Σ^*. The value of Θ_σ is the union of inducing words of every σ_i.

Example 7.1.4 (Set of inducing words)**.** We now illustrate the notion of inducing words for two regular expressions.

- Consider the DECREASING = '>' regular expression, which is disjunction-capsuled. The word $v =$ '>' is the unique inducing word of $\mathcal{L}_{\text{DECREASING}}$. Hence, $\Theta_{\text{DECREASING}} = \{\text{'>'}\}$.
- Consider the INFLEXION = '$< (< \mid =)^* > \mid > (> \mid =)^* <$' regular expression. It can be represented as $\text{INFLEXION}_1|\text{INFLEXION}_2$, where INFLEXION_1 is the '$< (< \mid =)^* >$' regular expression, INFLEXION_2 is the '$> (> \mid =)^* <$' regular expression, and both INFLEXION_1 and INFLEXION_2 are disjunction-capsuled. The word $z =$ '<>' is the inducing word of $\mathcal{L}_{\text{INFLEXION}_1}$, the word $v =$ '><' is the inducing word of $\mathcal{L}_{\text{INFLEXION}_2}$, and both z and v are inducing words of $\mathcal{L}_{\text{INFLEXION}}$. Hence, the set of inducing words of INFLEXION is $\{\text{'<>'}, \text{'><'}\}$. △

7.1.6 Overlap

This section introduces the *overlap* regular-expression characteristic; it will be used in Theorem 7.2.2 for computing the sharp upper bound on the number of occurrences of a regular expression within the signature of a time series. The overlap of a regular expression σ wrt an integer interval domain $[\ell, u]$ is the maximum number of common variables between two extended σ-patterns among all time series over $[\ell, u]$. Note that, as it will be illustrated in Example 7.1.6, a small value for $u - \ell$ can potentially induce a smaller overlap. Intuitively, to maximise the number of occurrences of σ within the signature of a time series, every two consecutive extended σ-patterns should have the maximum number of common time-series variables, i.e. this value is the overlap of σ wrt $[\ell, u]$. To define the overlap of a regular expression σ wrt to an integer interval domain $[\ell, u]$, Definition 7.1.8 first introduces the notion of *set of superpositions* of two words v and w in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$. Intuitively, the superposition of v and w wrt $\langle \ell, u \rangle$ is the signature z of some ground time series over $[\ell, u]$ that contains at least two σ-patterns, i.e. z has a prefix v and a suffix w and its length does not exceed the length of vw.

Definition 7.1.8 (Set of superpositions)**.** Consider a regular expression σ and an integer interval domain $[\ell, u]$. The *set of superpositions* of two words, v and w in $\mathcal{L}_\sigma$, wrt $\langle \ell, u \rangle$, denoted by $\Gamma_\sigma^{\langle \ell, u \rangle}(v, w)$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{Z} \times \mathbb{Z} \times \Sigma^* \times \Sigma^*$ to $\mathcal{P}(\Sigma^*)$, where $\mathcal{P}(\Sigma^*)$ is the power set of Σ^*. Each element z in $\Gamma_\sigma^{\langle \ell, u \rangle}(v, w)$ is a word over Σ, called a superposition of v and w wrt $\langle \ell, u \rangle$ and satisfying all the following conditions:

(1) $z \notin \mathcal{L}_\sigma$, (2) $\Omega_\sigma^{\langle \ell, u \rangle}(z) \neq \emptyset$, (3) v is a prefix of z, (4) w is a suffix of z, (5) $|z| \leq |vw|$.

Example 7.1.5 (Set of superpositions)**.** We now illustrate the concept of superposition on two examples.

- Consider $\sigma =$ ZIGZAG, and an integer interval domain $[\ell, u]$ allowing to have at least one zigzag, i.e. $u - \ell \geq 1$. We compute a superposition of the pair $\langle v, v \rangle$, where $v =$ '<><' $\in \mathcal{L}_\sigma$. Let z denote the word '<><><'.
  - First, assume that $u - \ell = 1$.
    - The word '<><><' is not a superposition of v and v wrt $\langle \ell, u \rangle$, because Condition (1) of Definition 7.1.8 is violated, although all other conditions of Definition 7.1.8 are satisfied.
    - Even if Conditions (1), (3), (4), and (5) of Definition 7.1.8 are satisfied for the word z, it is not a superposition of v and v wrt $\langle \ell, u \rangle$, since the number of consecutive increases in the word z is 2, which is strictly greater than $u - \ell = 1$, and thus Condition (2) of Definition 7.1.8 is violated. Hence, $\Omega_\sigma^{\langle \ell, u \rangle}(z) = \emptyset$.

    Indeed, when $u - \ell = 1$, there is no superposition of v and v, because any word different from z satisfying the first four conditions of Definition 7.1.8 will violate Condition (5) of Definition 7.1.8, i.e. will be strictly longer than $2 \cdot |v|$, thus $\Gamma_\sigma^{\langle \ell, u \rangle}(v, v) = \emptyset$.
  - Now assume that $u - \ell > 1$. Then, $\Omega_\sigma^{\langle \ell, u \rangle}(z) \neq \emptyset$, and the word z is the only superposition of v and v wrt $\langle \ell, u \rangle$, thus $\Gamma_\sigma^{\langle \ell, u \rangle}(v, v) = \{\text{'<><><>'}\}$.
- Consider $\sigma =$ DECREASING_TERRACE, and an integer interval domain $[\ell, u]$ allowing to have at least one occurrence of σ in the signature of a time series over $[\ell, u]$, i.e. $u - \ell \geq 2$. We compute a superposition of the pair $\langle v, v \rangle$, where $v =$ '>=>' $\in \mathcal{L}_\sigma$. Let z denote '>=>=>'.

- First, assume that $u - \ell = 2$. The word z is not a superposition of v and v, since the number of consecutive decreases in the word z is 3, which is strictly greater than $u - \ell = 2$, and thus $\Omega_\sigma^{\langle \ell,u \rangle}(z) = \emptyset$. Indeed, when $u - \ell = 2$, there is no superposition of v and v, because any word different from z satisfying the first four conditions of Definition 7.1.8 will violate Condition (5) of Definition 7.1.8, i.e. will be strictly longer than $2 \cdot |v|$, thus $\Gamma_\sigma^{\langle \ell,u \rangle}(v, v) = \emptyset$.
- Now assume that $u - \ell = 3$. Then, $\Omega_\sigma^{\langle \ell,u \rangle}(z) \neq \emptyset$, and the word z is the only superposition of v and v, thus $\Gamma_\sigma^{\langle \ell,u \rangle}(v, v) = \{\text{`>=>=>'}\}$.
- Finally, assume that $u - \ell \geq 4$. The sets of supporting time series of both words `>=>=>` and `>=>>=>` wrt $\langle \ell, u \rangle$ are not empty, and these two words are the only superpositions of v and v wrt $\langle \ell, u \rangle$, thus $\Gamma_\sigma^{\langle \ell,u \rangle}(v, v) = \{\text{`>=>=>'}, \text{`>=>>=>'}\}$. △

For a regular expression σ and an integer interval domain $[\ell, u]$, we now introduce the *overlap* characteristic of σ wrt $\langle \ell, u \rangle$, which is a crucial component in the sharp upper bound formula stated in Theorem 7.2.2. It corresponds to the maximum number of time-series variables that can be shared by two consecutive extended σ-patterns: when maximising the number of σ-patterns in a time series, we need to enforce as many consecutive extended σ-patterns as possible to have as many common time-series variables as possible.

Definition 7.1.9 (Overlap of two words)**.** Consider a regular expression σ and an integer interval domain $[\ell, u]$. The *overlap* of two words, v and w in $\mathcal{L}_\sigma$, wrt $\langle \ell, u \rangle$, denoted by $o_\sigma^{\langle \ell,u \rangle}(v, w)$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{Z} \times \mathbb{Z} \times \Sigma^* \times \Sigma^*$ to $\mathbb{N}$. It is defined by

$$o_\sigma^{\langle \ell,u \rangle}(v, w) = \begin{cases} \left(|vw| - \min_{z \in \Gamma_\sigma^{\langle \ell,u \rangle}(v,w)} |z| \right) + 1 & \text{if } \Gamma_\sigma^{\langle \ell,u \rangle}(v, w) \neq \emptyset, \quad (7.1) \\ 0, & \text{otherwise.} \quad (7.2) \end{cases}$$

Case (7.1) of Definition 7.1.9 corresponding to condition $\Gamma_\sigma^{\langle \ell,u \rangle}(v, w) \neq \emptyset$ states that the overlap is strictly greater than 0 iff there exists at least one ground time series over $[\ell, u]$ that is not strictly longer than $|vw|$ and that has at least two σ-patterns corresponding to the occurrences of v and w in its signature. The term $|vw| - \min_{z \in \Gamma_\sigma^{\langle \ell,u \rangle}(v,w)} |z|$ denotes the maximum possible overlap that is actually achieved by the shortest superposition. The increment $+1$ denotes the fact that we count the number of time-series variables rather than the number of signature variables.

We now generalise in Definition 7.1.10 the notion of overlap wrt $\langle \ell, u \rangle$ upon a regular expression.

Definition 7.1.10 (Overlap)**.** Consider a regular expression σ and an integer interval domain $[\ell, u]$. The *overlap* of σ wrt $\langle \ell, u \rangle$, denoted by $o_\sigma^{\langle \ell,u \rangle}$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{Z} \times \mathbb{Z}$ to $\mathbb{N}$. If there exists a constant c in $\mathbb{N}$ such that for any pair of words v, w in $\mathcal{L}_\sigma$, the value of $o_\sigma^{\langle \ell,u \rangle}(v, w)$ is bounded by c, then the overlap of σ wrt $\langle \ell, u \rangle$ is defined by $o_\sigma^{\langle \ell,u \rangle} = \max_{v,w \in \mathcal{L}_\sigma} o_\sigma^{\langle \ell,u \rangle}(v, w)$. Otherwise, $o_\sigma^{\langle \ell,u \rangle}$ is undefined.

By Definition 7.1.10, we need to compute the overlap of σ wrt every pair of values $\langle \ell, u \rangle$, i.e. every domain $[\ell, u]$. Note that it is enough to compute the overlap of σ wrt $\langle \ell, u \rangle$ once for every value of the difference $u - \ell$ permitting an occurrence of σ in the signature of a time series, i.e. for a difference that is greater than or equal to the height of the regular expression σ. While in the general case, we do need to consider every value of $u - \ell$, this is not required for all the 22 regular expressions in Table 5.2, where we only need to consider at most two different values of $u - \ell$.

Note that the overlap of a regular expression wrt an integer interval domain is similar to the regular-expression overlap introduced in [68] (see Definition 5.2.1), however Definition 7.1.10 operates with time-series variables while Definition 5.2.1 operates only with the regular language.

Example 7.1.6 (Overlap)**.** We successively consider values of the overlap of three regular expressions.

- Consider the $\sigma =$ ZIGZAG regular expression, whose height η_σ is 1.
  - If $u - \ell \leq \eta_\sigma = 1$, then $o_\sigma^{\langle \ell,u \rangle} = 0$, because as shown in Example 7.1.5, for any pair of words in $\mathcal{L}_\sigma$, the set of their superpositions wrt $\langle \ell, u \rangle$ is empty.
  - If $u - \ell \geq \eta_\sigma + 1 = 2$, then $o_\sigma^{\langle \ell,u \rangle} = 1$ and is obtained, for example, for the pair '<><' and '<><', which share one time-series variable in the superposition '<><<><'.
  - For any other value of $u - \ell \geq 2$, the value of the overlap of σ wrt $\langle \ell, u \rangle$ equals 2 as well.
- Consider the $\sigma =$ DECREASING_TERRACE regular expression, whose height η_σ is 2.
  - If $u - \ell \leq \eta_\sigma = 2$, then $o_\sigma^{\langle \ell,u \rangle} = 0$, because as shown in Example 7.1.5, for any pair of words in $\mathcal{L}_\sigma$, the set of their superpositions wrt $\langle \ell, u \rangle$ is empty.
  - If $u - \ell \geq \eta_\sigma + 1 = 3$, then $o_\sigma^{\langle \ell,u \rangle} = 2$ and is obtained, for example, for the pair '>=>' and '>=>', and their superposition '>=>=>'.
  - For any other value of $u - \ell \geq 4$, the value of the overlap of σ wrt $\langle \ell, u \rangle$ equals 2 as well.
- Consider the $\sigma =$ '<=* | =*>' regular expression and an integer interval domain $[\ell, u]$ such that $u > \ell$. The overlap of σ wrt $\langle \ell, u \rangle$ is undefined, because for any constant c in $\mathbb{N}$, there always exists a pair of words of length $c + 1$ whose overlap wrt $\langle \ell, u \rangle$ equals $c + 1$. △

7.1.7 Smallest Variation of Maxima

This section introduces the *smallest variation of maxima* regular-expression characteristic, which is used in Theorem 7.2.2 for computing the sharp upper bound on the number of occurrences of the regular expression within the signature of a time series. To maximise the number of occurrences of a regular expression σ in the signature of a time series over an integer interval domain $[\ell, u]$, we select extended σ-patterns of minimum length $\omega_\sigma + 1$ such that two consecutive extended σ-patterns maximise the number of shared time-series variables, i.e. share $o_\sigma^{\langle \ell,u \rangle}$ variables. Unfortunately, for a few regular expressions like DECREASING_TERRACE, it is not always possible that all extended σ-patterns share $o_\sigma^{\langle \ell,u \rangle}$ time-series variables: since we decrease by at least one unit between two consecutive overlapping extended σ-patterns we will be blocked at some point by the lower limit ℓ, even if we start from the upper limit u. To maximise the number of σ-patterns in a time series, we must decrease as little as possible on two consecutive overlapping extended σ-patterns. Definition 7.1.13 formalises the notion of *smallest variation of the maxima* of a regular expression wrt $\langle \ell, u \rangle$. First, Definition 7.1.11 defines the notion of shift of a proper factor in a word in the language of a regular expression wrt some integer interval domain. Then, using this notion, Definition 7.1.12 (respectively Definition 7.1.13) introduces the smallest variation of the maxima of two words (respectively a language $\mathcal{L}_\sigma$) wrt $\langle \ell, u \rangle$.

Definition 7.1.11 (Shift)**.** Consider a regular expression σ and an integer interval domain $[\ell, u]$. The *shift* of a proper factor w in a word v in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$, denoted by $\overline{\delta}_\sigma^{\langle \ell,u \rangle}(v, w, i)$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{Z} \times \mathbb{Z} \times \Sigma^* \times \Sigma^* \times \mathbb{N}$ to $\mathbb{N}$. It is defined by

$$\overline{\delta}_\sigma^{\langle \ell,u \rangle}(v, w, i) = \min_{t \in \Omega_\sigma^{\langle \ell,u \rangle}(v)} \min_{x \in t_{w_i}} (\max(t) - x),$$

where $\max(t)$ is the maximum value of a time series t, a supporting time series of v wrt $\langle \ell, u \rangle$, and t_{w_i} is a subseries of t corresponding to the i^{th} extended σ-pattern whose signature contains w. If w is not a proper factor of v, or i is strictly greater than the number of occurrences of w in v, then $\overline{\delta}_\sigma^{\langle \ell,u \rangle}(v, w, i)$ is undefined.

Consider a regular expression σ and an integer interval domain $[\ell, u]$. For any v in $\mathcal{L}_\sigma$, if $u - \ell \geq \eta_\sigma(v)$, then the value of $\overline{\delta}_\sigma^{\langle \ell,u \rangle}(v, w, i)$ does not depend on the domain $[\ell, u]$, because there always exists a time series in $\Omega_\sigma^{\langle \ell,u \rangle}(v)$ where each variable has its largest value compared to the other time series of $\Omega_\sigma^{\langle \ell,u \rangle}(v)$. Then, $\overline{\delta}_\sigma^{\langle \ell,u \rangle}(v, w, i)$ does not depend on the values in the domain, but only on the structure of the word v. Hence, w.l.o.g. the notation for $\overline{\delta}_\sigma^{\langle \ell,u \rangle}(v, w, i)$ can be simplified to $\overline{\delta}_\sigma(v, w, i)$.

Example 7.1.7 (Shift)**.** Consider $\sigma = \text{DECREASING_TERRACE}$ when $u - \ell \geq \eta_\sigma = 3$, and two words $v =$ '>=>=>' and $w =$ '>=>'. The word v contains two occurrences of w, thus the value of $\overline{\delta}_\sigma(v, w, i)$ is defined when $i \in \{1, 2\}$:

- When i is 1, the value of $\overline{\delta}_\sigma(v, w, 1)$ equals 0, since the first extended σ-pattern whose signature is w necessarily contains the maximum of any time series in $\Omega_\sigma^{\langle \ell, u \rangle}(v)$.
- When i is 2, the value of $\overline{\delta}_\sigma(v, w, 2)$ equals 1, since the maximum of the second extended σ-pattern whose signature is w has a difference of at least one with the maximum of any time series in $\Omega_\sigma^{\langle \ell, u \rangle}(v)$.

△

Definition 7.1.12 (Smallest variation of maxima of two words)**.** Consider a regular expression σ and an integer interval domain $[\ell, u]$. The *smallest variation of maxima* of superpositions of two words w and v in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$, denoted by $\delta_\sigma^{\langle \ell, u \rangle}(v, w)$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{Z} \times \mathbb{Z} \times \Sigma^* \times \Sigma^*$ to $\mathbb{N}$. It is defined by

$$\delta_\sigma^{\langle \ell, u \rangle}(v, w) = \begin{cases} \overline{\delta}_\sigma(z_*, v, 1) - \overline{\delta}_\sigma(z_*, w, p) & \text{if } \Gamma_\sigma^{\langle \ell, u \rangle}(v, w) \neq \emptyset, \\ 0 & \text{otherwise,} \end{cases}$$

where the word z_* belongs to $\Gamma_\sigma^{\langle \ell, u \rangle}(v, w)$, the value $\min_{z \in \Gamma_\sigma^{\langle \ell, u \rangle}(v, w)} |\overline{\delta}_\sigma(z, v, 1) - \overline{\delta}_\sigma(z, w, p)|$ is reached when z is z_*, and p is the number of occurrences of the word w in z_*.

In Definition 7.1.12, either $\overline{\delta}_\sigma(z_*, v, 1)$ or $\overline{\delta}_\sigma(z_*, w, p)$ equals zero, since for any time series t whose signature is z_*, at least one of the two extended σ-patterns contains the maximum of t.

The next lemma introduces a property of words whose smallest variation of maxima wrt some integer interval domain is not zero.

Lemma 7.1.1. Consider a regular expression σ and an integer interval domain $[\ell, u]$. If $\delta_\sigma^{\langle \ell, u \rangle}(v, w)$, the smallest variation of maxima of two words v and w in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$, is positive (respectively negative), then v (respectively w) does not contain any '>' (respectively '<').

Proof. For brevity, we consider only the case of $\delta_\sigma^{\langle \ell, u \rangle}(v, w)$ being positive, the case of a negative value of $\delta_\sigma^{\langle \ell, u \rangle}(v, w)$ being symmetric, and w.l.o.g. we assume that $v \neq w$.

Since $\delta_\sigma^{\langle \ell, u \rangle}(v, w) > 0$, there exists at least one superposition z of v and w wrt $\langle \ell, u \rangle$ such that $\overline{\delta}_\sigma(z, v, 1) = \delta_\sigma^{\langle \ell, u \rangle}(v, w)$, and $\overline{\delta}_\sigma(z, w, p) = 0$, where p is the number of occurrences of the word w in z. Assume that v contains at least one '>'. Let i denote the position of the first '>' in z, which is necessarily within its proper factor v. Since there exists a time series in $\Omega_\sigma^{\langle \ell, u \rangle}(z)$ such that the time-series variable at position i equals u, $\overline{\delta}_\sigma(z, v, 1)$ equals 0. This contradicts the fact that $\overline{\delta}_\sigma(z, v, 1) = \delta_\sigma^{\langle \ell, u \rangle}(v, w) > 0$, thus the word v does not contain any '>'. □

The following lemma describes the structures of words whose smallest variation of maxima wrt some integer interval domain is zero. A corollary to this lemma will be further used in Lemma 7.2.2, which describes the structure of a maximal time series under certain conditions.

Lemma 7.1.2. Consider a regular expression σ and an integer interval domain $[\ell, u]$. If for two non-empty words w and v in $\mathcal{L}_\sigma$, the value of $\delta_\sigma^{\langle \ell, u \rangle}(v, w)$ is zero, then the four following conditions must be satisfied:

1. If there exists a suffix of v that is in the language of '$>=^*$', then the word w does not belong to the language of '$(> \mid =)^+$'.
2. If the word v is in the language of '$(< \mid =)^* < (< \mid =)^*$', then there is no prefix of w belonging to the language of '$=^*<$'.
3. If there exists a suffix of v that is in the language of '$<=^*$' and '>' is a proper factor of v, then w is any word.

4. If the word v is in the language of '$=^+$', then there is no prefix of w belonging to the language of '$(< \mid =)^* < (< \mid =)^*$'.

In addition, the premise of exactly one of the four conditions must be true.

Proof. Each of the four conditions is of the form 'if A_i then B_i' with i in $\{1, 2, 3, 4\}$. It is easy to see that the premises A_1, A_2, A_3 and A_4 are mutually exclusive and $A_1 \vee A_2 \vee A_3 \vee A_4$ is true. By cases analysis on the satisfied condition A_i, we can show by contradiction that if $\delta_\sigma^{\langle \ell,u \rangle}(v, w)$ is zero, then B_i is also satisfied. □

Corollary 7.1.1. Consider a regular expression σ and an integer interval domain $[\ell, u]$, and two non-empty words v and w in the language of σ. If both $\delta_\sigma^{\langle \ell,u \rangle}(v, w)$ and $\delta_\sigma^{\langle \ell,u \rangle}(w, v)$ equal zero, then one of the following conditions must be satisfied:

1. Both words are in the language of '$=^+$'.
2. One of the words is in the language of '$=^+$', and the other is in the language of '$=^*> (> \mid = \mid <)^* <=^*$'.
3. Both words are in the language of '$=^*> (> \mid = \mid <)^* <=^* \mid =^*< (> \mid = \mid <)^* >=^*$'.

Proof. Directly follows from Lemma 7.1.2. □

Definition 7.1.13 (Smallest variation of maxima)**.** Consider a regular expression σ and an integer interval domain $[\ell, u]$. The *smallest variation of maxima* of σ wrt $\langle \ell, u \rangle$, denoted by $\delta_\sigma^{\langle \ell,u \rangle}$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{Z} \times \mathbb{Z}$ to $\mathbb{N}$. It is defined by

$$\delta_\sigma^{\langle \ell,u \rangle} = \begin{cases} \text{undefined} & \text{if } \exists\, v_1, v_2, w_1, w_2 \in \mathcal{L}_\sigma \text{ s.t. } \delta_\sigma^{\langle \ell,u \rangle}(v_1, w_1) > 0 \text{ and } \delta_\sigma^{\langle \ell,u \rangle}(v_2, w_2) < 0, \\ 0 & \text{if } o_\sigma^{\langle \ell,u \rangle} = 0, \\ \delta_\sigma^{\langle \ell,u \rangle}(v_*, w_*), & \text{otherwise ,} \end{cases}$$

where the words v_* and w_* both belong to $\mathcal{L}_\sigma$ and the value $\min\limits_{\substack{v,w \in \mathcal{L}_\sigma \\ o_\sigma^{\langle \ell,u \rangle}(v,w) \neq 0}} |\delta_\sigma^{\langle \ell,u \rangle}(v, w)|$ is reached when v is v_* and w is w_*.

Example 7.1.8 (Smallest variation of maxima)**.** Consider the $\sigma =$ DECREASING_TERRACE regular expression, an integer interval domain $[\ell, u]$ such that $u - \ell \geq 3$, and the superposition $z =$ '$>=>=>$' of the words $v =$ '$>=>$' and $v =$ '$>=>$' in $\mathcal{L}_\sigma$. The value of $\overline{\delta}_\sigma(z, v, 1) - \overline{\delta}_\sigma(z, v, 2)$ is equal to $0 - 1 = -1$. For any other pair of words of $\mathcal{L}_\sigma$ whose set of superpositions wrt $\langle \ell, u \rangle$ is not empty, we obtain the same or a smaller negative value. Hence, if two extended σ-patterns share some time-series variables, then the maximum of a second extended σ-pattern is at least one unit below, i.e. $\delta_\sigma^{\langle \ell,u \rangle} = -1$, from the maximum of the first extended σ-pattern. △

If $\delta_\sigma^{\langle \ell,u \rangle}$ is positive (respectively negative), then for any two extended σ-patterns that have at least one common time-series variable, the maximum of the first extended σ-pattern is strictly less (respectively greater) than the maximum of the second extended σ-pattern, e.g., for DECREASING_TERRACE, $\delta_\sigma^{\langle \ell,u \rangle}$ equals -1, but for INCREASING_TERRACE, $\delta_\sigma^{\langle \ell,u \rangle}$ equals $+1$.

7.1.8 Summary Example Illustrating All Regular-Expression Characteristics

This section illustrates the various regular-expression characteristics introduced in the previous sections.

Example 7.1.9 (Various regular-expression characteristics)**.** Consider the $\sigma =$ DECREASING_TERRACE regular expression and a time series X of length 6 over an integer interval domain $[0, 3]$. Let us recall the characteristics mentioned in Examples 7.1.1, 7.1.2, 7.1.6, and 7.1.8, which are illustrated in Figure 7.1.

- The *size* of σ, denoted by ω_σ, equals 3.

- The *height* of σ, denoted by η_σ, equals 2. This is the difference between the y-coordinates of the points L_1 and S in Figure 7.1, which are respectively the maximum and the minimum points of the first extended σ-pattern of X.
- The *range* of σ wrt $\langle n \rangle$, denoted by $\phi_\sigma^{\langle n \rangle}$, equals 2, with $n \in \mathbb{N}$ being greater than or equal to $\omega_\sigma = 3$.
- The *overlap* of σ wrt $\langle 0, 3 \rangle$, denoted by $o_\sigma^{\langle 0,3 \rangle}$, equals 2. It is the number of common points of the first and the second extended σ-patterns in Figure 7.1, i.e. the number of points coloured in violet.
- The *smallest variation of maxima* of σ wrt $\langle 0, 3 \rangle$, denoted by $\delta_\sigma^{\langle 0,3 \rangle}$, equals 1. It is the difference between the y-coordinates of the L_1 and the L_2 points in Figure 7.1, which are the maximum points of the first, respectively the second, extended σ-pattern of X. △

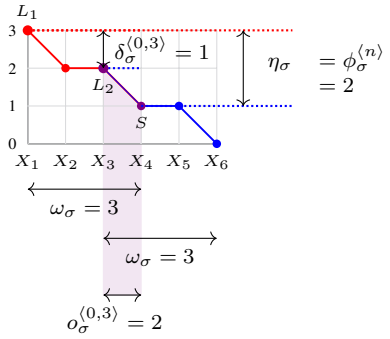


Figure 7.1 – A time series of length $n = 6$ over the integer interval domain $[0, 3]$ containing two extended σ-patterns, where σ is DECREASING_TERRACE. The x-axis is for time-series variables, the y-axis is for domain values. The first (respectively second) extended σ-pattern is shown in red (respectively blue). The common time-series variables of the two extended σ-patterns are coloured in violet. L_1 (respectively L_2) is the point whose y-coordinate is maximum among all points of the first (respectively second) extended σ-pattern. S is the point whose y-coordinate is minimum among all points of the first extended σ-pattern.

7.1.9 Necessary and Sufficient Condition for the Existence of an Occurrence of a Regular Expression

Consider a regular expression σ and a time series $X = \langle X_1, X_2, \ldots, X_n \rangle$ with every X_i ranging over the same integer interval domain. There exists a necessary and sufficient condition, based on the domains and the number of time-series variables, for σ to occur at least once within the signature of X. In order to define this condition we use the *size* of a regular expression, introduced in Definition 7.1.1, and the *height* of a regular expression, introduced in Definition 7.1.4.

Property 7.1.1 (necessary-sufficient condition)**.** Consider a regular expression σ and a time series $\langle X_1, X_2, \ldots, X_n \rangle$ with every X_i ranging over the same integer interval domain $[\ell, u]$. The *necessary-sufficient condition* is satisfied if the two following conditions hold:

(i) The value of n is strictly greater than ω_σ, the size of σ.

(ii) The difference between u and ℓ is greater than or equal to η_σ, the height of σ.

Example 7.1.10 (necessary-sufficient condition)**.** Consider the σ = DECREASING_TERRACE regular expression and a time series of length n over an integer interval domain $[\ell, u]$. We recall the values computed in Examples 7.1.2 and 7.1.1, namely the height of σ is 2, and the size of σ is 3. Hence, the necessary-sufficient condition is satisfied if $n > 3$ and $u - \ell \geq 2$. △

All formulae presented in all the next sections of this chapter assume that Property 7.1.1 holds. Table 7.1 provides for each of the regular expressions in Table 5.2 the corresponding value of each regular-expression characteristic.

name σ	ω_σ	η_σ	$\langle e_\sigma, c_\sigma \rangle$	$\phi_\sigma^{\langle n \rangle}$	Θ_σ	$o_\sigma^{\langle \ell,u \rangle}$	$\delta_\sigma^{\langle \ell,u \rangle}$
BUMP	5	2	undefined	$\begin{cases} 2 & \text{if } n = 6 \\ \text{undefined} & \text{otherwise} \end{cases}$	{' >><<>> '}	3	0
DEC	1	1	undefined	$\begin{cases} 1 & \text{if } n = 2 \\ \text{undefined} & \text{otherwise} \end{cases}$	{' > '}	$\begin{cases} 0 & \text{if } u - \ell \leq 1 \\ 1 & \text{otherwise} \end{cases}$	$\begin{cases} 0 & \text{if } u - \ell \leq 1 \\ -1 & \text{otherwise} \end{cases}$
DECSEQ	1	1	$\langle 0, 1 \rangle$	$\begin{cases} 1 & \text{if } n = 2 \\ 2 & \text{if } n > 2 \end{cases}$	{' > '}	0	0
DECTER	3	2	$\langle 0, 0 \rangle$	2	{' >=> '}	$\begin{cases} 0 & \text{if } u - \ell \leq 2 \\ 2 & \text{otherwise} \end{cases}$	$\begin{cases} 0 & \text{if } u - \ell \leq 2 \\ -1 & \text{otherwise} \end{cases}$
DIP	5	2	undefined	$\begin{cases} 2 & \text{if } n = 6 \\ \text{undefined} & \text{otherwise} \end{cases}$	{' <<><<< '}	3	0
GORGE	2	1	$\langle 0, 1 \rangle$	$\begin{cases} 1 & \text{if } n = 3 \\ 2 & \text{if } n > 3 \end{cases}$	{' >< '}	1	0
INC	1	1	undefined	$\begin{cases} 1 & \text{if } n = 2 \\ \text{undefined} & \text{otherwise} \end{cases}$	{' < '}	$\begin{cases} 0 & \text{if } u - \ell \leq 1 \\ 1 & \text{otherwise} \end{cases}$	$\begin{cases} 0 & \text{if } u - \ell \leq 1 \\ 1 & \text{otherwise} \end{cases}$
INCSEQ	1	1	$\langle 0, 1 \rangle$	$\begin{cases} 1 & \text{if } n = 2 \\ 2 & \text{if } n > 2 \end{cases}$	{' < '}	0	0
INCTER	3	2	$\langle 0, 0 \rangle$	2	{' <=< '}	$\begin{cases} 0 & \text{if } u - \ell \leq 2 \\ 2 & \text{otherwise} \end{cases}$	$\begin{cases} 0 & \text{if } u - \ell \leq 2 \\ 1 & \text{otherwise} \end{cases}$
INFLEXION	2	1	$\langle 0, 0 \rangle$	1	{' <> ', ' >< '}	2	0
PEAK	2	1	$\langle 0, 0 \rangle$	1	{' <> '}	1	0
PLAIN	2	1	$\langle 0, 0 \rangle$	1	{' >< '}	1	0
PLATEAU	2	1	$\langle 0, 0 \rangle$	1	{' <> '}	1	0
PROPPLAIN	3	1	$\langle 0, 0 \rangle$	1	{' >=< '}	1	0
PROPPLATEAU	3	1	$\langle 0, 0 \rangle$	1	{' <=> '}	1	0
STEADY	1	0	undefined	$\begin{cases} 0 & \text{if } n = 2 \\ \text{undefined} & \text{otherwise} \end{cases}$	{' = '}	1	0
STEADYSEQ	1	0	$\langle 0, 0 \rangle$	0	{' = '}	0	0
SDECSEQ	1	1	$\langle 1, 0 \rangle$	$n - 1$	{' > '}	0	0
SINCSEQ	1	1	$\langle 1, 0 \rangle$	$n - 1$	{' < '}	0	0
SUMMIT	2	1	$\langle 0, 1 \rangle$	$\begin{cases} 1 & \text{if } n = 3 \\ 2 & \text{if } n > 3 \end{cases}$	{' <> '}	1	0
VALLEY	2	1	$\langle 0, 0 \rangle$	1	{' >< '}	1	0
ZIGZAG	3	1	$\langle 0, 0 \rangle$	1	{' <>< ', ' ><> '}	$\begin{cases} 0 & \text{if } u - \ell \leq 1 \\ 1 & \text{otherwise} \end{cases}$	0

Table 7.1 – Regular-expression names σ and corresponding *size* ω_σ, *height* η_σ, *range* $\phi_\sigma^{\langle n \rangle}$ (for a non-fixed-length regular expression σ and for any $n > \omega_\sigma + 1$, $\phi_\sigma^{\langle n \rangle} = e_\sigma \cdot (n - 1 - \eta_\sigma) + c_\sigma + \eta_\sigma$), *set of inducing words* Θ_σ, *overlap* $o_\sigma^{\langle \ell,u \rangle}$, and *smallest variation of maxima* $\delta_\sigma^{\langle \ell,u \rangle}$, where BUMP, DEC, DECSEQ, DECTER, DIP, INC, INCSEQ, INCTER, PROPPLAIN, PROPPLATEAU, STEADYSEQ, SDECSEQ, SINCSEQ are respectively shortcuts for BUMP_ON_DECREASING_SEQUENCE, DECREASING, DECREASING_SEQUENCE, DECREASING_TERRACE, DIP_ON_INCREASING_SEQUENCE, INCREASING, INCREASING_SEQUENCE, INCREASING_TERRACE, PROPER_PLAIN, PROPER_PLATEAU, STEADY_SEQUENCE, STRICTLY_DECREASING_SEQUENCE, STRICTLY_INCREASING_SEQUENCE.

7.2 Time-Series Constraints with Feature ONE

The first family of time-series constraints we consider is the NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ family. Given a sequence $X = \langle X_1, X_2, \ldots, X_n\rangle$, where all X_i are integer variables, it enforces the number of occurrences of regular expression σ in X to be equal to R. Within this constraint family the aggregator is `sum`, and the feature is `one`. The main results of Section 7.2 are described by Theorems 7.2.1 and 7.2.2, which respectively provide a sharp lower bound and a sharp upper bound on the number of occurrences of a regular expression σ in the signature of a time series provided all X_i (with $i \in [1, n]$) have the same integer interval domain $[\ell, u]$. Section 7.2 is structured in the following way:

- ◇ First, Section 7.2.1 introduces Property 7.2.1, and gives a sharp lower bound on R provided Property 7.2.1 holds.
- ◇ Second, Section 7.2.2 provides an upper bound, not necessarily sharp, on R. This bound is valid for any regular expression σ for which the overlap characteristics is defined and does not exceed the size of σ.
- ◇ Third, Section 7.2.3 extends the upper bound on R of Section 7.2.2, and shows that the extended formula is sharp under some hypothesis on the regular-expression characteristics:
  - ∗ Section 7.2.3.1 defines Properties 7.2.2 and 7.2.3 of regular expressions that allow to obtain a sharp upper bound.
  - ∗ Section 7.2.3.2 describes the structure of a maximal time series for NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ provided either Property 7.2.2 or Property 7.2.3 holds.
  - ∗ Based on the structure of a maximal time series for NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$, Section 7.2.3.3 provides a sharp upper bound on R, provided either Property 7.2.2 or Property 7.2.3 holds.
- ◇ Finally, Section 7.2.3.4 gives a sharp upper bound on R in a special case of σ being STEADY_SEQUENCE, where neither Property 7.2.2 nor Property 7.2.3 is satisfied.

7.2.1 A Sharp Lower Bound on the Number of Pattern Occurrences

Consider a NB_$\sigma(X, R)$ time-series constraint with $X = \langle X_1, X_2, \ldots, X_n\rangle$, where every X_i (with $i \in [1, n]$) is over the same integer interval domain $[\ell, u]$. This section focusses on providing a sharp lower bound on R. For almost every regular expression of Table 5.2, we can assign the variables of X to values in $[\ell, u]$ in a way that the signature of X does not contain any occurrence of the regular expression σ. The only two exceptions are the STEADY $=$ '$=$' and the STEADY_SEQUENCE $=$ '$=^+$' regular expressions when $\ell = u$. The next theorem, namely Theorem 7.2.1, provides a sharp lower bound on R assuming the property that we now introduce holds.

Property 7.2.1 (NB-simple property)**.** A regular expression σ has the NB-*simple* property for an integer interval domain $[\ell, u]$ if σ is a disjunction of disjunction-capsuled regular expressions and if at least one of the following conditions holds:

(i) Every inducing word of σ includes at least one letter that is different from '$=$'.

(ii) Every inducing word of σ includes at least one '$=$', and $u > \ell$.

Theorem 7.2.1 (sharp lower bound for NB_σ)**.** Consider a NB_$\sigma(X, R)$ time-series constraint with $X = \langle X_1, X_2, \ldots, X_n\rangle$, where every X_i (with $i \in [1, n]$) is over the same integer interval domain $[\ell, u]$, and, where σ is a disjunction of disjunction-capsuled regular expressions. If σ has the NB-simple property for $[\ell, u]$, then a sharp lower bound on R is 0.

Proof. If Condition (i) of Property 7.2.1 is satisfied, then by definition of an inducing word, every word of $\mathcal{L}_\sigma$ contains at least one letter that is not '$=$'. Hence, the time series X, where all variables are assigned to the same value, has no occurrences of σ in its signature, and thus a sharp lower bound on R is 0.

If Condition (ii) of Property 7.2.1 is satisfied, then every word in $\mathcal{L}_\sigma$ contains at least one '$=$'. The ground time series of length n with alternating ℓ and $\ell + 1$ has no equalities in its signature, and thus no occurrences of σ. Hence, a sharp lower bound on R equals 0. □

Every regular expression in Table 5.2 has the NB-simple property for any integer interval domain $[\ell, u]$, except STEADY and STEADY_SEQUENCE for the domain $[\ell, u]$ such that $\ell = u$. We now consider the cases of STEADY and STEADY_SEQUENCE where neither condition of Property 7.2.1 holds, which means that Theorem 7.2.1 cannot be applied for computing a sharp lower bound on R.

Proposition 7.2.1 (sharp lower bound for NB_STEADY)**.** Consider a NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint with σ being the STEADY regular expression, and with every X_i ranging over the same integer interval domain $[\ell, u]$ such that $\ell = u$. A sharp lower bound on R equals $n - 1$.

Proof. Since $\ell = u$, there exists a single ground time series t of length n over $[\ell, u]$. All the time-series variables of t have the same value, namely ℓ, and thus its signature consists of $n - 1$ equalities. The number of occurrences of σ in the signature of t equals $n - 1$, which is thus a sharp lower bound on R. □

Proposition 7.2.2 (sharp lower bound for NB_STEADY_SEQUENCE)**.** Consider a NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint with σ being the STEADY_SEQUENCE regular expression, and with every X_i ranging over the same integer interval domain $[\ell, u]$ such that $\ell = u$. A sharp lower bound on R equals 1.

Proof. Since $\ell = u$, there exists a single ground time series t of length n over $[\ell, u]$. All the time-series variables of t have the same value, namely ℓ, and thus its signature consists of $n - 1$ equalities. The number of occurrences of σ in the signature of t equals 1, which is thus a sharp lower bound on R. □

7.2.2 A First Not Necessarily Sharp Upper Bound

Consider a NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[\ell, u]$. Lemma 7.2.1 of this section provides an upper bound, not necessarily sharp, on R. Intuitively, to get a maximum number of σ-patterns, every extended σ-pattern should be as short as possible and every two consecutive extended σ-patterns should have a maximum number of time-series variables in common. Although, it is not sharp in general, it is sharp for all regular expressions in Table 5.2, except DECREASING, INCREASING, DECREASING_TERRACE, and INCREASING_TERRACE.

We first define the notion of *interval without restart*, in order to identify a subseries such that every two consecutive extended σ-patterns within this subseries have $o_\sigma^{\langle \ell, u\rangle}$ common time-series variables. This notion will be reused in Section 7.2.3 for deriving a sharp upper bound on R.

Definition 7.2.1 (interval without restart)**.** Consider a regular expression σ and a ground time series $X = \langle X_1, X_2, \ldots, X_n\rangle$ over $[\ell, u]$. An *interval without restart* of X is any interval $[\alpha, \beta]$ (with $1 \leq \alpha \leq \beta \leq n$), such that all the following conditions hold:

(1) Every X_k (with $k \in [\alpha, \beta]$) belongs to at least one extended σ-pattern for which all time-series variables have indices in $[\alpha, \beta]$.

(2) The width of every extended σ-pattern whose time-series variable indices are in $[\alpha, \beta]$ is equal to $\omega_\sigma + 1$.

(3) Every pair of consecutive extended σ-patterns, whose time-series variable indices are in $[\alpha, \beta]$, share $o_\sigma^{\langle \ell, u\rangle}$ time-series variables.

(4) When $o_\sigma^{\langle \ell, u\rangle} > 0$ every extended σ-pattern, whose time-series variable indices are in $[\alpha, \beta]$, has no common time-series variables with any extended σ-pattern that has an index outside $[\alpha, \beta]$.

Note that, by Condition (4) of Definition 7.2.1, the intervals without restart of a ground time series are always disjoint. Consequently two consecutive extended σ-patterns belonging to distinct intervals without restart do not share any time-series variable.

Example 7.2.1 (interval without restart)**.** We consider an example of intervals without restart for the $\sigma =$ DECREASING_TERRACE regular expression. For the time series $X = \langle 4, 3, 3, 2, 2, 1, 4, 2, 2, 1\rangle$, the intervals $[1, 6]$ and $[7, 10]$ are intervals without restart corresponding to the subseries $t_1 = \langle 4, 3, 3, 2, 2, 1\rangle$ and $t_2 = \langle 4, 2, 2, 1\rangle$, because:

- ∗ Each X_i (with $i \in [1,6]$ or $i \in [7,10]$) belongs to at least one extended σ-pattern (Condition (1) of Definition 7.2.1).
- ∗ The subseries t_1 (respectively t_2) contains 2 (respectively 1) extended σ-patterns of shortest length 4 (Condition (2) of Definition 7.2.1).
- ∗ The two consecutive extended σ-patterns of t_1 have $o_\sigma^{\langle 1,4\rangle} = 2$ time-series variables in common (Condition (3) of Definition 7.2.1).
- ∗ There is no extended σ-pattern straddling between $[1,6]$ and $[7,10]$ (Condition (4) of Definition 7.2.1). △

Lemma 7.2.1 (not necessarily sharp upper bound for NB_σ)**.** Consider a regular expression σ, and a time series $X = \langle X_1, X_2, \ldots, X_n\rangle$, with every X_i ranging over the same integer interval domain $[\ell, u]$ such that $o_\sigma^{\langle \ell,u\rangle} \leq \omega_\sigma$.

(i) The number of σ-patterns in X is bounded by $\left\lfloor \frac{\max(0, n - o_\sigma^{\langle \ell,u\rangle})}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}} \right\rfloor$.

(ii) In addition, if $n \leq \omega_\sigma$ or there exists at least one ground time series of length n over $[\ell, u]$ that contains a single interval without restart longer than $n - \omega_\sigma - 1 + o_\sigma^{\langle \ell,u\rangle}$, then the mentioned upper bound is sharp.

Proof. Since $o_\sigma^{\langle \ell,u\rangle} \leq \omega_\sigma$ the denominator $\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}$ of the considered bound is always positive. When $n \leq \omega_\sigma$ the formula $\left\lfloor \frac{\max(0, n - o_\sigma^{\langle \ell,u\rangle})}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}} \right\rfloor$ gives 0 as the result, which is the upper bound on R. Now consider the case when $n > \omega_\sigma$.

[Proof of (i)] We construct a time series t that we prove to have the maximum number of σ-patterns among all ground time series of length n without considering any domain restrictions.

- ◇ We assume that the constructed time series t has a single interval without restart, which is longer than $n - \omega_\sigma - 1 + o_\sigma^{\langle \ell,u\rangle}$. Note that such a time series may not be feasible over $[\ell, u]$.
- ◇ By definition of an interval without restart, every pair of consecutive extended σ-patterns of t has $o_\sigma^{\langle \ell,u\rangle}$ common time-series variables. In addition, every extended σ-pattern has exactly $\omega_\sigma + 1$ time-series variables and every time-series variable whose indice is in the interval without restart belongs to at least one extended σ-pattern.
- ◇ We now prove that, for any ground time series of length n, its number of σ-patterns cannot exceed the number of σ-patterns of the constructed time series t.
  - ∗ Assume that this is not true, then there exists a ground time series whose extended σ-patterns are either strictly shorter than $\omega_\sigma + 1$ or have a number of common time-series variables strictly greater than $o_\sigma^{\langle \ell,u\rangle}$.
  - ∗ Neither of these statements can be possible by construction of t and the definitions of ω_σ and $o_\sigma^{\langle \ell,u\rangle}$.
  - ∗ Since the smallest length of an extended σ-pattern equals $\omega_\sigma + 1$, and since the number of time-series variables outside the interval without restart of t is strictly smaller than $\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}$, the time series t does not have any σ-pattern outside of its single interval without restart.
  - ∗ Hence, t has the maximum number of σ-patterns compared to all ground time series of length n.

Let us now estimate the maximum number P of potential σ-patterns in the time series t. From the construction of t we have

$$\underbrace{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle} + \omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle} + \cdots + \omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}}_{P-1 \text{ times}} + \underbrace{\omega_\sigma + 1}_{1 \text{ time}} + n_r = n, \tag{7.3}$$

where n_r is the number of time-series variables outside the interval without restart of t. From Equality 7.3 and from $n_r < \omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}$ we obtain

$$P \cdot (\omega_\sigma + 1) - (P-1) \cdot o_\sigma^{\langle \ell,u\rangle} + n_r = n \;\Rightarrow\; P = \left\lfloor \frac{n - o_\sigma^{\langle \ell,u\rangle}}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}} \right\rfloor. \tag{7.4}$$

From the right-hand side of Implication (7.4) we have that the maximum number of σ-patterns among all time series of length n over $[\ell, u]$ is less than or equal to $\left\lfloor \frac{n - o_\sigma^{\langle \ell,u \rangle}}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u \rangle}} \right\rfloor$.
[Proof of (ii)] If the time series t constructed in the first part of this proof is feasible wrt $[\ell, u]$, then the obtained bound is sharp. □

7.2.3 Extending the Upper Bound to Get a Sharp Bound Under Some Hypothesis

Consider a NB_$\sigma(X, R)$ time-series constraint with $X = \langle X_1, X_2, \ldots, X_n \rangle$, where every X_i (with $i \in [1, n]$) is over the same integer interval domain $[\ell, u]$. This section focusses on computing a *sharp* upper bound on R under some hypothesis on the regular-expression characteristics of σ.

7.2.3.1 Required Properties of Regular Expressions

Building in a greedy way a time series that maximises the number of σ-patterns requires finding a pair of words in $\mathcal{L}_\sigma$ such that the superposition of these two words wrt an integer interval domain *simultaneously optimises* several regular-expression characteristics. Depending on the value of the overlap of σ wrt $\langle \ell, u \rangle$, we have either the $\overline{\text{NB}}$-*overlapping* property when $o_\sigma^{\langle \ell,u \rangle} > 0$, introduced in Property 7.2.2, or the $\overline{\text{NB}}$-*non-overlapping* property when $o_\sigma^{\langle \ell,u \rangle} = 0$, introduced in Property 7.2.3.

- The $\overline{\text{NB}}$-overlapping property holds for $[\ell, u]$ when there exists a pair of words in $\mathcal{L}_\sigma$, whose lengths and heights are minimum, and both the overlap and the smallest variation of maxima are reached for a superposition of these words, which is not a factor of any word in $\mathcal{L}_\sigma$.
- The $\overline{\text{NB}}$-non-overlapping property holds when there exists a word in $\mathcal{L}_\sigma$, whose length and height are minimum, and this word can be a maximal occurrence of σ in the signature of a time series over $[\ell, u]$.

Property 7.2.2 ($\overline{\text{NB}}$-overlapping property)**.** A regular expression σ has the $\overline{\text{NB}}$-*overlapping* property for an integer interval domain $[\ell, u]$, if there exists a pair of not necessarily distinct words v and w in $\mathcal{L}_\sigma$, and there exists a superposition z_1 (respectively z_2) of v and w (respectively w and v) wrt $\langle \ell, u \rangle$, i.e. $o_\sigma^{\langle \ell,u \rangle} > 0$, such that the following conditions are all satisfied:

(i) $\eta_\sigma(v) = \eta_\sigma(w) = \eta_\sigma$, i.e. v and w have their heights being equal to the height of σ.

(ii) $|v| = |w| = \omega_\sigma$, i.e. v and w are shortest words in $\mathcal{L}_\sigma$.

(iii) $|v|+|w|-|z_1|+1 = |w|+|v|-|z_2|+1 = o_\sigma^{\langle \ell,u \rangle} \leq \omega_\sigma$, i.e. the overlap between v and w (respectively w and v) wrt $\langle \ell, u \rangle$ is maximum, and its value is bounded by the size of σ.

(iv) Both superpositions z_1 and z_2 are not factors of any word in $\mathcal{L}_\sigma$.

(v)
$$\delta_\sigma^{\langle \ell,u \rangle} = \begin{cases} \overline{\delta}_\sigma(z_1, v, 1) - \overline{\delta}_\sigma(z_1, w, 1) = \overline{\delta}_\sigma(z_2, w, 1) - \overline{\delta}_\sigma(z_2, v, 1) & \text{if } v \neq w, \\ \overline{\delta}_\sigma(z_1, v, 1) - \overline{\delta}_\sigma(z_1, w, 2) & \text{if } v = w, \end{cases}$$
i.e. the smallest variation of maxima of superpositions of v and w (respectively w and v) wrt $\langle \ell, u \rangle$ is reached for their superposition z_1 (respectively z_2), and is equal to the smallest variation of maxima of σ wrt $\langle \ell, u \rangle$.

(vi) $\eta_\sigma(z_1) = \eta_\sigma(z_2) = c$, where c is a constant such that
- $c = \eta_\sigma + |\delta_\sigma^{\langle \ell,u \rangle}|$ if $|\delta_\sigma^{\langle \ell,u \rangle}| > 0$, and
- $c \leq u - \ell$ if $|\delta_\sigma^{\langle \ell,u \rangle}| = 0$,

i.e. the height of each of these two superpositions z_1 and z_2 is the height of σ plus the absolute value of the smallest variation of maxima of σ wrt $\langle \ell, u \rangle$ if $\delta_\sigma^{\langle \ell,u \rangle} \neq 0$, and it is not greater than the difference between the domain maximum and minimum, otherwise.

(vii) If $\delta_\sigma^{\langle \ell,u \rangle} < 0$ (respectively $\delta_\sigma^{\langle \ell,u \rangle} > 0$), then neither '$v <$' (respectively '$v >$') nor '$w <$' (respectively '$w >$') is a factor of any word in $\mathcal{L}_\sigma$.

Every regular expression σ in Table 5.2 has the $\overline{\text{NB}}$-overlapping property for any integer interval domain $[\ell, u]$ such that $o_\sigma^{\langle \ell,u \rangle} > 0$.

Example 7.2.2 ($\overline{\text{NB}}$-overlapping property)**.** We now illustrate the $\overline{\text{NB}}$-overlapping property on three regular expressions.

- The $\sigma =$ ZIGZAG regular expression has the $\overline{\text{NB}}$-overlapping property for the integer interval domain $[\ell, u]$ such that $u - \ell \geq 2$, because there exists a pair of words $v = w =$ '$<><$' of $\mathcal{L}_\sigma$ such that the superposition of v and w wrt $\langle \ell, u \rangle$, $z =$ '$<><<><$', satisfies all the following conditions:
  - $\eta_\sigma(v) = \eta_\sigma(w) = \eta_\sigma = 1$. (Cond. (i) of Prop. 7.2.2)
  - $|v| = |w| = \omega_\sigma = 3$. (Cond. (ii) of Prop. 7.2.2)
  - $|v| + |w| - |z| + 1 = o_\sigma^{\langle \ell,u \rangle} = 1 \leq \omega_\sigma = 3$. (Cond. (iii) of Prop. 7.2.2)
  - Since no word of $\mathcal{L}_\sigma$ contains two consecutive '$<$', z is not a factor for any word in $\mathcal{L}_\sigma$. (Cond. (iv) of Prop. 7.2.2)
  - $\delta_\sigma^{\langle \ell,u \rangle} = 0$. (Cond. (v, vii) of Prop. 7.2.2)
  - The height of z is 2, which is less than or equal to $u - \ell$. (Cond. (vi) of Prop. 7.2.2)
- The $\sigma =$ DECREASING_TERRACE regular expression has the $\overline{\text{NB}}$-overlapping property for the integer interval domain $[\ell, u]$ such that $u - \ell \geq 3$, because there exists a pair of words $v = w =$ '$>=>$' in $\mathcal{L}_\sigma$ and their superposition $z =$ '$>=>=>$' wrt $\langle \ell, u \rangle$, such that all the following conditions are satisfied:
  - $\eta_\sigma(v) = \eta_\sigma(w) = \eta_\sigma = 2$. (Cond. (i) of Prop. 7.2.2)
  - $|v| = |w| = \omega_\sigma = 3$. (Cond. (ii) of Prop. 7.2.2)
  - $|v| + |w| - |z| + 1 = o_\sigma^{\langle \ell,u \rangle} = 2 \leq \omega_\sigma = 3$. (Cond. (iii) of Prop. 7.2.2)
  - Since any word in $\mathcal{L}_\sigma$ contains only consecutive equalities, the word z is not a factor of any word in $\mathcal{L}_\sigma$. (Cond. (iv) of Prop. 7.2.2)
  - $\delta_\sigma^{\langle \ell,u \rangle} = \overline{\delta}_\sigma^{\langle \ell,u \rangle}(z, v, 1) - \overline{\delta}_\sigma^{\langle \ell,u \rangle}(z, w, 2) = -1$. (Cond. (v) of Prop. 7.2.2)
  - The height of z is 3, which equals $\eta_\sigma + |\delta_\sigma^{\langle \ell,u \rangle}|$. (Cond. (vi) of Prop. 7.2.2)
  - No word in $\mathcal{L}_\sigma$ has '$<$', thus '$v <$' is not a factor of any word in $\mathcal{L}_\sigma$. (Cond. (vii) of Prop. 7.2.2)
- The $\sigma =$ STEADY_SEQUENCE regular expression does not have the $\overline{\text{NB}}$-overlapping property for any integer interval domain $[\ell, u]$, because for any pair of words v, w in $\mathcal{L}_\sigma$, the set of superpositions of v and w wrt $\langle \ell, u \rangle$ is empty, and thus $o_\sigma^{\langle \ell,u \rangle} = 0$. △

Property 7.2.3 ($\overline{\text{NB}}$-non-overlapping property)**.** A regular expression σ has the $\overline{\text{NB}}$-*non-overlapping* property for an integer interval domain $[\ell, u]$, if $o_\sigma^{\langle \ell,u \rangle} = 0$ and if there exists a word v in $\mathcal{L}_\sigma$ such that all the following conditions are satisfied:

(i) $|v| = \omega_\sigma$, i.e. v is a shortest word in $\mathcal{L}_\sigma$.

(ii) $\eta_\sigma(v) = \eta_\sigma$, i.e. v has a minimum height among all words in $\mathcal{L}_\sigma$.

(iii) Either both words '$v >$' and '$v <$' are not factors of any word in $\mathcal{L}_\sigma$, or at least one of the three words $\{$'$v > v$', '$v < v$', '$v = v$'$\}$ is not a factor of any word in $\mathcal{L}_\sigma$, and its height is equal to η_σ.

(iv) For any integer $n > \omega_\sigma$, there exists at least one ground time series of length n over $[\ell, u]$, whose signature contains v as a maximal occurrence of σ.

Any regular expression σ in Table 5.2 has the $\overline{\text{NB}}$-non-overlapping property for any integer interval domain $[\ell, u]$ such that $o_\sigma^{\langle \ell,u \rangle} = 0$, except the STEADY_SEQUENCE regular expression for $[\ell, u]$ such that $\ell = u$. The case of STEADY_SEQUENCE when $\ell = u$ is discussed in Example 7.2.3.

Example 7.2.3 ($\overline{\text{NB}}$-non-overlapping property)**.** We illustrate the $\overline{\text{NB}}$-non-overlapping property on three regular expressions.

- The σ = ZIGZAG regular expression has the $\overline{\text{NB}}$-non-overlapping property for any integer interval domain $[\ell, u]$ such that $u - \ell = 1$ because (1) as shown in Example 7.1.5, for any two words of $\mathcal{L}_\sigma$, the set of their superpositions wrt $\langle \ell, u \rangle$ is empty, and (2) there exists a word $v =$ '><>' in $\mathcal{L}_\sigma$ that satisfies all the following conditions:
  - $|v| = \omega_\sigma = 3$. (Cond. (i) of Prop. 7.2.3)
  - $\eta_\sigma(v) = \eta_\sigma = 1$. (Cond. (ii) of Prop. 7.2.3)
  - No word of $\mathcal{L}_\sigma$ contains '=', hence '$v = v$' is not a factor of any word in $\mathcal{L}_\sigma$. Furthermore the height of v is equal to η_σ. (Cond. (iii) of Prop. 7.2.3)
  - For any integer $n > \omega_\sigma$, there exists a ground time series of length n over $[\ell, u]$ whose signature contains v as a maximal occurrence of σ. (Cond. (iv) of Prop. 7.2.3)
- The σ = DECREASING_TERRACE regular expression has the $\overline{\text{NB}}$-non-overlapping property for any integer interval domain $[\ell, u]$ such that $u - \ell = 2$ because (1) as shown in Example 7.1.5, for any two words of $\mathcal{L}_\sigma$, the set of their superpositions wrt $\langle \ell, u \rangle$ is empty, and (2) there exists a word $v =$ '>=>' in $\mathcal{L}_\sigma$ that satisfies all the following conditions:
  - $|v| = \omega_\sigma = 3$. (Cond. (i) of Prop. 7.2.3)
  - $\eta_\sigma(v) = \eta_\sigma = 2$. (Cond. (ii) of Prop. 7.2.3)
  - '$v < v$' is not a factor of any word in $\mathcal{L}_\sigma$, and its height is 2. (Cond. (iii) of Prop. 7.2.3)
  - For any integer $n > \omega_\sigma$, there exists a ground time series of length n over $[\ell, u]$ whose signature contains v as a maximal occurrence of σ. (Cond. (iv) of Prop. 7.2.3)
- Consider the σ = STEADY_SEQUENCE regular expression.
  - First, σ does not have the $\overline{\text{NB}}$-non-overlapping property for an integer interval domain $[\ell, u]$ such that $u - \ell = 0$, since Condition (iv) of Property 7.2.3 is violated: the shortest word of $\mathcal{L}_\sigma$, namely $v =$ '=' cannot be a maximal occurrence of σ in the signature of any ground time series longer than 2 over $[\ell, u]$; indeed, for any time-series length, there exists a single ground time series with all equal values, thus its signature contains only equalities, which prevents v to be a maximal occurrence of σ.
  - Second, σ has the $\overline{\text{NB}}$-non-overlapping property for an integer interval domain $[\ell, u]$ such that $u - \ell > 0$ because there exists a word $v =$ '=' in $\mathcal{L}_\sigma$ that satisfies all the following conditions:
    - $|v| = \omega_\sigma = 1$. (Cond. (i) of Prop. 7.2.3)
    - $\eta_\sigma(v) = \eta_\sigma = 0$. (Cond. (ii) of Prop. 7.2.3)
    - No word of $\mathcal{L}_\sigma$ contains '>' or '<', hence neither '$v >$', nor '$v <$' are factors of any word in $\mathcal{L}_\sigma$. (Cond. (iii) of Prop. 7.2.3)
    - For any integer $n > \omega_\sigma$, there exists a ground time series of length n over $[\ell, u]$ whose signature contains v as a maximal occurrence of σ. (Cond. (iv) of Prop. 7.2.3)

△

7.2.3.2 Structure of a Maximal Time Series

Consider a NB_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with every X_i having the same integer interval domain $[\ell, u]$. Lemma 7.2.2 describes the structure of a maximal time series for NB_$\sigma(\langle X_1, \ldots, X_n \rangle, R)$ under the hypothesis that σ has either the $\overline{\text{NB}}$-overlapping or the $\overline{\text{NB}}$-non-overlapping property for $[\ell, u]$.

Lemma 7.2.2 (structure of a maximal time series)**.** Consider a regular expression σ that has either the $\overline{\text{NB}}$-overlapping or the $\overline{\text{NB}}$-non-overlapping property for an integer interval domain $[\ell, u]$. Then for any integer number $n > \omega_\sigma$, there exists a word q such that any ground time series t of length n over $[\ell, u]$ whose signature contains q has the maximum number of σ-patterns among all ground time series of length n over $[\ell, u]$.

Proof We first construct a word q and we show that there is at least one time series of length n over $[\ell, u]$ whose signature contains q. Then, we prove that any time series t of length n over $[\ell, u]$ whose signature contains q is maximal for the NB_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with every X_i ranging over $[\ell, u]$.

Case (1): *σ has the $\overline{\text{NB}}$-overlapping property for $[\ell, u]$.*
Then there exist two words v and w of $\mathcal{L}_\sigma$ and a superposition z_1 (respectively z_2) of v and w (respectively w and v) wrt $\langle \ell, u \rangle$ such that all the six conditions of Property 7.2.2 are satisfied. Let w_1 and w_2 be the words such that $z_1 = vw_2$ and $z_2 = ww_1$. Figure 7.2 shows the relations between the words z_1, z_2, v, w, w_1, and w_2.

- **Step 1**: *Construction of the word q.*
  When constructing the word q we consider two cases.
  - **Case (1.1)**: *The smallest variation of maxima $\delta_\sigma^{\langle \ell,u \rangle}$ equals zero.*
    In this case, the time series t has a single interval without restart that contains all σ-patterns of t. We construct the signature q of this interval without restart by imposing the following conditions:

    (a) The word q is in the language of the '$v \mid z_1(w_1w_2)^*(w_1 \mid \varepsilon)$' regular expression.

    (b) The length of q is less than n.

    (c) The length of q is maximum among all words satisfying Conditions (a), and (b).

    By condition (i) of Property 7.2.2, the heights of both v and w equal η_σ, the height of σ. Since $\delta_\sigma^{\langle \ell,u \rangle} = 0$, by Condition (vi) of Property 7.2.2, the height of both words z_1 and z_2 is not greater than $u - \ell$. Furthermore, by Condition (v) of Property 7.2.2 and by Corollary 7.1.1, describing the structure of words v and w, it can be shown that the height of q is also not greater than $u - \ell$, thus q indeed appears in the signature of some ground time series of length n over $[\ell, u]$, and t is feasible.

  - **Case (1.2)**: *The smallest variation of maxima $\delta_\sigma^{\langle \ell,u \rangle}$ does not equal zero.*
    For brevity, we consider only the case when $\delta_\sigma^{\langle \ell,u \rangle} > 0$, the case of a negative $\delta_\sigma^{\langle \ell,u \rangle}$ being symmetric. The time series t may have $p \geq 1$ intervals without restart, hence in order to construct q we first construct the signature b of every, except possibly the last one, interval without restart of t by imposing the following conditions:

    (d) The word b is in the language of the '$v \mid z_1(w_1w_2)^*(w_1 \mid \varepsilon)$' regular expression.

    (e) The set of supporting time series of b wrt $\langle \ell, u \rangle$ is not empty.

    (f) The length of b is less than n.

    (g) The length of b is maximum among all words satisfying Conditions (d), (e) and (f).

    Note that b always exists, since there is at least one word, namely v, satisfying Conditions (d), (e) and (f). Then, the word q must satisfy the following conditions:

    (h) The word q belongs to the language of the '$(b >)^*\tilde{q}_{rest}$' regular expression, where $\tilde{q}_{rest}$ is a word in the language of the '$v \mid z_1(w_1w_2)^*(w_1 \mid \varepsilon)$' regular expression such that $|\tilde{q}_{rest}| \leq |b|$.

    (i) The length of q is less than n.

    (j) The length of q is maximum among all words satisfying Conditions (h) and (i).

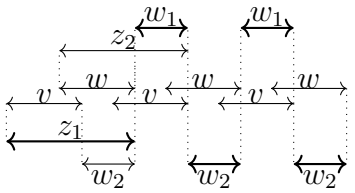


Figure 7.2 – Lemma 7.2.2 Case (1.1): Illustration of the word $z_1w_1w_2w_1w_2$ belonging to the language of '$v \mid z_1(w_1w_2)^*(w_1 \mid \varepsilon)$'

Since $\delta_\sigma^{\langle \ell,u \rangle} > 0$, by Lemma 7.1.1, and by construction of b, the word b does not contain any '$>$'. Then, the concatenation of b and '$>$' has the same height as b. Hence, the height of q equals the height of b, whose set of supporting time series wrt $\langle \ell, u \rangle$ is not empty, thus q indeed appears in the signature of some ground time series of length n over $[\ell, u]$, and t is feasible.

- **Step 2**: *Maximality of any time series t whose signature contains q.*
  We now prove that t is a maximal time series for NB_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$.
  - First, we show that the number p of σ-patterns of t equals the number of occurrences of the words v and w in its signature. By Conditions (iv) and (vii) of Property 7.2.2, the words v and w appearing in q cannot be factors of any other occurrence of σ in q, hence p is not less than the number of occurrences of the words v and w in q. By Conditions (iii) of Property 7.2.2, no extended σ-pattern can straddle between two other extended σ-patterns. In addition, by the maximality of the length of q there is no occurrence of σ in the part of the signature of t that is the complement of q. Hence, neither is p greater than the number of occurrences of the words v and w in q, and thus these values are equal.
  - Second, we prove that t is maximal for NB_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$. Suppose that t is not maximal for NB_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ and there exists a time series t' of length n over $[\ell, u]$ that has a number of σ-patterns strictly greater than t. Then at least one of the following conditions must be satisfied:

    (k) There is a smaller number of intervals without restart of the same total length.

    (l) Some extended σ-patterns of such a time series are of length shorter than $\omega_\sigma + 1$.

    (m) Some pairs of consecutive extended σ-patterns have more than $o_\sigma^{\langle \ell,u \rangle}$ time-series variables in common.

    (n) There is an extended σ-pattern that straddles between two other extended σ-patterns.

    Assumption (k) contradicts Condition (v) of Property 7.2.2 and the construction of the signature of intervals without restart. Assumptions (l) and (m) contradict Conditions (ii) and (iii) of Property 7.2.2. Finally, Assumption (n) is not possible because of the bound imposed on the value of the overlap in Condition (iii) of Property 7.2.2. Hence, t has the maximum number of σ-patterns among all ground time series of the same length over $[\ell, u]$.

Case (2): *σ has the $\overline{\text{NB}}$-non-overlapping property for $[\ell, u]$.*
There exists a word v such that all the conditions of Property 7.2.3 are satisfied. The construction of q is similar to Case (1), but the word q will always be the signature of a single interval without restart. The word q is built using the following rules:

(o) If both words '$v >$' and '$v <$' are not proper factors of any word in $\mathcal{L}_\sigma$, then q is a word in the language of the '$(v > v <)^* v$' regular expression.

(p) If at least one word w in {'$v >$', '$v =$', '$v <$'} is not a proper factor of any word in $\mathcal{L}_\sigma$, and its height equals η_σ, then q is in the language of the '$w^* v$' regular expression.

(q) The length of q is less than n.

(r) The length of q is maximum among all words satisfying Conditions (o), (p), and (q).

Since all the conditions of Property 7.2.3 are satisfied, it can be shown that the height of q is not greater than $u - \ell$, and thus at least one time series of length n over $[\ell, u]$ contains q in its signature. Then, in a similar fashion as in Case (1), one can prove that any time series whose signature contains q is maximal for NB_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$. □

7.2.3.3 A Sharp Upper Bound on the Number of Occurrences of Regular Expression

Consider a NB_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[\ell, u]$. First, Lemma 7.2.3 gives an upper bound on the maximum length of an

interval without restart in a time series over $[\ell, u]$. Second, based on this upper bound and the structure of a maximal time series for NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ showed in Lemma 7.2.2, Theorem 7.2.2 provides a sharp upper bound on R under some hypothesis on the regular-expression characteristics of σ.

Lemma 7.2.3 (maximum length of an interval without restart)**.** Consider a regular expression σ and an integer interval domain $[\ell, u]$ such that one of the following conditions is satisfied:

(i) The value of $\delta_\sigma^{\langle \ell,u\rangle}$ equals zero.

(ii) The value of $\delta_\sigma^{\langle \ell,u\rangle}$ does not equal zero and σ has the $\overline{\text{NB}}$-overlapping property.

Then, the *maximum length of an interval without restart* of any ground time series over $[\ell, u]$ is bounded by

$$m_\sigma^{\langle \ell,u\rangle} = \begin{cases} \left\lfloor \frac{u-\ell-\eta_\sigma+|\delta_\sigma^{\langle \ell,u\rangle}|}{|\delta_\sigma^{\langle \ell,u\rangle}|} \right\rfloor \cdot \left(\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}\right) + o_\sigma^{\langle \ell,u\rangle} & \text{if } \delta_\sigma^{\langle \ell,u\rangle} \neq 0, \\ +\infty, & \text{otherwise .} \end{cases}$$

Proof. **Case (1)**: *Condition (i) is satisfied.*

Since $\delta_\sigma^{\langle \ell,u\rangle} = 0$, the condition that the maximum length of an interval without restart is bounded by $+\infty$ is trivially satisfied. This upper bound reflects the fact that when $\delta_\sigma^{\langle \ell,u\rangle} = 0$, the maximum length of an interval without restart does not depend on the domain $[\ell, u]$.

Case (2): *Condition (ii) is satisfied.*

Now consider the case when $\delta_\sigma^{\langle \ell,u\rangle} \neq 0$ and σ has the $\overline{\text{NB}}$-overlapping property. Let b be a word such that (1) b is the signature of an interval without restart of maximum length constructed in Lemma 7.2.2 for a time series of some length n over $[\ell, u]$; (2) for any time series of length $n' > n$ over $[\ell, u]$, b is also the signature of an interval without restart of maximum length. Note that such b necessarily exists as the set of supporting time series of b wrt $\langle \ell, u\rangle$ must not be empty. Then, there exists a ground time series t of length n over $[\ell, u]$ whose signature is b. By construction of b, the maximum of every extended σ-pattern of t, except the first one, is $|\delta_\sigma^{\langle \ell,u\rangle}|$ units smaller or greater, depending on the sign of $\delta_\sigma^{\langle \ell,u\rangle}$, compared to the maximum of the preceding extended σ-pattern. Thus, the maxima of these extended σ-patterns form a monotonously decreasing (respectively increasing) sequence of integer numbers. By Conditions (i), (iii), (iv) and (v) of Property 7.2.2, the number of elements of such a sequence is bounded by $\left\lfloor \frac{u-\ell-\eta_\sigma+|\delta_\sigma^{\langle \ell,u\rangle}|}{|\delta_\sigma^{\langle \ell,u\rangle}|} \right\rfloor$. Since every extended σ-pattern is of length $\omega_\sigma + 1$, has a height η_σ, and the number of common time-series variable between two extended σ-patterns equals $o_\sigma^{\langle \ell,u\rangle}$, the value $\left\lfloor \frac{u-\ell-\eta_\sigma+|\delta_\sigma^{\langle \ell,u\rangle}|}{|\delta_\sigma^{\langle \ell,u\rangle}|} \right\rfloor \cdot (\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}) + o_\sigma^{\langle \ell,u\rangle}$ is the maximum length of an interval without restart of a ground time series among all ground time series over $[\ell, u]$. □

Theorem 7.2.2 (sharp upper bound for NB_σ)**.** Consider a NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[\ell, u]$. If σ has either the $\overline{\text{NB}}$-overlapping or the $\overline{\text{NB}}$-non-overlapping properties for $[\ell, u]$, then a sharp upper bound on R is

$$\underbrace{\left\lfloor \frac{\max(0, m - o_\sigma^{\langle \ell,u\rangle})}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}} \right\rfloor}_{A} \cdot \underbrace{\left\lfloor \frac{n}{m} \right\rfloor}_{B} + \underbrace{\left\lfloor \frac{\max(0, (n \bmod m) - o_\sigma^{\langle \ell,u\rangle})}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}} \right\rfloor}_{C}, \tag{7.5}$$

where:

- $m = \min(n, \max(1, m_\sigma^{\langle \ell,u\rangle}))$, where $m_\sigma^{\langle \ell,u\rangle}$ is the upper bound on the maximum length of an interval without restart in a time series over $[\ell, u]$, introduced by Lemma 7.2.3.
- A is the maximum number of σ-patterns in an interval without restart of maximum length.
- B is the number of intervals without restart of maximum length in a maximal time series for the NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint.

- C is the maximum number of σ-patterns in an interval without restart of non-maximum length in a maximal time series for NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$.

Proof. Lemma 7.2.2 showed the existence of a word q such that any time series t of length n over $[\ell, u]$ whose signature contains q is maximal for NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$. Hence, a sharp upper bound on R can be obtained by counting the number of occurrences of σ in q.

Case (1): $m_\sigma^{\langle \ell,u\rangle} \geq n - \omega_\sigma + o_\sigma^{\langle \ell,u\rangle}$. Then, t contains a single interval without restart longer than $n - \omega_\sigma + o_\sigma^{\langle \ell,u\rangle}$. Further, the value of $\min(n, \max(1, m_\sigma^{\langle \ell,u\rangle}))$ equals n, and the components B and C become respectively equal to 1 and 0, thus Formula (7.5) simplifies to A. By Lemma 7.2.1, the obtained value is a sharp upper bound on R.

Case (2): $m_\sigma^{\langle \ell,u\rangle} < n - \omega_\sigma + o_\sigma^{\langle \ell,u\rangle}$. Then t may contain multiple intervals without restart. Furthermore, the length of all intervals without restart of t, except maybe the last one, equals $m_\sigma^{[\ell,u]}$. By Lemma 7.2.1, the maximum number of σ-patterns within every interval without restart of maximum length is $\left\lfloor \frac{\max(0, m - o_\sigma^{\langle \ell,u\rangle})}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}} \right\rfloor$, i.e. the term A. The number of intervals without restart of maximum length is $\lfloor \frac{n}{m} \rfloor$, i.e. the term B. The last interval without restart of t may be shorter than $m_\sigma^{\langle \ell,u\rangle}$, then its length is computed as $n \bmod m$, and the number of σ-patterns in the last interval without restart is computed as $\left\lfloor \frac{\max(0, (n \bmod m) - o_\sigma^{\langle \ell,u\rangle})}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}} \right\rfloor$, which is C. □

Example 7.2.4 (sharp upper bound for NB_σ)**.** Consider a NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[\ell, u]$.

- Let σ be the ZIGZAG regular expression.
  - First, assume that $u - \ell = 1$, and recall some of the computed regular-expression characteristics, namely $o_\sigma^{\langle \ell,u\rangle} = 0$, $\omega_\sigma = 3$ and $\delta_\sigma^{\langle \ell,u\rangle} = 0$. It was shown in Example 7.2.3 that σ has the $\overline{\text{NB}}$-non-overlapping property for $[\ell, u]$, thus Theorem 7.2.2 can be applied for computing a sharp upper bound on R. Since $\delta_\sigma^{\langle \ell,u\rangle}$ is 0, by Lemma 7.2.3, $m_\sigma^{\langle \ell,u\rangle} = +\infty$, and thus a sharp upper bound on R is $\left\lfloor \frac{\max(0, \min(n, \max(1, m_\sigma^{\langle \ell,u\rangle})) - o_\sigma^{\langle \ell,u\rangle})}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}} \right\rfloor = \left\lfloor \frac{\max(0, \min(n, +\infty) - 0)}{3 + 1 - 0} \right\rfloor = \lfloor \frac{n}{4} \rfloor$.
  - Second, assume $u - \ell \geq 2$, then $o_\sigma^{\langle \ell,u\rangle}$ is now equal to 1, and $\delta_\sigma^{\langle \ell,u\rangle}$ is equal to 0. It was shown in Example 7.2.2 that σ has the $\overline{\text{NB}}$-overlapping property for $[\ell, u]$, thus Theorem 7.2.2 can be applied for computing a sharp upper bound on R, and a sharp upper bound on R is equal to $\left\lfloor \frac{\max(0, \min(n, +\infty) - 1)}{3 + 1 - 1} \right\rfloor = \lfloor \frac{n-1}{3} \rfloor$.
- Let σ be the DECREASING_TERRACE regular expression.
  - First, assume that $u - \ell = 2$, and recall some of the computed regular-expression characteristics, namely $o_\sigma^{\langle \ell,u\rangle} = 0$, $\omega_\sigma = 3$ and $\delta_\sigma^{\langle \ell,u\rangle} = 0$. It was shown in Example 7.2.3 that σ has the $\overline{\text{NB}}$-non-overlapping property for $[\ell, u]$, thus Theorem 7.2.2 can be applied for computing a sharp upper bound on R. By Lemma 7.2.3, we have that $m_\sigma^{\langle \ell,u\rangle} = +\infty$, and thus a sharp upper bound on R is $\left\lfloor \frac{\max(0, \min(n, \max(1, m_\sigma^{\langle \ell,u\rangle})) - o_\sigma^{\langle \ell,u\rangle})}{\omega_\sigma + 1 - o_\sigma^{\langle \ell,u\rangle}} \right\rfloor = \left\lfloor \frac{\max(0, \min(n, \max(1, +\infty)) - 0)}{3 + 1 - 0} \right\rfloor = \lfloor \frac{n}{4} \rfloor$.
  - Second, assume $u - \ell \geq 3$, then $o_\sigma^{\langle \ell,u\rangle}$ is now equal to 2, and $\delta_\sigma^{\langle \ell,u\rangle}$ is equal to -1. It was shown in Example 7.2.2 that σ has the $\overline{\text{NB}}$-overlapping property for $[\ell, u]$, thus Theorem 7.2.2 can be applied for computing a sharp upper bound on R, and a sharp upper bound on R is equal to $\left\lfloor \frac{\max(0, m-2)}{2} \right\rfloor \cdot \lfloor \frac{n}{m} \rfloor + \left\lfloor \frac{\max(0, (n \bmod m) - 2)}{2} \right\rfloor$ where $m = \min(n, \max(1, m_\sigma^{\langle \ell,u\rangle})) = \min(n, \max(1, (u - \ell - 1)) \cdot 2 + 2)$, computed by using Lemma 7.2.3. △

All the 22 regular expression in Table 5.2 have either the $\overline{\text{NB}}$-overlapping or the $\overline{\text{NB}}$-non-overlapping property for any integer interval domain $[\ell, u]$, except the STEADY_SEQUENCE regular expression when $\ell = u$. A sharp upper bound on the result variable of a time-series constraint in this case is given in Proposition 7.2.3.

7.2.3.4 A Sharp Upper Bound: Special Case for STEADY_SEQUENCE

Proposition 7.2.3 provides a sharp upper bound on the number of occurrences of the STEADY_SEQUENCE regular expression in the signature of a time series over an integer interval domain $[\ell, u]$ such that $\ell = u$.

Proposition 7.2.3 (sharp upper bound for NB_STEADY_SEQUENCE)**.** Consider a NB_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint with σ being the STEADY_SEQUENCE regular expression and with every X_i ranging over the same integer interval domain $[\ell, u]$ such that $\ell = u$. A sharp upper bound on R equals 1.

Proof. Since $\ell = u$, there exists a single time series of length n over $[\ell, u]$, and all its time-series variables have the same value, namely ℓ. The entire signature of this time series is a word in $\mathcal{L}_\sigma$, thus a sharp upper bound on R equals 1. □

7.3 Time-Series Constraints with Feature WIDTH

We now consider the g_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ family of time-series constraints with every X_i ranging over the same integer interval domain $[\ell, u]$, i.e. the case when the feature is `width`, g is in the set $\{\texttt{max}, \texttt{min}, \texttt{sum}\}$ and σ is a non-fixed length regular expression. Section 7.3.1 defines Properties 7.3.1 and 7.3.2 of regular expressions that we use to obtain sharp upper bounds on R. All the regular expressions in Table 5.2 have both Properties 7.3.1 and 7.3.2. Based on these properties, Section 7.3.2 (respectively Section 7.3.3) provides a sharp upper bound on R when g is `max` (respectively `sum`). Finally, Section 7.3.4 gives a sharp lower bound on R when g is `sum`. Note that we do not consider a lower (respectively upper) bound for the case when the aggregator is `max` (respectively `min`), since when σ has the NB-simple property (see Property 7.2.1) for $[\ell, u]$, there exists a time series of length n over $[\ell, u]$ that has no σ-patterns, and thus yields the identity value of the aggregator, namely 0 (respectively $n + 1$). Among the 22 regular expressions in Table 5.2 only the STEADY and the STEADY_SEQUENCE regular expressions do not have the NB-simple property for a domain with a single element, i.e. $\ell = u$.

7.3.1 Properties of Regular Expressions

Property 7.3.1 is used for deriving a sharp upper bound on R for a MAX_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint. Property 7.3.1 requires the range of a regular expression be a monotonically increasing linear function of n.

Property 7.3.1 ($\overline{\text{WIDTH}}$-monotonous property)**.** A regular expression σ has the $\overline{\text{WIDTH}}$-*monotonous* property if the following conditions are all satisfied:

(i) There exists a shortest word in $\mathcal{L}_\sigma$ whose height equals η_σ, the height of σ.

(ii) For every time-series length $n > \omega_\sigma + 1$, the range of σ wrt $\langle n\rangle$, $\phi_\sigma^{\langle n\rangle}$, is defined and equals $e_\sigma \cdot (n - 1 - \eta_\sigma) + c_\sigma + \eta_\sigma$ with $\langle e_\sigma, c_\sigma\rangle \in \{\langle 0, 0\rangle, \langle 0, 1\rangle, \langle 1, 0\rangle\}$.

Property 7.3.2 is used for deriving a sharp upper bound on R for a SUM_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint.

Property 7.3.2 ($\overline{\text{WIDTH}}$-sum property)**.** A regular expression σ has the $\overline{\text{WIDTH}}$-*sum* property for an integer interval domain $[\ell, u]$ if the following conditions are all satisfied:

(i) $o_\sigma^{\langle \ell, u\rangle} \leq a_\sigma + b_\sigma$.

(ii) If for every time-series length $n > \omega_\sigma + 1$, the range of σ wrt $\langle n\rangle$, $\phi_\sigma^{\langle n\rangle}$, equals $n - 1$, then a_σ, b_σ and $o_\sigma^{\langle \ell, u\rangle}$ are all equal to 0, and ω_σ, the size of σ, is equal to 1.

Condition (i) of Property 7.3.2 withdraws from consideration a regular expression σ whose σ-patterns overlap, i.e. some time-series variables belong simultaneously to two σ-patterns, which will be formalised in Lemma 7.3.1. Condition (ii) of Property 7.3.2 restricts further a class of regular expressions whose range depends linearly on n.

7.3.2 Upper Bound for MAX_WIDTH_σ

We first consider the case when the aggregator is max, i.e. the MAX_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ family of time-series constraints with σ being a non-fixed length regular expression and every X_i ranging over the same integer interval domain $[\ell, u]$. To compute a sharp upper bound on R, we maximise the width of a σ-pattern in $X = \langle X_1, X_2, \ldots, X_n\rangle$. We do so by detecting a longest word in $\mathcal{L}_\sigma$ that may appear in the signature of X. The transition from the length of a σ-pattern to the length of the corresponding word in $\mathcal{L}_\sigma$ is sound because the width of the σ-pattern is the length of the corresponding word plus 1 and minus the sum of a_σ and b_σ, which are constant parameters of σ, introduced in Table 5.2.

A trivial but, possibly not sharp upper bound on R is $n - a_\sigma - b_\sigma$, where b_σ and a_σ are parameters of regular expression used for trimming the left and right borders of the regular expression as introduced in Section 5.1. Further, for regular expressions that have the $\overline{\text{WIDTH}}$-monotonous property, we show that the sharpness of the mentioned upper bound depends only on the difference between u and ℓ.

The idea for computing a sharp upper bound on R when σ has the $\overline{\text{WIDTH}}$-monotonous property is to identify the minimum value d of $u - \ell$ such that the bound $n - a_\sigma - b_\sigma$ is still sharp. When $u - \ell$ is smaller than d we need to find the maximum value of $k < n$, such that $k - a_\sigma - b_\sigma$ is a sharp upper bound on R for a MAX_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_k\rangle, R)$ time-series constraint with every X_i ranging over $[\ell, u]$.

The next theorem provides a sharp upper bound on R when the regular expression σ has the $\overline{\text{WIDTH}}$-monotonous property.

Theorem 7.3.1 (sharp upper bound for MAX_WIDTH_σ)**.** Consider a MAX_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint with σ being a non-fixed length regular expression, and all X_i ranging over the same integer interval domain $[\ell, u]$. If σ has the $\overline{\text{WIDTH}}$-monotonous property, then a sharp upper bound on R is

$$\begin{cases} n - a_\sigma - b_\sigma & \text{if } u - \ell \geq \phi_\sigma^{\langle n\rangle}, \quad (7.1) \\ e_\sigma \cdot (u - \ell + 1 - a_\sigma - b_\sigma) + c_\sigma \cdot (\omega_\sigma + 1 - a_\sigma - b_\sigma) & \text{if } u - \ell < \phi_\sigma^{\langle n\rangle}, \quad (7.2) \end{cases}$$

where e_σ and c_σ are parameters of the regular expression σ, introduced in Property 7.3.1.

Proof. When the regular expression σ has the $\overline{\text{WIDTH}}$-monotonous property, the range $\phi_\sigma^{\langle n\rangle}$ of σ wrt $\langle n\rangle$ is a monotonically increasing function of n. It implies that, if the upper bound $n - a_\sigma - b_\sigma$ is sharp for some interval integer domain $[\ell_1, u_1]$, then it is also sharp for any interval integer domain $[\ell_2, u_2]$ such that $u_2 - \ell_2 > u_1 - \ell_1$. Hence, the sharpness of the upper bound $n - a_\sigma - b_\sigma$ depends only on $u - \ell$.
[Case (7.1): $u - \ell \geq \phi_\sigma^{\langle n\rangle}$]. By definition of $\phi_\sigma^{\langle n\rangle}$, we have that if $u - \ell \geq \phi_\sigma^{\langle n\rangle}$, then there exists a word in $\mathcal{L}_\sigma$ of length $n - 1$ whose height is not greater than $u - \ell$. Hence, $n - a_\sigma - b_\sigma$ is a sharp upper bound on R.
[Case (7.2): $u - \ell < \phi_\sigma^{\langle n\rangle}$]. This case requires a more detailed analysis than Case (7.1). Let us consider the three distinct pairs of $\langle e_\sigma, c_\sigma\rangle$ from Condition (ii) of Property 7.3.1:

(a) The case of $\langle e_\sigma, c_\sigma\rangle$ being $\langle 0, 0\rangle$. Since $u - \ell < 0 \cdot (n - 1 - \eta_\sigma) + 0 + \eta_\sigma = \eta_\sigma$, the necessary-sufficient condition, i.e. Property 7.1.1, is not satisfied, thus R is equal to the identity value of the aggregator, namely 0.

(b) The case of $\langle e_\sigma, c_\sigma\rangle$ being $\langle 0, 1\rangle$. Since $u - \ell < 0 \cdot (n - 1 - \eta_\sigma) + 1 + \eta_\sigma = \eta_\sigma + 1$, the only words in $\mathcal{L}_\sigma$ that can appear in the signature of a ground time series over $[\ell, u]$ are the ones with the minimum height, namely η_σ. For every time-series length $n > \omega_\sigma + 1$, we have that $\phi_\sigma^{\langle n\rangle} = \eta_\sigma + 1$, which implies that for every word in $\mathcal{L}_\sigma$ of length strictly greater than ω_σ, the height is at least $\eta_\sigma + 1$. Hence, only a word of length ω_σ and of height η_σ can be an occurrence of σ in the signature of a ground time series

over $[\ell, u]$. By Condition (i) of Property 7.3.1, such a word exists in $\mathcal{L}_\sigma$ and thus, a sharp upper bound on R is $\omega_\sigma + 1 - a_\sigma - b_\sigma$.

(c) The case of $\langle e_\sigma, c_\sigma \rangle$ being $\langle 1, 0 \rangle$. Since $u - \ell < 1 \cdot (n-1-\eta_\sigma) + 0 + \eta_\sigma = n-1$, we have that $u - \ell < n-1$. Hence, we aim at finding the longest time-series length $k < n$ such that $u - \ell = k - 1$, and a sharp upper bound on R will be $k - a_\sigma - b_\sigma$. The largest value of such k equals $u - \ell + 1$, thus a sharp upper bound on R is $u - \ell + 1 - a_\sigma - b_\sigma$.

□

Example 7.3.1 (sharp upper bound for MAX_WIDTH_σ)**.** Consider a MAX_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with every X_i having the same integer interval domain $[\ell, u]$. The three items of this example cover each value of $\langle e_\sigma, c_\sigma \rangle$ in the set $\{\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle\}$.

- Consider the $\sigma =$ INFLEXION regular expression. Recall that both a_σ and b_σ are equal to 1, the size ω_σ of σ is equal to 2, the height η_σ of σ is equal to 1, and for any time-series length $n > \omega_\sigma + 1$, the range $\phi_\sigma^{\langle n \rangle}$ of σ wrt $\langle n \rangle$ is equal to $e_\sigma \cdot (n - 1 - \eta_\sigma) + c_\sigma + \eta_\sigma = \eta_\sigma = 1$. Since there exists a word, namely $v =$ '<>', in $\mathcal{L}_\sigma$ whose length equals 2 and whose height is equal to 1, and $\langle e_\sigma, c_\sigma \rangle$ is $\langle 0, 0 \rangle$, σ has the $\overline{\text{WIDTH}}$-monotonous property. Hence, we apply Theorem 7.3.1 for computing a sharp upper bound on R.
  - If $u - \ell \geq \phi_\sigma^{\langle n \rangle} = 1$, then a sharp upper bound on R is equal to $n - a_\sigma - b_\sigma = n - 2$.
  - If $u - \ell < \phi_\sigma^{\langle n \rangle} = 1$, then a sharp upper bound on R is equal to 0.
- Consider the $\sigma =$ GORGE regular expression. Recall that both a_σ and b_σ are equal to 1, the size ω_σ of σ is equal to 2, the height η_σ of σ is equal to 1, and for any time-series length $n > \omega_\sigma + 1$, the range $\phi_\sigma^{\langle n \rangle}$ of σ wrt $\langle n \rangle$ is equal to $e_\sigma \cdot (n - 1 - \eta_\sigma) + c_\sigma + \eta_\sigma = \eta_\sigma + 1 = 2$. Since there exists a word, namely $v =$ '><', in $\mathcal{L}_\sigma$ whose length equals 2 and whose height is equal to 1, and $\langle e_\sigma, c_\sigma \rangle$ is $\langle 0, 1 \rangle$, σ has the $\overline{\text{WIDTH}}$-monotonous property. Hence, we apply Theorem 7.3.1 for computing a sharp upper bound on R.
  - If $u - \ell \geq 2$, then a sharp upper bound on R is equal to $n - a_\sigma - b_\sigma = n - 2$.
  - If $u - \ell < 2$, then a sharp upper bound on R is equal to $\omega_\sigma + 1 - a_\sigma - b_\sigma = 1$.
- Consider the $\sigma =$ STRICTLY_DECREASING_SEQUENCE regular expression. Recall that both a_σ and b_σ are equal to 0, the size ω_σ of σ is equal to 1, the height η_σ of σ is equal to 1, and for any time-series length $n > \omega_\sigma + 1$, the range $\phi_\sigma^{\langle n \rangle}$ of σ wrt $\langle n \rangle$ is equal to $e_\sigma \cdot (n - 1 - \eta_\sigma) + c_\sigma + \eta_\sigma = n - 1 - \eta_\sigma + \eta_\sigma = n - 1$. Since there exists a word, namely $v =$ '>', in $\mathcal{L}_\sigma$ whose length is equal to 1 and whose height is equal to 1, and $\langle e_\sigma, c_\sigma \rangle$ is $\langle 1, 0 \rangle$, σ has the $\overline{\text{WIDTH}}$-monotonous property. Hence, we apply Theorem 7.3.1 for computing a sharp upper bound on R.
  - If $u - \ell \geq \phi_\sigma^{\langle n \rangle} = n - 1$, then a sharp upper bound on R is equal to $n - a_\sigma - b_\sigma = n$.
  - If $u - \ell < \phi_\sigma^{\langle n \rangle} = n - 1$, then a sharp upper bound on R is equal to $u - \ell + 1$. △

7.3.3 Upper Bound for SUM_WIDTH_σ

We now consider the SUM_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ family of time-series constraints with σ being a non-fixed length regular expression and with every X_i ranging over the same integer interval domain $[\ell, u]$. Under some hypothesis on the overlap of σ wrt $\langle \ell, u \rangle$, Lemma 7.3.1 provides an upper bound on R and a condition when this bound is sharp. Then, Theorem 7.3.2 extends the bound of Lemma 7.3.1 and gives a more general condition under which the extended bound on R is sharp.

Lemma 7.3.1 (not necessarily sharp upper bound for SUM_WIDTH_σ)**.** Consider a SUM_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[\ell, u]$, and with σ being a non-fixed length regular expression.

(i) If $o_\sigma^{\langle \ell, u \rangle} \leq a_\sigma + b_\sigma$ then $n - a_\sigma - b_\sigma$ is an upper bound on R.

(ii) If, in addition, $u - \ell \geq \phi_\sigma^{\langle n \rangle}$, then this bound is sharp.

Proof. [Proof of (i)] Let us consider a time series t of length n over $[\ell, u]$ that has $p > 1$ σ-patterns. Let ω_i be the length of the σ-pattern i (with i in $[1, p]$); let n_r be the number of time-series variables that are not in any extended σ-pattern of t; and let o_i be the number of common time-series variables of the extended σ-patterns i and $i+1$. Then, the following equality holds

$$n = \omega_1 + a_\sigma + b_\sigma + \sum_{i=1}^{p-1}(\omega_{i+1} + a_\sigma + b_\sigma - o_i) + n_r. \tag{7.1}$$

The time series t yields $\sum_{i=1}^{p} \omega_i$ as the value of R, thus we express this quantity from Equality 7.1 and obtain

$$R = n - n_r - p \cdot (a_\sigma + b_\sigma) + \sum_{i=1}^{p-1} o_i. \tag{7.2}$$

In order to prove that $n - a_\sigma - b_\sigma$ is a valid upper bound on R, we show that the difference between $n - a_\sigma - b_\sigma$ and the right-hand side of Equality 7.2 is always non-negative if $o_\sigma^{\langle \ell,u \rangle} \leq a_\sigma + b_\sigma$.

$$n - (a_\sigma + b_\sigma) - n + n_r + p \cdot (a_\sigma + b_\sigma) - \sum_{i=1}^{p-1} o_i = n_r + (p-1) \cdot (a_\sigma + b_\sigma) - \sum_{i=1}^{p-1} o_i. \tag{7.3}$$

The value of n_r is non-negative, and by the definition of $o_\sigma^{\langle \ell,u \rangle}$, every o_i is not greater than $o_\sigma^{\langle \ell,u \rangle}$. In addition, we have the following inequality $o_\sigma^{\langle \ell,u \rangle} \leq a_\sigma + b_\sigma$. Hence, a lower estimate of the right-hand side of Equality 7.3 is given by the following inequality

$$n_r + (p-1) \cdot (a_\sigma + b_\sigma) - \sum_{i=1}^{p-1} o_i \geq 0 + (p-1) \cdot (a_\sigma + b_\sigma) - (p-1) \cdot (a_\sigma + b_\sigma) = 0. \tag{7.4}$$

By Inequality 7.4 we obtain that, when $o_\sigma^{\langle \ell,u \rangle} \leq a_\sigma + b_\sigma$, the difference between $n - a_\sigma - b_\sigma$ and the value of R is always non-negative. Hence, $n - a_\sigma - b_\sigma$ is an upper bound on R.

[Proof of (ii)] We now show that $n - a_\sigma - b_\sigma$ is a sharp upper bound on R, when $u - \ell \geq \phi_\sigma^{\langle n \rangle}$. By definition of $\phi_\sigma^{\langle n \rangle}$, the range of σ wrt $\langle n \rangle$, there exists a word v of length $n-1$ in $\mathcal{L}_\sigma$ whose height is at most $u - \ell$. Hence, there exists at least one ground time series of length n over $[\ell, u]$ whose signature is v, all its time-series variables belong to a single extended σ-pattern. For such a time series, the value of p equals 1, and n_r equals 0. By the right-hand side of Equality 7.2, we have that R equals $n - a_\sigma - b_\sigma - 0 - (1-1)(a_\sigma + b_\sigma) = n - a_\sigma - b_\sigma$, which was proved to be an upper bound. Hence, in this case $n - a_\sigma - b_\sigma$ is a sharp upper bound on R. □

Theorem 7.3.2 (sharp upper bound for SUM_WIDTH_σ)**.** Consider a SUM_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with σ being a non-fixed-length regular expression and every X_i ranging over the same integer interval domain $[\ell, u]$. If σ has both the $\overline{\text{WIDTH}}$-monotonous property and the $\overline{\text{WIDTH}}$-sum property for $[\ell, u]$, then a sharp upper bound on R is

$$\begin{cases} n - a_\sigma - b_\sigma & \text{if } u - \ell \geq \phi_\sigma^{\langle n \rangle}, \quad (7.1) \\ e_\sigma \cdot (n - \rho_\sigma^{\langle \ell,u,n \rangle}) + c_\sigma \cdot (\omega_\sigma + 1 - a_\sigma - b_\sigma) \cdot \tau_\sigma^{\langle \ell,u,n \rangle} & \text{if } u - \ell < \phi_\sigma^{\langle n \rangle}, \quad (7.2) \end{cases}$$

where:

- e_σ and c_σ are parameters of the regular expression σ, introduced in Property 7.3.1.
- $\rho_\sigma^{\langle \ell,u,n \rangle}$ equals $\min(1, \max(0, \eta_\sigma + 1 - (u - \ell))) \cdot (n \mod 2)$.
- $\tau_\sigma^{\langle \ell,u,n \rangle}$ is the maximum number of σ-patterns of shortest length in a time series among all ground time series of length n over $[\ell, u]$.

Proof. When a regular expression σ has the $\overline{\text{WIDTH}}$-sum property for $[\ell, u]$, Condition (i) of Lemma 7.3.1 is satisfied and thus, $n - a_\sigma - b_\sigma$ is an upper bound on R.
[Case (7.1): $u - \ell \geq \phi_\sigma^{\langle n \rangle}$]. Since Condition (ii) of Lemma 7.3.1 is also satisfied, by Lemma 7.3.1, $u - \ell \geq \phi_\sigma^{\langle n \rangle}$ is a sharp upper bound on R.
[Case (7.2): $u - \ell < \phi_\sigma^{\langle n \rangle}$]. Let us consider the three potential values of $\langle e_\sigma, c_\sigma \rangle$ from Condition (ii) of Property 7.3.1:

(a) The case of $\langle e_\sigma, c_\sigma \rangle$ being $\langle 0, 0 \rangle$. Since $u - \ell < \eta_\sigma$, the necessary-sufficient condition, i.e. Property 7.1.1, is not satisfied, and thus no word of $\mathcal{L}_\sigma$ can occur in the signature of $\langle X_1, X_2, \ldots, X_n \rangle$. Hence, R is equal to the identity value of the aggregator, namely 0.

(b) The case of $\langle e_\sigma, c_\sigma \rangle$ being $\langle 0, 1 \rangle$. Since $u - \ell \leq \eta_\sigma$, only a shortest word with a height being η_σ may occur in a signature of $\langle X_1, X_2, \ldots, X_n \rangle$, as it was shown in the proof of Theorem 7.3.1. By Condition (i) of Property 7.3.1, such a word exists, and thus a sharp upper bound on R is equal to $\omega_\sigma + 1 - a_\sigma - b_\sigma$. Hence, any σ-pattern of any ground time series of length n over $[\ell, u]$ is of length $\omega_\sigma + 1 - a_\sigma - b_\sigma$. Since it is not possible to increase the length of any σ-patterns, in order to maximise R, it is necessary to maximise the number of σ-patterns of shortest length in a time series of length n over $[\ell, u]$. Since $\tau_\sigma^{\langle \ell, u, n \rangle}$ is the maximum number of σ-patterns of minimum length, a sharp upper bound on R equals $(\omega_\sigma + 1 - a_\sigma - b_\sigma) \cdot \tau_\sigma^{\langle \ell, u, n \rangle}$.

(c) The case of $\langle e_\sigma, c_\sigma \rangle$ being $\langle 1, 0 \rangle$. When σ has the $\overline{\text{WIDTH}}$-sum-property for $[\ell, u]$, it belongs to the following class of regular expressions: a_σ, b_σ, $o_\sigma^{\langle \ell, u \rangle}$ are all equal to 0, and ω_σ is equal to 1. Consider a time series t of length n over $[\ell, u]$ with $p \geq 1$ σ-patterns, where ω_i is the length of the σ-pattern i, o_i is the overlap of the extended σ-patterns i and $i + 1$, and $\rho_\sigma^{\langle \ell, u, n \rangle}$ is the number of time-series variables of t that do not belong to any extended σ-pattern of t. Then, the following equality holds

$$R = n - \rho_\sigma^{\langle \ell, u, n \rangle} - p \cdot (a_\sigma + b_\sigma) + \sum_{i=1}^{p-1} o_i.$$

In this equality we replace a_σ, and b_σ with their actual values, namely 0, which gives a simplified equality $R = n - \rho_\sigma^{\langle \ell, u, n \rangle}$. Since the smaller $\rho_\sigma^{\langle \ell, u, n \rangle}$, the larger is R, the aim is to find a time series for which $\rho_\sigma^{\langle \ell, u, n \rangle}$ is minimum. Assume that in such a time series p equals the maximum number of σ-patterns in a time series among all ground time series of length n over $[\ell, u]$. Then, $\rho_\sigma^{\langle \ell, u, n \rangle}$ is strictly less than $\omega_\sigma + 1 = 2$, otherwise there would be a contradiction with the maximality of p. Hence, t has at most one time-series variable that is outside of any extended σ-pattern of t. By definition of $\phi_\sigma^{\langle n \rangle}$, the number of time-series variables in any extended σ-pattern is at most $u - \ell + 1$, thus if t contains at least one σ-pattern shorter than $u - \ell + 1$ the value of $\rho_\sigma^{\langle \ell, u, n \rangle}$ can be decreased by extending this σ-pattern with one time-series variable. Furthermore, if $u - \ell \geq \eta_\sigma + 1$, then $\rho_\sigma^{\langle \ell, u, n \rangle} = 0$, otherwise $\rho_\sigma^{\langle \ell, u, n \rangle} = n \mod 2$. Hence, the minimum value of $\rho_\sigma^{\langle \ell, u, n \rangle}$ equals $\min(1, \max(0, \eta_\sigma + 1 - (u - \ell))) \cdot (n \mod 2)$.

□

Note that for the 22 regular expressions in Table 5.2, the maximum number of σ-patterns of shortest length in a time series coincides with the maximum number of σ-patterns in the same time series. Although, in the general case it may not be true.

Example 7.3.2 (sharp upper bound for SUM_WIDTH_σ)**.** Consider a SUM_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[\ell, u]$, and each value of $\langle e_\sigma, c_\sigma \rangle$ in $\{\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle\}$.

- Consider the $\sigma =$ INFLEXION regular expression. In Example 7.3.1, we showed that the regular expression σ has the $\overline{\text{WIDTH}}$-monotonous property. Recall that $o_\sigma^{\langle \ell, u \rangle}$ is equal to 2 and both a_σ and b_σ are equal to 1. Hence, Condition (i) of Property 7.3.2 is also satisfied. Since for any time-series

length greater than $\omega_\sigma + 1$, the value of $\phi_\sigma^{\langle n \rangle}$ equals η_σ, Condition (ii) of Property 7.3.2 is trivially satisfied. Hence, σ has also the $\overline{\text{WIDTH}}$-sum property, and Theorem 7.3.2 can be used for computing a sharp upper bound on R:
  - If $u - \ell \geq \eta_\sigma = 1$, then a sharp upper bound on R is equal to $n - a_\sigma - b_\sigma = n - 2$.
  - If $u - \ell < \eta_\sigma = 1$, then a sharp upper bound on R is equal to 0.
- Consider the $\sigma =$ GORGE regular expression. In Example 7.3.1, we showed that the regular expression σ has the $\overline{\text{WIDTH}}$-sum property. Recall that $o_\sigma^{\langle \ell, u \rangle}$ is equal to 1 and both a_σ and b_σ are equal to 1. Hence, Condition (i) of Property 7.3.2 is also satisfied. Since for any time-series length greater than $\omega_\sigma + 1$, the value of $\phi_\sigma^{\langle n \rangle}$ equals $\eta_\sigma + 1$, Condition (ii) of Property 7.3.2 is trivially satisfied. Hence, σ has also the $\overline{\text{WIDTH}}$-sum property, and Theorem 7.3.2 can be used for computing a sharp upper bound on R:
  - If $u - \ell \geq 2$, then a sharp upper bound on R equals $n - a_\sigma - b_\sigma = n - 2$.
  - If $u - \ell < 2$, then a sharp upper bound on R is equal to $\tau_\sigma^{\langle \ell, u, n \rangle} \cdot (\omega_\sigma + 1 - a_\sigma - b_\sigma) = \tau_\sigma^{\langle \ell, u, n \rangle} \cdot (2 + 1 - 1 - 1) = \tau_\sigma^{\langle \ell, u, n \rangle}$.

  For this particular regular expression, $\tau_\sigma^{\langle \ell, u, n \rangle}$ equals the maximum number of σ-patterns in a time series among all ground time series of length n over $[\ell, u]$, namely $\left\lfloor \frac{n-1}{2} \right\rfloor$, which is the upper bound obtained in Section 7.2.
- Consider the $\sigma =$ STRICTLY_DECREASING_SEQUENCE regular expression. It was shown in Example 7.3.1 that σ has the $\overline{\text{WIDTH}}$-monotonous-property. Recall that $o_\sigma^{\langle \ell, u \rangle}$ is equal to 0, and both a_σ and b_σ are equal to 0, thus Condition (i) of Property 7.3.2 is also satisfied. Since $o_\sigma^{\langle \ell, u \rangle}$, a_σ and b_σ are all equal to 0, and ω_σ is equal to 1, Condition (ii) of Property 7.3.2 is also satisfied. Hence, σ has the $\overline{\text{WIDTH}}$-sum property, and Theorem 7.3.2 can be used for computing a sharp upper bound on R:
  - If $u - \ell \geq n - 1$, then a sharp upper bound on R is equal to n.
  - If $u - \ell < n - 1$, then a sharp upper bound on R is equal to $n - \rho_\sigma^{\langle \ell, u, n \rangle} = n - \min(1, \max(0, (2 - (u - \ell)) \cdot (n \mod 2)))$. △

7.3.4 Lower Bound for MIN_WIDTH_σ

Finally, consider the MIN_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ family of time-series constraints with σ being a non-fixed-length regular expression and with every X_i ranging over the same integer interval domain $[\ell, u]$. The next theorem, Theorem 7.3.3, provides a sharp lower bound on R assuming the property that we now introduce holds.

Property 7.3.3 ($\underline{\text{WIDTH}}$-occurrence property)**.** A non-fixed-length regular expression σ has the $\underline{\text{WIDTH}}$-occurrence property for an integer interval domain $[\ell, u]$, if there exists a shortest word v in $\mathcal{L}_\sigma$, i.e. $|v| = \omega_\sigma$, and a word w in $\{v <, v =, v >\}$ such that the following conditions are all satisfied:

(i) The height of v equals η_σ, the height of σ.

(ii) The height of w is less than or equal to $u - \ell$.

(iii) The word w is not a factor of any word in $\mathcal{L}_\sigma$.

Theorem 7.3.3 (sharp lower bound for MIN_WIDTH_σ)**.** Consider a MIN_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with σ being a non-fixed-length regular expression, and with every X_i having the same integer interval domain $[\ell, u]$. If σ has the $\underline{\text{WIDTH}}$-occurrence property for $[\ell, u]$, then a sharp lower bound on R equals $\omega_\sigma + 1 - a_\sigma - b_\sigma$.

Proof. Since ω_σ is the length of a shortest word in $\mathcal{L}_\sigma$, the length of any σ-pattern is at least $\omega_\sigma + 1 - a_\sigma - b_\sigma$, and thus it is a lower bound on R. When σ has the $\underline{\text{WIDTH}}$-occurrence property, there exists a shortest word v in $\mathcal{L}_\sigma$ and a word w in $\{v <, v =, v >\}$ such that the three conditions of Property 7.3.3 are all satisfied. We now show that in this case, the bound is sharp.

Case (a): $n = \omega_\sigma + 1$. When Condition (i) of Property 7.3.3 is satisfied, there exists a ground time series of length $n = \omega_\sigma + 1$ over $[\ell, u]$ whose signature is v. Hence, $\omega_\sigma + 1 - a_\sigma - b_\sigma$ is a sharp lower bound on R.

Case (b): $n > \omega_\sigma + 1$. When Condition (ii) of Property 7.3.3 is satisfied, there exists a ground time series t of length n over $[\ell, u]$ whose signature is a word in the language of the '$w =^*$' regular expression. If Condition (iii) of Property 7.3.3 is also satisfied, then the v in the signature of t is a maximal occurrence of σ, because w is not a factor of any word in $\mathcal{L}_\sigma$. The length of the corresponding σ-pattern is $\omega_\sigma + 1 - a_\sigma - b_\sigma$, thus this value is a sharp lower bound on R. □

Example 7.3.3 (sharp lower bound for MIN_WIDTH_σ)**.** Consider a MIN_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with σ being the INFLEXION regular expression and with every X_i ranging over the same integer interval domain $[\ell, u]$ such that $u - \ell \geq \eta_\sigma = 1$. It was shown in Example 2.0.2 that σ is a non-fixed-length regular expression. Furthermore, there exists a word $v =$ '<>' and a word $w =$ '<>=' in {'$v <$', '$v =$', '$v >$'} such that the following conditions are all satisfied:

- The height of v equals $\eta_\sigma = 1$. (Cond. (i) of Prop. 7.3.3)
- The height of w equals 1, and thus is less than or equal to $u - \ell$. (Cond. (ii) of Prop. 7.3.3)
- The word w is not a factor of any word in $\mathcal{L}_\sigma$. (Cond. (iii) of Prop. 7.3.3)

Hence, σ has the WIDTH-occurrence property for $[\ell, u]$, and by Theorem 7.3.3, a sharp lower bound on R equals $\omega_\sigma + 1 - a_\sigma - b_\sigma = 2 + 1 - 1 - 1 = 1$. △

All the 22 regular expressions in Table 5.2 have the WIDTH-occurrence-property for any integer interval domain $[\ell, u]$, except the STEADY_SEQUENCE regular expression when $\ell = u$. This special case is considered in Proposition 7.3.1.

Proposition 7.3.1 (sharp lower bound for MIN_WIDTH_STEADY_SEQUENCE)**.** Consider a MIN_WIDTH_$\sigma($ $\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with σ being the STEADY_SEQUENCE regular expression and with every X_i being over an integer interval domain $[\ell, u]$ such that $\ell = u$. A sharp lower bound on R equals n.

Proof. When ℓ equals u, there exists a single ground time series t of length n over $[\ell, u]$ with all time-series variables having the same value, namely ℓ. The signature of t is a sequence of $n - 1$ equalities, which is a word in $\mathcal{L}_\sigma$. Hence, every time-series variable of t belongs to a single extended σ-pattern of t, and thus a sharp lower bound on R equals $n - a_\sigma - b_\sigma = n$. □

7.4 Synthesis

Consider a $g_f_\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with every X_i being over the same integer interval domain $[\ell, u]$. Table 7.2 provides a synthesis of the bounds on R obtained in Sections 7.2, 7.3 and in [8], when $\langle g, f \rangle$ is in $\{\langle \texttt{max}, \texttt{min} \rangle, \langle \texttt{max}, \texttt{width} \rangle, \langle \texttt{min}, \texttt{width} \rangle, \langle \texttt{sum}, \texttt{one} \rangle, \langle \texttt{sum}, \texttt{width} \rangle\}$. The theorems and the propositions mentioned in Table 7.2 were applied for computing sharp bounds on R for 93 time-series constraints of Volume II of the Global Constraint Catalogue [10]. An entry of Table 7.2 corresponds to an upper (respectively lower) bound on R for a $g_f_\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[\ell, u]$, if the corresponding "Type" column contains $\overline{R}$ (respectively $\underline{R}$). The "Theorem" column contains the theorem or the proposition providing the corresponding sharp bound under the hypothesis that σ has the properties mentioned in the corresponding "Properties" column. The "Theorem" (respectively "Property") column also recalls the set of characteristics used in the bound of the corresponding theorem or proposition (respectively property).

Note that when the aggregator is `max` (respectively `min`) we do not consider a lower (respectively upper) bound on R. When σ has the NB-simple property for $[\ell, u]$, there exists a time series of length n over $[\ell, u]$ that contains no σ-patterns, and thus such a time series yields the identity value of `max` (respectively `min`), which is $-\infty$ (respectively $n + 1$).

$\langle g, f \rangle$	Type	Theorem	Properties
$\langle$ `sum`, `one` $\rangle$	$\underline{R}$	Theorem 7.2.1	$\underline{\text{NB}}$-simple (Θ_σ)
	$\underline{R}$	Proposition 7.2.1	$\sigma =$ STEADY, $u = \ell$
	$\underline{R}$	Proposition 7.2.2	$\sigma =$ STEADY_SEQUENCE, $u = \ell$
	$\overline{R}$	Theorem 7.2.2 ($\omega_\sigma, \eta_\sigma, o_\sigma^{\langle \ell,u \rangle}, \delta_\sigma^{\langle \ell,u \rangle}$)	$\overline{\text{NB}}$-overlapping or $\overline{\text{NB}}$-non-overlapping ($\omega_\sigma, \eta_\sigma, o_\sigma^{\langle \ell,u \rangle}, \delta_\sigma^{\langle \ell,u \rangle}$)
	$\overline{R}$	Proposition 7.2.3	$\sigma =$ STEADY_SEQUENCE, $u = \ell$
$\langle$ `max`, `width` $\rangle$	$\overline{R}$	Theorem 7.3.1 ($\omega_\sigma, \eta_\sigma, \phi_\sigma^{\langle \ell,u \rangle}$)	$\overline{\text{WIDTH}}$-monotonous ($\omega_\sigma, \eta_\sigma, \phi_\sigma^{\langle \ell,u \rangle}$)
$\langle$ `sum`, `width` $\rangle$	$\overline{R}$	Theorem 7.3.2 ($\omega_\sigma, \eta_\sigma, \phi_\sigma^{\langle \ell,u \rangle}$)	$\overline{\text{WIDTH}}$-monotonous and $\overline{\text{WIDTH}}$-sum ($\omega_\sigma, \eta_\sigma, \phi_\sigma^{\langle \ell,u \rangle}, o_\sigma^{\langle \ell,u \rangle}$)
$\langle$ `min`, `width` $\rangle$	$\underline{R}$	Theorem 7.3.3 (ω_σ)	$\underline{\text{WIDTH}}$-occurrence ($\omega_\sigma, \eta_\sigma$)
	$\underline{R}$	Proposition 7.3.1	$\sigma =$ STEADY_SEQUENCE, $u = \ell$
$\langle$ `max`, `min` $\rangle$	$\overline{R}$	Theorem 1 in [8]	The Condition of Theorem 1 in [8]

Table 7.2 – A synthesis of all the bounds presented in Sections 7.2, 7.3, and in [8].

The 22 regular expressions in Table 5.2 have the $\underline{\text{NB}}$-simple property for any domain, except the STEADY and the STEADY_SEQUENCE regular expressions when $\ell = u$. Table 7.3 classifies the 22 regular expressions according to the set of properties they share. There are three main groups, and two special ones, namely for the STEADY and for the STEADY_SEQUENCE regular expressions. The partitioning into three main groups is related to the fact that the entry of Table 7.2 with Theorem 7.2.2, contains a disjunction between the $\overline{\text{NB}}$-overlapping and the $\overline{\text{NB}}$-non-overlapping properties. Furthermore, a regular expression σ cannot have both properties for the same integer interval domain $[\ell, u]$. This allows to partition the 22 regular expressions into three classes, namely:

1. The regular expressions that have the $\overline{\text{NB}}$-overlapping property for any $[\ell, u]$, i.e. the first group in Table 7.3.
2. The regular expressions that have the $\overline{\text{NB}}$-non-overlapping property for any $[\ell, u]$, i.e. the second group in Table 7.3.
3. The regular expressions that have the $\overline{\text{NB}}$-non-overlapping property for any $[\ell, u]$ such that $u - \ell = \eta_\sigma$, and have the $\overline{\text{NB}}$-overlapping property for any $[\ell, u]$ such that $u - \ell > \eta_\sigma$, i.e. the third group in Table 7.3.

The STEADY_SEQUENCE represents a special case, because when $u - \ell = \eta_\sigma$, σ has neither property for $[\ell, u]$, and when $u - \ell > \eta_\sigma$, σ has the $\overline{\text{NB}}$-non-overlapping property for $[\ell, u]$.

	Regular Expressions	Set of Properties
Overlapping Class	BUMP_ON_DECREASING_SEQUENCE DIP_ON_INCREASING_SEQUENCE GORGE INFLEXION PEAK PLAIN PLATEAU PROPER_PLAIN PROPER_PLATEAU SUMMIT VALLEY	$\underline{\text{NB}}$-simple $\overline{\text{NB}}$-overlapping $\overline{\text{WIDTH}}$-monotonous $\overline{\text{WIDTH}}$-sum $\underline{\text{WIDTH}}$-occurrence Condition of Theorem 1 in [8]
Non-Overlapping Class	DECREASING_SEQUENCE INCREASING_SEQUENCE STRICTLY_DECREASING_SEQUENCE STRICTLY_INCREASING_SEQUENCE	$\underline{\text{NB}}$-simple $\overline{\text{NB}}$-non-overlapping $\overline{\text{WIDTH}}$-monotonous $\overline{\text{WIDTH}}$-sum $\underline{\text{WIDTH}}$-occurrence Condition of Theorem 1 in [8]
Overlapping Non-Overlapping Class	DECREASING INCREASING DECREASING_TERRACE INCREASING_TERRACE ZIGZAG	$\underline{\text{NB}}$-simple $\overline{\text{NB}}$-non-overlapping when $u - \ell = \eta_\sigma$ $\overline{\text{NB}}$-overlapping when $u - \ell \geq \eta_\sigma + 1$ $\overline{\text{WIDTH}}$-monotonous $\overline{\text{WIDTH}}$-sum $\underline{\text{WIDTH}}$-occurrence Condition of Theorem 1 in [8]
Special Case	STEADY	$\underline{\text{NB}}$-simple when $u - \ell > \eta_\sigma$ $\overline{\text{NB}}$-non-overlapping when $u - \ell = \eta_\sigma$ $\overline{\text{NB}}$-overlapping $\overline{\text{WIDTH}}$-monotonous $\overline{\text{WIDTH}}$-sum $\underline{\text{WIDTH}}$-occurrence Condition of Theorem 1 in [8]
	STEADY_SEQUENCE	$\underline{\text{NB}}$-simple when $u - \ell > \eta_\sigma$ $\overline{\text{NB}}$-non-overlapping when $u - \ell = \eta_\sigma$ $\overline{\text{NB}}$-overlapping $\overline{\text{WIDTH}}$-monotonous $\overline{\text{WIDTH}}$-sum $\underline{\text{WIDTH}}$-occurrence when $u - \ell > \eta_\sigma$ Condition of Theorem 1 in [8]

Table 7.3 – Classification of regular expressions: regular expression names σ, their properties and conditions on domain $[\ell, u]$ when they hold.

7.5 Conclusion

We introduce the concept of regular-expression characteristic as a way to unify combinatorial aspects of quantitative extensions of regular languages. We illustrate that approach for time-series constraints where, introducing six regular-expression characteristics, allows coming up with generic bounds for families of time-series constraints

Summary of this Chapter:

The main contribution of this chapter is a systematic method for synthesising bounds for time-series constraints using the introduced concept of regular-expression characteristic. For the NB_σ and the SUM_WIDTH_σ families, we defined classes of regular expressions, for which our method applies.

Chapter 8

Synthesising Parameterised AMONG Implied Constraints

This chapter is an extended version of an article published in the proceedings of the *CP'17* conference [12]. The final authenticated version of this article is available online at: `http://dx.doi.org/10.1007/978-3-319-66158-2_3`.

This chapter focusses on the g_SURF_$\sigma(X, R)$ families of time-series constraints with g being either `max` or `sum`, and with σ being one of the 22 regular expressions of Table 5.2, as they were reported to be the most difficult in the preceding work of [8]. Each constraint of one of the two families restricts R to be the result of applying the aggregator g to the sum of the elements corresponding to the occurrences of a regular expression σ [5] in a time series X. These constraints play an important role in modelling power systems [28]. If the measured values correspond to the power input/output, then the surface feature `surf` describes the energy used/generated during the period of pattern occurrence. The `sum` aggregator imposes a bound on the total energy during all pattern occurrences in the time series, the `max` aggregator is used to limit the maximal energy during a single pattern occurrence. Generating time series verifying a set of specific time-series constraints is also useful in different contexts like trace generation, i.e. generating typical energy consumption profiles of a data centre [62, 84], or staff scheduling problems, e.g. generating manpower profiles over time subject to work regulations and to a demand [3, 11].

Many constraints of these families are intractable, thus in order to improve the efficiency of the solving we need to address the combinatorial aspect of time-series constraints. We improve the reasoning for such time-series constraints by identifying AMONG implied constraints. Learning parameters of global constraints like AMONG [25] is a well-known method for strengthening constraint models [37, 36, 111] with the drawback that it is instance specific, so this alternative was not explored here. Taking exact domains into account would lead to filtering algorithms rather than to implied constraints which assume the same minimum/maximum.

While coming up with implied constraints is usually problem specific, the theoretical contribution of this chapter is a *unique per family* AMONG implied constraint, *that is valid for all regular expressions* of Table 5.2 and that covers all the 22 time-series constraints of the corresponding family. Hence, it covers 66 time-series constraints in total. The main focus of this work is on reusable necessary conditions that can be associated with a class of time-series constraints.

First, we show in Section 8.1 that saying whether the SUM_SURF_PEAK time-series constraint has a solution or not is an NP-complete problem. Then, after introducing several regular-expression characteristics, Section 8.2 presents the main contribution, Theorems 8.2.1, 8.2.2 and 8.2.3, while Tables 8.2 and 8.3 provide the corresponding derived concrete implied constraints for some subset of the MAX_SURF_σ and the SUM_SURF_σ time-series constraints, respectively.

8.1 Complexity of the SUM_SURF_PEAK Time-Series Constraint

In this section, we prove that saying whether the SUM_SURF_PEAK time-series constraint has a solution or not is an NP-complete problem.

Theorem 8.1.1 (NP-completness of SUM_SURF_PEAK)**.** Consider a sequence of integer variables $X = \langle X_1, X_2, \ldots, X_n \rangle$ with finite domains. The problem of identifying whether there exists an assignment of the variables of X satisfying the constraint SUM_SURF_PEAK(X, R) with a fixed value of R is NP-complete.

Proof. For a given ground time series of length n and an integer value R, we can use the register automaton for SUM_SURF_PEAK to check whether the constraint holds or not. Such a check requires $O(n)$ time complexity and $O(1)$ space complexity. Hence our problem is in class NP.

We now show that the considered problem is NP-complete by a polynomial reduction of the consistency check of B in the domain of R for an instance of SUBSET SUM [72] to SUM_SURF_PEAK(X, R). The SUBSET SUM problem is described as follows: given a multiset of strictly positive integers $A = \{A_1, A_2, \ldots, A_m\}$, and a strictly positive integer B, does there exist a subset A' of A such that $\sum_{A_i \in A'} A_i = B$?

Let X be a sequence $\langle 0, Y_1, 0, Y_2, 0, \ldots, 0, Y_m, 0 \rangle$, where the domain of every Y_i is $\{0, A_i\}$ with $A_i > 0$; let B be a strictly positive integer; and let E be a multiset containing the upper limits of the domains of all elements of X. Note that X and E have the same number of elements. We now show that a solution to SUM_SURF_PEAK(X, B) is a solution to SUBSET SUM on (E, B) and vice versa.

$\Rightarrow$ A solution to SUM_SURF_PEAK(X, B) is a solution to the equality $\sum_{i=1}^{m} X_i = B$, i.e. a solution to SUBSET SUM on (E, B).

$\Leftarrow$ Let A' be a solution to SUBSET SUM on (E, B). For each $A_i \in A'$, assign A_i to a variable $X_i \in X$ such that the upper limit of the domain of X_i is A_i. Assign 0 to all remaining variables. The resulting complete assignment of X is by construction a solution to SUM_SURF_PEAK(X, B). Indeed, for each X_i such that X_i is A_i there is an extended PEAK-pattern $\langle 0, A_i, 0 \rangle$, otherwise $X_i = 0$ and does not belong to any PEAK-pattern. The sum of the surfaces of the PEAK-patterns of X is $\sum_{A_i \in A'} A_i = B$, as A' is a solution to SUBSET SUM on (E, B). □

8.2 Deriving an AMONG Implied Constraint

Consider a $g_f_\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with g being in $\{\texttt{sum}, \texttt{max}, \texttt{min}\}$, with f being the `surf` feature, and with every X_i ranging over the same integer interval domain $[\ell, u]$ such that $u > 0$. For brevity, we do not consider here the case when $u \leq 0$, since it can be handled in a symmetric way. We derive an AMONG$(\mathcal{N}, \langle X_1, X_2, \ldots, X_n \rangle, \left\langle \underline{\mathcal{I}}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle}, \underline{\mathcal{I}}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle} + 1, \ldots, \overline{\mathcal{I}}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle} \right\rangle)$ implied constraint, where:

- For any value of R, $\mathcal{N}$ is an integer variable whose lower bound only depends on R, σ, f, ℓ, u, and n.
- The interval $\mathcal{I}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle} = [\underline{\mathcal{I}}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle}, \overline{\mathcal{I}}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle}]$ is a subinterval of $[\ell, u]$, which is called the *interval of interest of* $\langle g, f, \sigma \rangle$ *wrt* $\langle \ell, u \rangle$ and defined in Section 8.2.1.

Such an AMONG [25, 40] constraint is satisfied if exactly $\mathcal{N}$ variables of $\langle X_1, X_2, \ldots, X_n \rangle$ are assigned a value in $\mathcal{I}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle}$. Before formally describing how to derive this implied constraint, we provide an illustrating example.

Example 8.2.1 (illustrating example)**.** Consider a MAX_SURF_$\sigma(\langle X_1, X_2, \ldots X_7 \rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[1, 4]$, and with σ being the DECREASING_SEQUENCE regular expression (see Table 5.2). Let us observe what happens when R is fixed, for example, to 18. The following table below gives the two distinct σ-patterns such that at least one of them appear in every ground time series $X = \langle X_1, X_2, \ldots, X_7 \rangle$ that yields 18 as the value of R:

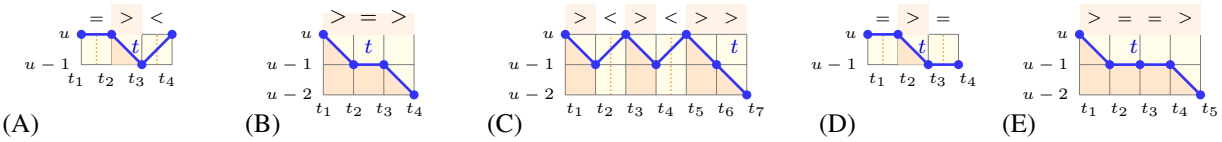


Figure 8.1 – For all the figures, σ is the DECREASING_SEQUENCE regular expression. A time series t (A) with one σ-pattern, which contains a single occurrence of value $u-1$; (B) with one σ-pattern, which contains 2 occurrences of value $u-1$; (C) with the maximum number, 3, of σ-patterns, which all contain one occurrence of value $u-1$, and only one contains an occurrence of value $u-2$; (D) with one σ-pattern, which contains one occurrence of both u and $u-1$; (E) with one σ-pattern, whose width is maximum among all other σ-patterns in ground time series of length 5 over the same domain $[u-2, u]$.

σ-pattern 1	σ-pattern 2
$\langle 4,3,3,3,3,2\rangle$	$\langle 4,3,3,3,2,2,1\rangle$

By inspection, we observe that for any ground time series X for which R equals 18, its single σ-pattern contains at least 4 time-series variables whose values are in $[3,4]$. Hence, we can impose an AMONG$(\mathcal{N}, \langle X_1, X_2, \ldots, X_7\rangle, \langle 3,4\rangle)$ implied constraint with $\mathcal{N} \geq 4$. △

We now formalise the ideas presented in Example 8.2.1 and systematise the way we obtain such an implied constraint even when R is *not initially fixed*.

- Section 8.2.1 introduces three new regular-expression characteristics σ, which were not presented in Chapter 7 and will be used to obtain a parameterised implied constraint:
  - the *interval of interest* of $\langle g, f, \sigma\rangle$ wrt $\langle \ell, u\rangle$ (see Definition 8.2.1),
  - the *maximum value occurrence* of σ wrt $\langle \ell, u, n\rangle$ (see Definition 8.2.2), and
  - the *big width* of σ wrt $\langle \ell, u, n\rangle$ (see Definition 8.2.3).
- Based on these three characteristics, the height of σ and the overlap of σ wrt $\langle \ell, u\rangle$, introduced in Definition 7.1.4 and Definition 7.1.10, respectively, Section 8.2.2 presents a systematic way of deriving AMONG implied constraints for the MAX_SURF_σ, MIN_SURF_σ and the SUM_SURF_σ families of time-series constraints.

8.2.1 Regular-Expression Characteristics

To get a unique per family AMONG implied constraint that is valid for any g_SURF_$\sigma(X, R)$ time-series constraint with g being in $\{\texttt{max}, \texttt{min}, \texttt{sum}\}$, we introduce three new regular-expression characteristics that will be used for parametrising our implied constraint. First, Definition 8.2.1 defines the specific range of values on which the AMONG implied constraint focusses on.

Definition 8.2.1 (interval of interest)**.** Consider a $g_f_\sigma(X, R)$ time-series constraint with X being a time series over an integer interval domain $[\ell, u]$. The *interval of interest* of $\langle g, f, \sigma\rangle$ wrt $\langle \ell, u\rangle$, denoted by $\mathcal{I}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$, is a function that maps an element of $\mathcal{T} \times \mathbb{Z} \times \mathbb{Z}$ to $\mathbb{Z} \times \mathbb{Z}$, where $\mathcal{T}$ denotes the set of all time-series constraints, and the result pair of integers is considered as an interval.

- The upper limit of $\mathcal{I}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$, denoted by $\overline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$, is the largest value in $[\ell, u]$ that can occur in a σ-pattern of a time series over $[\ell, u]$.
- The lower limit of $\mathcal{I}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$, denoted by $\underline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$, is the smallest value v in $[\max(\ell, u-\eta_\sigma-1), u]$ such that for any n in $\mathbb{N}$, the number of occurrences of v in the union of the σ-patterns of any maximal time series for g_f_σ of length n over $[\ell, u]$, is a non-constant function of n. If such v does not exist, then $\underline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$ equals $\overline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle} - \eta_\sigma$.

We focus on such intervals of interests because they consist of the largest values appearing in maximal time series for g_f_σ.

Example 8.2.2 (interval of interest)**.** Consider a g_SURF_$\sigma(X, R)$ time-series constraint with X being a time series over an integer interval domain $[\ell, u]$ such that $u > 0$. We consider different combinations of triples $\langle g, f, \sigma\rangle$ and their corresponding intervals of interest wrt $\langle \ell, u\rangle$. Note that the value of $\overline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$ depends only on σ, ℓ, and u and not on g and f.

- Let σ be the DECREASING_SEQUENCE $=$ '$(> (> \mid =)^*)^* >$' regular expression. The largest value appearing in the σ-patterns of X is u, and thus $\overline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle} = u$. We compute the value of $\underline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$ wrt two time-series constraints:
  - Let g be the `max` aggregator.
    - If $u - \ell = 1$, then any σ-pattern of X has a signature '$>$', i.e. contains only two elements. Then, the maximum value of R is reached for a time series t that contains the $\langle u, u-1\rangle$ σ-pattern. The rest of the variables of t are assigned any value, e.g. all other variables have a value of u. Such a time series t for the length 4 is shown in Part (A) of Figure 8.1. Further, for any v in $[\ell, u]$, the number of occurrences of v in the union of the σ-patterns of t is at most 1, which is a constant, and does not depend on n. By definition $\underline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle} = \overline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle} - \eta_\sigma = u - 1$.
    - If $u - \ell > 1$, then for any $n \geq 2$, any maximal time series t for g_f_σ contains a single σ-pattern whose signature is in the language of '$>=^*>$'. If, for example, $n = 4$, then t has $n - 2 = 2$ time-series variables with the values $u - 1$, which is depicted in Part (B) of Figure 8.1. In addition, the σ-pattern of t has a single occurrence of the value $u - 2$. Hence $\underline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle} = u - 1$.
  - Let g be the `sum` aggregator.
    For any integer $n \geq 2$, any maximal time series t for g_f_σ contains $\left\lfloor \frac{n}{2} \right\rfloor$ σ-patterns, which contains u and $u - 1$, and at most one of them has the value $u - 2$. Such a time series t for the length $n = 7$ is depicted in Part (C) of Figure 8.1. Hence $\underline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle} = u - 1$.
- Let σ be the PEAK $=$ '$< (< \mid =)^* (> \mid =)^* >$' regular expression whose $a_\sigma = b_\sigma = 1$. The value of $\overline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$ is equal to u, since it is the largest value appearing in σ-patterns of a time series over $[\ell, u]$. We consider the values of $\underline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$ wrt two time-series constraints:
  - Let g be the `max` aggregator. For any $n \geq 2$, any maximal time series for g_f_σ of length n contains a single σ-pattern, whose time-series variables equal u. Hence $\underline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle} = u$.
  - Let g be the `sum` aggregator. The set of maximal time series for g_f_σ is the same as for the MAX_SURF_σ constraint. Hence $\underline{\mathcal{I}}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle} = u$. △

The next regular-expression characteristic, we introduce, is a function of ℓ, u and n related to the maximum number of value occurrences in a σ-pattern.

Definition 8.2.2 (maximum value occurrence number)**.** Consider a regular expression σ, and a time series X of length n over an integer interval domain $[\ell, u]$. The *maximum value occurrence number* of v in $\mathbb{Z}$ wrt $\langle \ell, u, n\rangle$, denoted by $\mu^{\langle \ell,u,n\rangle}_\sigma(v)$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{N}^+ \times \mathbb{Z}$ to $\mathbb{N}$. It equals the maximum number of occurrences of the value v in one σ-pattern of X.

Example 8.2.3 (maximum value occurrence number)**.** Consider a regular expression σ and a time series X of length n over an integer interval domain $[\ell, u]$ such that $u > \ell$. We compute the maximum value occurrence number of various v in $\mathbb{Z}$ wrt $\langle \ell, u, n\rangle$.

If v is not in $[\ell, u]$, then $\mu^{\langle \ell,u,n\rangle}_\sigma(v) = 0$. Hence we focus on the case when $v \in [\ell, u]$.

- Let σ be the DECREASING_SEQUENCE regular expression.
  - If $u - \ell = 1$, then any σ-pattern of X has a signature '$>$', and thus it may have at most one occurrence of any value v in $[\ell, u]$. Hence for any v in $[\ell, u]$, $\mu^{\langle \ell,u,n\rangle}_\sigma(v) = 1$.
  - If $u - \ell > 1$, then we consider two subsets of $[\ell, u]$:

— For either v in the set $\{\ell, u\}$, the value of $\mu_\sigma^{\langle \ell,u,n\rangle}(v)$ is 1, since in any σ-pattern the lower and upper limits of the domain, namely ℓ and u, can appear at most once, as it illustrated in Part (D) of Figure 8.1. for the length $n = 4$.

— For any v in $[\ell + 1, u - 1]$, the value of $\mu_\sigma^{\langle \ell,u,n\rangle}(v)$ is $n - 2$, since v can occur at most $n - 2$ times in a σ-pattern of X. The time series in Part (B) of Figure 8.1. has a single σ-pattern, namely $\langle t_1, t_2, t_3, t_4\rangle$, which has $n - 2 = 4 - 2 = 2$ occurrences of the value $u - 1$.

∗ Let σ be the PEAK = '$< (< \mid =)^*(> \mid =)^* >$' regular expression.

◦ Any value v in $[\ell + 1, u]$ can occur at most $n - a_\sigma - b_\sigma = n - 2$ times in any σ-pattern of X. Hence for any v in $[\ell + 1, u]$, $\mu_\sigma^{\langle \ell,u,n\rangle}(v) = n - 2$.

◦ Since a_σ and b_σ equal 1, the value ℓ cannot appear in any σ-pattern of X. Hence, $\mu_\sigma^{\langle \ell,u,n\rangle}(\ell) = 0$.

△

The last regular-expression characteristic, we introduce, is the largest width of a σ-pattern in a time series.

Definition 8.2.3 (big width)**.** Consider a regular expression σ, and a time series X of length n over an integer interval domain $[\ell, u]$. The *big width* of σ wrt $\langle \ell, u, n\rangle$, denoted by $\beta_\sigma^{\langle \ell,u,n\rangle}$, is a function that maps an element of $\mathcal{R}_\Sigma \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{N}^+$ to $\mathbb{N}$. It equals the maximum width of a σ-pattern in X. If X cannot have any σ-patterns, then $\beta_\sigma^{\langle \ell,u,n\rangle}$ is 0.

Example 8.2.4 (big width)**.** Consider a regular expression σ and a time series X of length n over an integer interval domain $[\ell, u]$. We compute the big width of different σ wrt $\langle \ell, u\rangle$.

∗ Let σ be the DECREASING_SEQUENCE regular expression.

◦ If $n \leq 1$, then X cannot have any σ-patterns, since a minimum width σ-pattern contains at least two elements. Hence $\beta_\sigma^{\langle \ell,u,n\rangle} = 0$.
If $u - \ell = 0$, then no word of $\mathcal{L}_\sigma$ can appear in the signature of any ground time series over $[\ell, u]$, and thus X cannot have any σ-patterns. Hence $\beta_\sigma^{\langle \ell,u,n\rangle} = 0$.

◦ If $u - \ell = 1$ and $n \geq 2$, then any σ-pattern of X has a signature '$>$'. The width of such a σ-pattern is 2. Hence $\beta_\sigma^{\langle \ell,u,n\rangle} = 2$.

◦ If $u - \ell > 1$ and $n \geq 2$, then there exists a word in $\mathcal{L}_\sigma$ that is also in the language of '$>=^*>$' and whose length is $n - 1$. This word is the signature of some ground time series t of length n over $[\ell, u]$, which contains a single σ-pattern of width n. Such a time series t for the length $n = 5$ is illustrated in Part (E) of Figure 8.1. The width of a σ-pattern cannot be greater than n, thus $\beta_\sigma^{\langle \ell,u,n\rangle} = n$.

∗ Let σ be the PEAK regular expression.

◦ If $u = \ell$ or $n \leq 2$, then no word in the language of σ can appear in the signature of a ground time series over $[\ell, u]$. Hence $\beta_\sigma^{\langle \ell,u,n\rangle} = 0$.

◦ If $u - \ell \geq 1$ and $n \geq 3$, then the maximum length of the words in the language of σ that can appear in the signature of a ground time series over $[\ell, u]$ is $n - 1$. Hence $\beta_\sigma^{\langle \ell,u,n\rangle} = n + 1 - b_\sigma - a_\sigma = n - 2$. △

Table 8.1 gives the values of the two regular-expression characteristics for some regular expressions of Table 5.2, while Tables 8.2 and 8.3 provide the intervals of interest for 12 time-series constraints.

8.2.2 Deriving an AMONG Implied Constraint for the MAX_SURF_σ, MIN_SURF_σ and the SUM_SURF_σ Families

Consider a g_SURF_$\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ time-series constraint with every X_i ranging over the same integer interval domain $[\ell, u]$, and with g being in $\{\texttt{max}, \texttt{min}, \texttt{sum}\}$. Our goal is to estimate a lower bound on $\mathcal{N}$, which is the number of time-series variables in the σ-patterns of $\langle X_1, X_2, \ldots, X_n\rangle$ that

σ	$\mu_\sigma^{\langle \ell,u,n\rangle}(v)$	$\beta_\sigma^{\langle \ell,u,n\rangle}$
BUMP_ON_DECREASING_SEQUENCE	$\begin{cases} 1, \text{ if } v \in \{u, \ell\} \\ 2, \text{ if } v \in [\ell+1, u-1] \end{cases}$	3
DECREASING	$1, \forall v \in [\ell, u]$	2
DECREASING_SEQUENCE	$\begin{cases} 1, \text{ if } v \in \{u, \ell\} \\ n-2, \text{ if } v \in [\ell+1, u-1] \end{cases}$	$\begin{cases} 2, \text{ if } u-\ell=1 \\ n, \text{ otherwise} \end{cases}$
GORGE	$\begin{cases} 0, \text{ if } v = u \\ 1, \text{ if } v = \ell \\ n-2, \text{ if } v \in [\ell+1, u-1] \end{cases}$	$\begin{cases} 1, \text{ if } u-\ell=1 \\ n-2, \text{ otherwise} \end{cases}$
PEAK	$\begin{cases} 0, \text{ if } v = \ell \\ n-2, \text{ if } v \in [\ell+1, u] \end{cases}$	$n-2$
ZIGZAG	$\left\lfloor \frac{n-1}{2} \right\rfloor, \forall v \in [\ell, u]$	$n-2$

Table 8.1 – For every regular expression σ, $[\ell, u]$ is an integer interval domain, and n is a time series length, such that there is at least one ground time series of length n over $[\ell, u]$ whose signature contains at least one occurrence of σ. Then, $\mu_\sigma^{\langle \ell,u,n\rangle}(v)$ is the maximum value occurrence number of $v \in [\ell, u]$ wrt $\langle \ell, u, n\rangle$, and $\beta_\sigma^{\langle \ell,u,n\rangle}$ is the the big width of σ wrt $\langle \ell, u, n\rangle$.

must be assigned a value in the interval of interest $\mathcal{I}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle}$ of $\langle g, f, \sigma\rangle$ wrt $\langle \ell, u\rangle$, in order to satisfy the $g_\text{SURF}_\sigma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ constraint. Theorems 8.2.1, 8.2.2 and 8.2.3 present such inequality for the cases when g is `max`, `min`, and `sum`, respectively, using the three regular-expression characteristics introduced in Section 8.2.1 and also the height of σ, and the overlap of σ wrt $\langle \ell, u\rangle$, introduced in Definitions 7.1.4 an 7.1.10, respectively. Example 8.2.5 first conveys the intuition behind Theorem 8.2.1.

Example 8.2.5 (intuition behind Theorem 8.2.1)**.** Consider a $g_f_\sigma(X, R)$ time-series constraint with g being `max`, with f being `surf`, with σ being the DECREASING_SEQUENCE regular expression, and with X being a time series of length $n = 9$ over the integer interval domain $[\ell, u] = [0, 4]$. Let us assign R to the value 24, and let us compute a lower bound on $\mathcal{N}$, the number of variables of X that must be assigned a value from $\mathcal{I}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle}$, which is $[3, 4]$ as it was shown in Example 8.2.2. Our aim is to show that for a σ-pattern in X, its number of time-series variables in $[3, 4]$ can be estimated as the difference between the value of the surface of this σ-pattern and some other value that is a function of σ, ℓ, u and n. In order to obtain this value, we construct a time series t of length $\beta_\sigma^{\langle \ell,u,n\rangle} = 9$ satisfying all the following conditions:

1. The number of time-series variables of t that are assigned to the value $\overline{\mathcal{I}}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle}$ equals $\mu_\sigma^{\langle \ell,u,n\rangle}(\overline{\mathcal{I}}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle}) = \mu_\sigma^{\langle 0,4,9\rangle}(4) = 1$.
2. The number of time-series variables of t that are assigned to the value $\underline{\mathcal{I}}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle}$, which is $\overline{\mathcal{I}}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle} - 1$, equals $\mu_\sigma^{\langle \ell,u,n\rangle}(\underline{\mathcal{I}}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle}) = \mu_\sigma^{\langle 0,4,9\rangle}(3) = n - 2 = 7$.
3. The rest of the time-series variables of t, namely $n - \mu_\sigma^{\langle \ell,u,n\rangle}(\overline{\mathcal{I}}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle}) - \mu_\sigma^{\langle \ell,u,n\rangle}(\underline{\mathcal{I}}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle}) = 1$ time-series variable, is assigned to the value $\underline{\mathcal{I}}_{\langle g,f,\sigma\rangle}^{\langle \ell,u\rangle} - 1 = 2$.

Part (A) of Figure 8.2 illustrates a ground time series t of length 9 over $[0, 4]$ satisfying all the three conditions. By construction, the sum of elements of t is greater than or equal to the surface of any σ-pattern of X. Furthermore, for any σ-pattern of X, its number of time-series variables whose values are in $[3, 4]$ is not greater than the number of such time-series variables of t.

Part (A) of Figure 8.2 contains three type of points: circled, squared and diamond-shaped points; thus our goal is to evaluate the number of circled points. The value of X_i is one plus the number of squared and diamond-shaped points under the point corresponding to X_i. Hence, the sum of all elements of t can be viewed as the total number of circled, squared and diamond-shaped points. Furthermore, the number

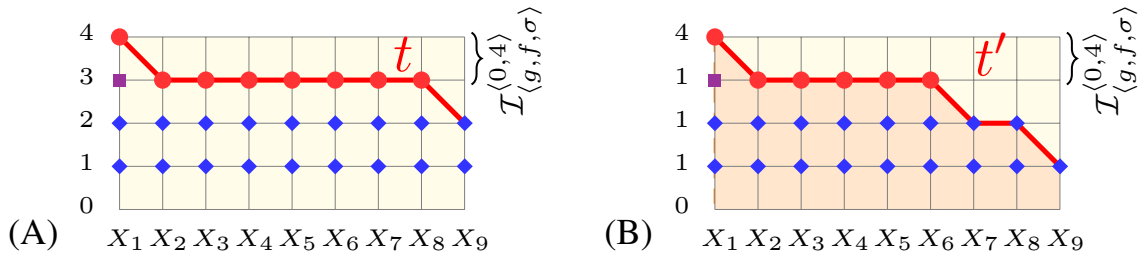


Figure 8.2 – Here, $\mathcal{I}^{\langle 0,4\rangle}_{\langle g,f,\sigma\rangle}$ is the interval of interest of $\langle g, f, \sigma\rangle$ wrt $\langle 0, 4\rangle$ for $\sigma =$ DECREASING_SEQUENCE; (A) a ground time series t, satisfying the three conditions in Example 8.2.5; (B) a ground time series t' with a single σ-pattern of surface 24.

of circled points is the difference between the total number of points and the number of squared points, namely 27 minus 19, which is 8.

For any σ-pattern of X, its corresponding number of squared and diamond-shaped points is at most 19. Then, its number of time-series variables whose values are in $[3, 4]$ can be estimated as the surface of the σ-pattern minus 19. Hence, when the surface of the σ-pattern is 24, a lower bound on $\mathcal{N}$ is 5. Part (B) of Figure 8.2 gives an example of a ground time series t' of length 9 over $[0, 4]$ that contains a σ-pattern with a surface of 24. This σ-pattern has $6 \geq 5$ values in $[3, 4]$, which agrees with our computed lower bound. △

Theorem 8.2.1 (AMONG implied constraint for MAX_SURF_σ)**.** Consider a $g_f_\sigma(X, R)$ time-series constraint with $g = \texttt{max}$, $f = \texttt{surf}$ and X being a time series of length n over an integer interval domain $[\ell, u]$; then AMONG$(\mathcal{N}, \langle X_{1+b_\sigma}, X_{2+b_\sigma}, \ldots, X_{n-a_\sigma}\rangle, \mathcal{I})$ is an implied constraint, where $\mathcal{N}$ is restricted by

$$\mathcal{N} \geq R - \max(0, \underline{\mathcal{I}} - 1) \cdot \beta - \sum_{v \in [\underline{\mathcal{I}}+1, \overline{\mathcal{I}}]} \mu_\sigma^{\langle \ell,u,n\rangle}(v) \cdot (v - \underline{\mathcal{I}}), \tag{8.1}$$

where β (respectively $\mathcal{I}$) is shorthand for $\beta_\sigma^{\langle \ell,u,n\rangle}$ (respectively $\mathcal{I}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$), and $\underline{\mathcal{I}}$ (respectively $\overline{\mathcal{I}}$) denotes the lower (respectively upper) limit of interval $\mathcal{I}$.

Proof We show that the right-hand side of the stated inequality is a lower bound on the number of time-series variables of a σ-pattern whose values are in $\mathcal{I}$, and the surface of the σ-pattern is R. Note that the first b_σ and the last a_σ time-series variables never belong to any σ-pattern, and thus we do not include them in our AMONG implied constraint.

In order to prove the lower bound on $\mathcal{N}$, we first compute a lower bound on the number $\mathcal{N}^{\underline{\mathcal{I}}}$ of time-series variables of the σ-pattern whose value is $\underline{\mathcal{I}}$, which is the smallest value of interval $\mathcal{I}$. To do so, we overestimate the value of R, namely for every $v > \underline{\mathcal{I}}$ in $\mathcal{I}$, we assume that the number of occurrences of v in the σ-pattern equals $\mu_\sigma^{\langle \ell,u,\beta\rangle}(v)$. Note that the number of time-series variables in any σ-pattern is not greater than $\beta = \beta_\sigma^{\langle \ell,u,n\rangle}$. We state the following inequality:

$$\begin{aligned} R \leq\ & \underbrace{\mathcal{N}^{\underline{\mathcal{I}}} \cdot \max(0, \underline{\mathcal{I}})}_{A} + \underbrace{\sum_{v \in [\underline{\mathcal{I}}+1, \overline{\mathcal{I}}]} \mu_\sigma^{\langle \ell,u,\beta\rangle}(v) \cdot \max(0, v)}_{B} \\ & \underbrace{+ \max(0, \underline{\mathcal{I}} - 1) \cdot \max(0, \beta - \mathcal{N}^{\underline{\mathcal{I}}} - \sum_{v \in [\underline{\mathcal{I}}+1, \overline{\mathcal{I}}]} \mu_\sigma^{\langle \ell,u,\beta\rangle}(v))}_{C}, \end{aligned} \tag{8.2}$$

where A, B, and C correspond to the sums of elements of the σ-pattern that equal $\underline{\mathcal{I}}$, are in $\mathcal{I}$ and are greater than $\underline{\mathcal{I}}$, and are outside $\mathcal{I}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle}$ respectively. From Inequality 8.2 we obtain the following lower bound on $\mathcal{N}^{\underline{\mathcal{I}}}$:

$$\mathcal{N}^{\underline{\mathcal{I}}} \geq R - \sum_{v \in S} \mu_\sigma^{\langle \ell,u,\beta \rangle}(v) \cdot \max(0, v) \\ - \max(0, \underline{\mathcal{I}} - 1) \cdot \max(0, \beta - \sum_{v \in S} \mu_\sigma^{\langle \ell,u,\beta \rangle}(v)). \tag{8.3}$$

In order to obtain a lower bound on $\mathcal{N}$ from the known lower bound on $\mathcal{N}^{\underline{\mathcal{I}}}$, we add the $\sum_{v \in S} \mu_\sigma^{\langle \ell,u,\beta \rangle}(v)$ term to the right-hand side of Inequality 8.3, regroup some terms, and obtain the inequality of the theorem. □

Example 8.2.6 (AMONG implied constraint for MAX_SURF_σ)**.** Consider the $g_f_\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint, with g being `max`, with f being `surf`, and with every X_i (with $i \in [1, n]$) ranging over the same domain $[\ell, u]$ with $u > 0$ and $u - \ell > 1$. We illustrate the derivation of AMONG implied constraints for two regular expressions.

- Consider the σ = DECREASING_SEQUENCE regular expression. In Example 8.2.2, we computed the interval of interest of MAX_SURF_σ wrt $\langle \ell, u \rangle$, which is $[u-1, u]$. In Example 8.2.3, we showed that $\mu_\sigma^{\langle \ell,u,n \rangle}(\ell) = \mu_\sigma^{\langle \ell,u,n \rangle}(u) = 1$, and for every value v in $[\ell + 1, u - 1]$, we have that $\mu_\sigma^{\langle \ell,u,n \rangle}(v)$ equals $n-2$. Finally, in Example 8.2.4 we demonstrated that when $u-\ell > 1$ and $n \geq 2$, $\beta_\sigma^{\langle \ell,u,n \rangle} = n$. By Theorem 8.2.1, we can impose the AMONG$(\mathcal{N}, X, \langle u-1, u \rangle)$ implied constraint with $\mathcal{N} \geq R - \mu_\sigma^{\langle \ell,u,n \rangle}(u) - \max(0, (\underline{\mathcal{I}}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle} - 1) \cdot \beta_\sigma^{\langle \ell,u,n \rangle}) = R - 1 - \max(0, (u-2) \cdot n)$. Turning back to Example 8.2.5 we observe that, in the obtained implied constraint, the term '1' corresponds to the number of squared points, and the term '$\max(0, (u-2) \cdot n)$' to the number of diamond-shaped points. The derived lower bound on $\mathcal{N}$ also appears in the third row of Table 8.2.
- Consider the σ = PEAK = '$< (< \mid =)^* (> \mid =)^* >$' regular expression whose values of a_σ and b_σ both equal 1. The maximum value in $[\ell, u]$ that appears in a σ-pattern is u. In addition, any maximal time series for $\langle g, f, \sigma \rangle$ contains a single σ-pattern whose values are all the same and equal u. Hence, the interval of interest of $\langle g, f, \sigma \rangle$ wrt $\langle \ell, u \rangle$ is $[u, u]$. Since both a_σ and b_σ equal 1, the smallest value in $[\ell, u]$ may not be in any σ-pattern and $\mu_\sigma^{\langle \ell,u,n \rangle}(\ell) = 0$. For any value $v \in [\ell - 1, u]$, we have $\mu_\sigma^{\langle \ell,u,n \rangle}(\ell) = n - 2$. By Theorem 8.2.3, we impose an AMONG$(\mathcal{N}, \langle X_2, X_3, \ldots, X_{n-1} \rangle, \langle u \rangle)$ implied constraint with $\mathcal{N} \geq R - \max(0, (u-1) \cdot (n-2))$. The derived lower bound on $\mathcal{N}$ also appears in the fifth row of Table 8.2. △

Table 8.2 illustrates for 6 regular expressions of Table 5.2 the corresponding intervals of interest of MAX_SURF_σ constraints wrt some integer interval domain $[\ell, u]$ such that $u > 1 \wedge u - \ell > 1$, as well as the lower bound LB on the parameter $\mathcal{N}$ of the derived AMONG constraint.

Theorem 8.2.2 (AMONG implied constraint for MIN_SURF_σ)**.** Consider a MIN_SURF_$\sigma(\langle X_1, \ldots, X_n \rangle, R)$ time-series constraint with every X_i being over an integer interval domain $[\ell, u]$. If $R < +\infty$, then the AMONG$(\mathcal{N}, \langle X_{1+b_\sigma}, X_{2+b_\sigma}, \ldots, X_{n-a_\sigma} \rangle, \mathcal{I}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle})$ constraint is an implied constraint, and $\mathcal{N}$ is restricted by the same inequality as in Theorem 8.2.1.

Proof. The implied constraint for a MIN_SURF_$\sigma(X, R)$ time-series constraint is the same as for the corresponding MAX_SURF_$\sigma(X, R)$ constraint, but can be imposed only when R does not equal the identity value of the aggregator, namely $+\infty$. □

Theorem 8.2.3 (AMONG implied constraint for SUM_SURF_σ)**.** Consider a $g_f_\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint with g = `sum`, f = `surf` and every X_i being over an integer interval domain $[\ell, u]$;

σ	$\mathcal{I}^{\langle \ell,u \rangle}_{\langle \text{MAX,SURF},\sigma \rangle}$	LB
BUMP_ON_DECREASING_SEQUENCE	$[u-2,u]$	$R-\max(0,(u-3)\cdot 3-3)$
DECREASING	$[u-1,u]$	$R-\max(0,(u-2)\cdot 2-1)$
DECREASING_SEQUENCE	$[u-1,u]$	$R-\max(0,(u-2)\cdot n-1)$
GORGE	$[u-1,u-1]$	$R-\max(0,(u-2)\cdot(n-2))$
PEAK	$[u,u]$	$R-\max(0,(u-1)\cdot(n-2))$
ZIGZAG	$[u-1,u]$	$R-\max(0,(u-2)\cdot(n-2)-\lfloor\frac{n-1}{2}\rfloor)$

Table 8.2 – Regular expression σ, the corresponding interval of interest of MAX_SURF_$\sigma(X,R)$ wrt an integer interval domain $[\ell,u]$ such that $u>1$ and $u-\ell>1$, and the lower bound LB on the parameter of the derived AMONG implied constraint. The value LB is obtained from a generic formula, which is parameterised by characteristics of regular expressions.

then AMONG$(\mathcal{N},\langle X_{1+b_\sigma},X_{2+b_\sigma},\ldots,X_{n-a_\sigma}\rangle,\mathcal{I})$ is an implied constraint, where $\mathcal{N}$ is restricted by

$$\begin{aligned}\mathcal{N} \geq R &- \max(0,\underline{\mathcal{I}}-1)\cdot\left(n-a_\sigma-b_\sigma+(p_o-1)\cdot o_\sigma^{\langle \ell,u\rangle}\right)\\ &- \sum_{v\in[\underline{\mathcal{I}}+1,\overline{\mathcal{I}}]}\mu_\sigma^{\langle \ell,u,n\rangle}(v)\cdot p_o\cdot(v-\underline{\mathcal{I}})\\ &- (p_o-1)\cdot o_\sigma^{\langle \ell,u\rangle},\end{aligned} \tag{8.4}$$

where $\mathcal{I}$ is shorthand for $\mathcal{I}^{\langle \ell,u\rangle}_{\langle g,f,\sigma\rangle}$, $\underline{\mathcal{I}}$ (respectively $\overline{\mathcal{I}}$) denotes the lower (respectively upper) limit of $\mathcal{I}$, and p_o is 1 if every maximal time series has a single σ-pattern, and is the maximum number of σ-patterns in a time series of length n over $[\ell,u]$, otherwise.

Proof. To prove Theorem 8.2.3 we consider a time series with $p\geq 0$ σ-patterns, where σ-pattern i (with $i\in[1,p]$) has a width of ω_i and a surface of R_i, and where $R=\sum_{i=1}^{p}R_i$. The proof consists of two steps:

1. First, for each σ-pattern i (with $i\in[1,p]$), we compute the minimum number $\mathcal{N}_i$ of time-series variables that must be assigned to a value within the interval of interest $\mathcal{I}$, in order to reach a surface of R_i.
2. Second, we take the sum of $\mathcal{N}_i$, and minimise the obtained value, which, in the end, will be a minimum value for $\mathcal{N}$.

First Step. We use Inequality (8.1) of Theorem 8.2.1 for a subseries X' of X of length $\omega_i'=\omega_i+a_\sigma+b_\sigma$, knowing that X' has a single σ-pattern and $\beta_\sigma^{\langle \ell,u,n\rangle}$ is ω_i. Then, by Theorem 8.2.1, we obtain the following estimation of $\mathcal{N}_i$:

$$\mathcal{N}_i \geq R_i-\omega_i\cdot\max(0,\underline{\mathcal{I}}-1)-\sum_{v\in[\underline{\mathcal{I}}+1,\overline{\mathcal{I}}]}(v-\underline{\mathcal{I}})\cdot\mu_\sigma^{\langle \ell,u,\omega_i'\rangle}(v). \tag{8.5}$$

Second Step. We obtain the minimum value of $\mathcal{N}$, by taking the sum of the derived minimum values for $\mathcal{N}_i$ over all the values of i:

$$\mathcal{N}=\sum_{i=1}^{p}\mathcal{N}_i\geq\sum_{i=1}^{p}(R_i-A_i-B_i)-C=R-\sum_{i=1}^{p}A_i-\sum_{i=1}^{p}B_i-C, \tag{8.6}$$

where $A_i=\omega_i\cdot\max(0,\underline{\mathcal{I}}-1),\ \forall i\in[1,p]$,
$B_i=\sum_{v\in[\underline{\mathcal{I}}+1,\overline{\mathcal{I}}]}\mu_\sigma^{\langle \ell,u,\omega_i'\rangle}(v)\cdot(v-\underline{\mathcal{I}}),\ \forall i\in[1,p]$,
$C=(p-1)\cdot o_\sigma^{\langle \ell,u\rangle}$.

σ	$\mathcal{I}^{\langle \ell,u \rangle}_{\langle \text{SUM,SURF},\sigma \rangle}$	LB
BUMP_ON_DECREASING_SEQUENCE	$[u-2,u]$	$R - \max(0,(u-2)\cdot 3\cdot \lfloor \frac{n-3}{3} \rfloor)$
DECREASING	$[u-1,u]$	$R - \max(0,(u-2)\cdot 2 - 1)$
DECREASING_SEQUENCE	$[u-1,u]$	$R - \max(0,(u-2)\cdot n - \lfloor \frac{n}{2} \rfloor)$
GORGE	$[u-1,u-1]$	$R - \max(0,(u-2)\cdot (n-2))$
PEAK	$[u,u]$	$R - \max(0,(u-1)\cdot (n-2))$
ZIGZAG	$[u-1,u]$	$R - \max(0,(u-2)\cdot (n-2) - \lfloor \frac{n-1}{2} \rfloor)$

Table 8.3 – Regular expression σ, the corresponding interval of interest of SUM_SURF_$\sigma(X,R)$ wrt an integer interval domain $[\ell,u]$ such that $u > 1$ and $u - \ell > 1$, and the lower bound LB on the parameter of the derived AMONG implied constraint. The value LB is obtained from a generic formula, which is parameterised by characteristics of regular expressions.

The terms A_i and B_i come from Inequality 8.5 and the term C is used because some variables may belong to two σ-patterns: in order to not count them twice we subtract a correction term. Let A (respectively B) denote $\sum_{i=1}^{p} A_i$ (respectively $\sum_{i=1}^{p} B_i$). In order to satisfy Condition 8.6, we need to find the upper bounds on the sum $A + B + C$ by choosing the value of p, and the sum of σ-patterns lengths. We consider two cases, but any additional information may be used for a more accurate estimation of these parameters:

- [EVERY MAXIMAL TIME SERIES HAS A SINGLE σ-PATTERN] Then, the maximum value of $A + B + C$ is reached for p being 1, and $\sum_{i=1}^{p} \omega_i$ being $n - b_\sigma - a_\sigma$. It implies that for any $v \in [\underline{\mathcal{I}}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle} + 1, \overline{\mathcal{I}}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle}]$, the value of $\sum_{i \in [1,p]} \mu_\sigma^{\langle \ell,u,\omega'_i \rangle}(v)$ equals $\mu_\sigma^{\langle \ell,u,\omega'_i \rangle}(v)$.
- [THERE IS AT LEAST ONE MAXIMAL TIME SERIES WITH MORE THAN ONE σ-PATTERN] We give an overestimation: we assign the value of p to its maximum value, which depends on σ, the value of $\sum_{i=1}^{p} \omega_i$ is overestimated by $n - a_\sigma - b_\sigma + p_o \cdot o_\sigma^{\langle \ell,u \rangle}$, and the value of $\sum_{i \in [1,p]} \mu_\sigma^{\langle \ell,u,\omega'_i \rangle}(v)$ is overestimated by $\mu_\sigma^{\langle \ell,u,n \rangle}(v) \cdot p$.

Hence, we obtain a lower bound for $\mathcal{N}$, which is the right-hand side of the inequality stated by Theorem 8.2.3. □

Example 8.2.7 (AMONG implied constraint for SUM_SURF_σ)**.** Consider a $g_f_\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint, with g being sum, with f being surf and with every X_i (with $i \in [1,n]$) ranging over the same domain $[\ell,u]$ with $u > 0$ and $u - \ell > 0$. We illustrate the derivation of AMONG implied constraints for two regular expressions.

- Consider the σ = DECREASING_SEQUENCE regular expression. In Example 8.2.2, we found that the interval of interest of $\langle g,f,\sigma \rangle$ wrt $\langle \ell,u \rangle$ is $[u-1,u]$, and in Example 8.2.3, we showed that $\mu_\sigma^{\langle \ell,u,n \rangle}(\ell) = \mu_\sigma^{\langle \ell,u,n \rangle}(u) = 1$, and for every value v in $[\ell+1, u-1]$, we have that $\mu_\sigma^{\langle \ell,u,n \rangle}(v)$ equals $n-2$. Every maximal time series for SUM_SURF_σ contains the maximum number of σ-patterns. Hence the value of p_o is the maximum number of decreasing sequences in a time series of length n, which is $\lfloor \frac{n}{2} \rfloor$. By Theorem 8.2.3, we impose an AMONG$(\mathcal{N}, \langle X_1, X_2, \ldots, X_n \rangle, \langle u-1, u \rangle)$ implied constraint with $\mathcal{N} \geq R - \lfloor \frac{n}{2} \rfloor - \max(0,(u-2)\cdot n)$. The derived lower bound on $\mathcal{N}$ also appears in the third row of Table 8.3.
- Consider the σ = PEAK = '$< (< \mid =)^* (> \mid =)^* >$' regular expression introduced in Example 8.2.6. The maximum value in $[\ell,u]$ that occurs in a σ-pattern is u. In addition, any maximal time series for $\langle g,f,\sigma \rangle$ contains a single σ-pattern whose values are all the same and equal u. Hence, the interval of interest of $\langle g,f,\sigma \rangle$ wrt $\langle \ell,u \rangle$ is $[u,u]$, and the value of p_o equals 1. We showed in Example 8.2.6 that $\mu_\sigma^{\langle \ell,u,n \rangle}(\ell) = 0$ and for any $v \in [\ell,u]$, we have $\mu_\sigma^{\langle \ell,u,n \rangle}(v) = n-2$. Furthermore, any two σ-patterns never have common time-series variables, thus the value of $o_\sigma^{\langle \ell,u \rangle}$

equals 0. By Theorem 8.2.3, we impose an AMONG$(\mathcal{N}, \langle X_2,\ X_3, \ldots, X_{n-1}\rangle,\ \langle u\rangle)$ implied constraint with $\mathcal{N} \geq R - \max(0, (u-1)\cdot(n-2))$. The derived lower bound on $\mathcal{N}$ also appears in the fifth row of Table 8.3. △

Table 8.3 illustrates for 6 regular expressions of Table 5.2 the corresponding intervals of interest of SUM_SURF_σ constraints wrt some integer interval domain $[\ell, u]$ such that $u > 1 \ \wedge\ u - \ell > 1$, as well as the lower bound LB on the parameter $\mathcal{N}$ of the derived AMONG implied constraint.

8.3 Conclusion

Using five regular-expression characteristics, we have defined a single per family generic AMONG implied constraint for all constraints of the MAX_SURF_σ, MIN_SURF_σ, and SUM_SURF_σ families. Two of the used characteristics, namely the height and the overlap, were also required for deriving sharp bounds on the result values of time-series constraints, presented in Chapter 7.

Summary of this Chapter:

The main contribution of this chapter is an AMONG implied constraint parameterised by regular-expression characteristics, one per each of three families of time-series constraints with the `surf` feature.

Chapter 9

Synthesising Parameterised Linear Invariants

This chapter is an extended version of an article published in the proceedings of the *CP'17* conference [13]. The final authenticated version of this article is available online at: `http://dx.doi.org/10.1007/978-3-319-66158-2_2`.

We present a systematic method for deriving linear invariants for a conjunction of global constraints that are each represented by a register automaton [29]. Since they do not encode explicitly all potential values of registers as states, register automata 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 global constraints as the intersection of the corresponding register automata [98, 97], see Definition 3.2.3, i.e. the intersection of the languages accepted by all register automata, without representing explicitly the Cartesian product of all register values. As a consequence, the size of such an intersection register automaton is often quite compact, even if maintaining domain consistency for such constraints is in general NP-hard [27]; for instance, the intersection of the 22 register automata for all NB_σ time-series constraints has only 16 states. The contributions of this chapter are twofold:

- First, Sections 9.1, 9.2 and 9.3 provide the basis of a simple, systematic method to precompute necessary conditions for a conjunction of AUTOMATON constraints on the same sequence. Each necessary condition is a linear inequality involving the result variables of the different register automata, representing the fact that the result variables cannot vary independently. These inequalities are parametrised by the sequence length and are independent of the domains of the sequence variables.

- Second, within the context of the time-series constraints, Chapter 15 of Part III shows that the method allows to precompute in less than five minutes a database of 7755 invariants that significantly speed up the search for time series satisfying multiple time-series constraints.

Adding implied constraints to a constraint model has been recognised from the very beginning of Constraint Programming as a major source of improvement [61]. Attempts to generate such implied constraints in a systematic way were limited (1) by the difficulty to manually prove a large number of conjectures [77, 24], (2) by the limitations of automatic proof systems [71, 52], or (3) to special cases for very few constraints like ALLDIFFERENT, CARDINALITY, ELEMENT [90, 7, 79]. Within the context of register automata, linear invariants relating consecutive register values of a same constraint were obtained [67] using Farkas' lemma [45] in a resource-intensive procedure.

9.1 Generating Linear Invariants

Consider k register automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ over the same alphabet Σ. Let r_i denote the number of registers 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

$$e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i \geq 0 \text{ with } e, e_0, e_1, \ldots, e_k \in \mathbb{Z}, \tag{9.1}$$

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

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

Property 9.1.1 (incremental-automaton property)**.** A register automaton $\mathcal{M}$ with r registers has the *incremental-automaton* property if the following conditions are all satisfied:

1. For every register A_j of $\mathcal{M}$, its initial value α_j^0 is a natural number.
2. For every register 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 $\alpha_{j,1}^t, \alpha_{j,2}^t, \ldots, \alpha_{j,r}^t \in \{0, 1\}$.
3. The register A_r is called the *main register* and verifies all the following three conditions:
   (a) the value returned by $\mathcal{M}$ is the last value of its main register A_r,
   (b) for every transition t of $\mathcal{M}$, $\alpha_{r,r}^t = 1$,
   (c) for a non-empty subset T of transitions of $\mathcal{M}$, $\sum_{i=1}^{r-1} \alpha_{r,i}^t > 0,\ \forall t \in T$.
4. For all other registers A_j with $j < r$, on every transition t of $\mathcal{M}$, we have $\sum_{i=1, i \neq j}^{r} \alpha_{j,i}^t = 0$ and if $\alpha_{r,j}^t > 0$, then $\alpha_{j,j}^t$ is 0.

The intuition behind the incremental-automaton property is that there is one register that we name the *main register*, whose last value is the final value, returned by the register automaton, (see 3a). At some transitions, the update of the main register is a linear combination of the other registers, while on the other transitions its value either does not change or incremented by a non-negative constant, (see 3b and 3c). All other registers 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 registers are called *potential registers*. Both register automata in Parts (A) and (B) of Figure 9.1 have the incremental-automaton property, and their single registers are the main registers. Volumes I and II of the global constraint catalogue contain more than 50 such register automata. In particular, the register 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 paper we assume that all register automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ have the incremental-automaton property.

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

1. Construct a non-negative function $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$, which represents the left-hand side of the sought linear invariant (see Section 9.1.1).
2. Select the coefficients $e_0, e_1, \ldots, e_k$, called the *relative coefficients* of the linear invariant, so that there exists a constant C such that $e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i \geq C$ (see Section 9.1.2).
3. Compute C and set the coefficient e, called the *constant term* of the linear invariant, to $-C$ (see Section 9.1.3).

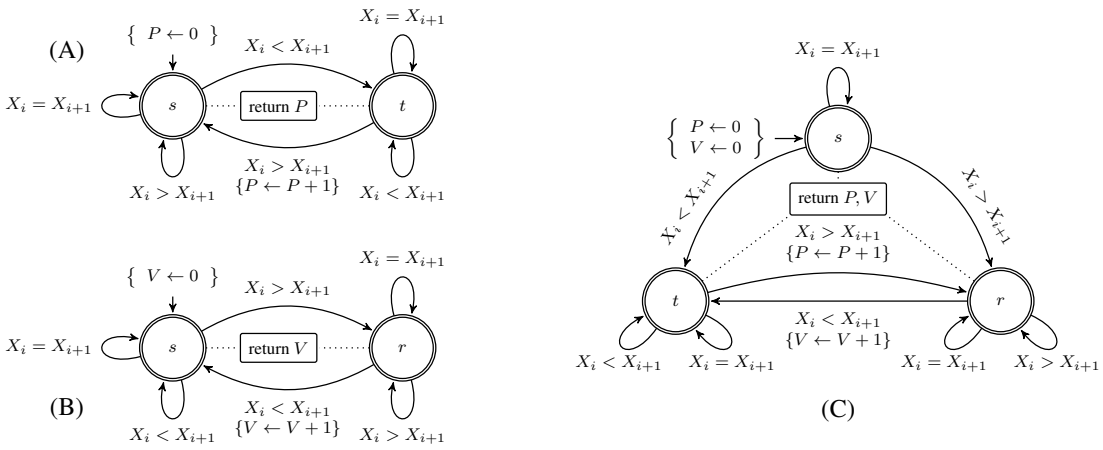


Figure 9.1 – (A) Register automaton for NB_PEAK. (B) Register automaton for NB_VALLEY. (C) Intersection of (A) and (B).

The three previous steps are performed as follows:

1. First, we assume a sign for each coefficient e_i (with $i \in [0, k]$), which tells whether we have to consider or not the contribution of the potential registers; note that each combination of signs of the coefficients e_i (with $i \in [0, k]$) will lead to a different linear invariant. Then, from the intersection $\mathcal{I}$ of $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$, 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 $e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$ while triggering t.
2. Second, we find the coefficients e_i (with $i \in [0, k]$) so that the invariant digraph does not contain any negative cycles. When the invariant digraph has no negative cycles, the value of $e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$ 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 $\mathcal{I}$.

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

First, Definition 9.1.1 introduces the notion of *invariant digraph* $G^v_{\mathcal{I}}$ of the register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ wrt a linear function v involving the values returned by these register automata. Second, Definition 9.1.2 introduces the notion of *weight of an accepting sequence* X wrt $\mathcal{I}$ in $G^v_{\mathcal{I}}$, which makes the link between a path in $G^v_{\mathcal{I}}$ and the vector of values returned by $\mathcal{I}$ after consuming the signature of X. Finally, Theorem 9.1.1 shows that the weight of X in $G^v_{\mathcal{I}}$ is a lower bound on the linear function v.

Definition 9.1.1 (invariant digraph)**.** Consider an accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt the register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$, and a linear function $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$, where $(R_1, R_2, \ldots, R_k)$ 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 $G^v_{\mathcal{I}}$ is a weighted digraph defined in the following way:

- The set of nodes of $G^v_{\mathcal{I}}$ is the set of states of $\mathcal{I}$.
- The set of arcs of $G^v_{\mathcal{I}}$ 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 $e_0 + \sum_{i=1}^{k} e_i \cdot \beta^t_i$, where β^t_i is defined

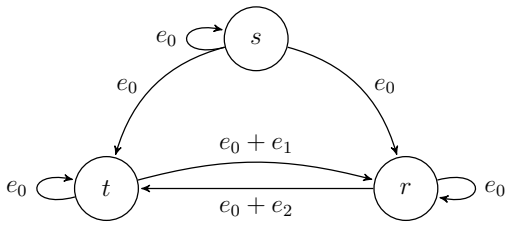


Figure 9.2 – Invariant digraph for NB_PEAK and NB_VALLEY wrt $e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V$

as follows:

$$\beta_i^t = \begin{cases} \alpha_{i,r_i,0}^t & \text{if } e_i \geq 0, \quad (9.2) \\ \sum_{j=1}^{r_i} \alpha_{i,j,0}^t & \text{if } e_i < 0, \quad (9.3) \end{cases}$$

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

Definition 9.1.2 (walk and weight of an accepting sequence)**.** Consider an accepting sequence X of length n wrt the register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$, and a linear function $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$, where $(R_1, R_2, \ldots, R_k)$ 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 registers and is called the *initialisation weight* in $G_{\mathcal{I}}^v$. It equals $e + e_0 \cdot (p-1) + \sum_{i=1}^{k} e_i \cdot \beta_i^0$, where p is the arity of the signature, and where β_i^0 is defined as follows:

$$\beta_i^0 = \begin{cases} \alpha_{i,r_i}^0 & \text{if } e_i \geq 0, \quad (9.4) \\ \sum_{j=1}^{r_i} \alpha_{i,j}^0 & \text{if } e_i < 0, \quad (9.5) \end{cases}$$

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

Example 9.1.1 (invariant digraph, weight of an accepting sequence)**.** Consider the NB_PEAK(X, P) and NB_VALLEY(X, V) time-series constraints with X being a time series of length n. Figure 9.1 gives the register 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 $X = \langle X_1, X_2, \ldots, X_n \rangle$, $\mathcal{I}$ returns a pair of values (P, V), which are the number of peaks (respectively 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 Figure 9.2. Since neither register automaton has any potential register, 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 $e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V$. $\triangle$

Theorem 9.1.1 (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 register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$, and a linear function $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$, where $(R_1, R_2, \ldots, R_k)$ is the vector of values return by $\mathcal{I}$. Then, the weight of X in $G^v_{\mathcal{I}}$ is less than or equal to $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$.

Proof. Since, when doing the intersection of register automata we do not merge registers, the registers of $\mathcal{I}$ that come from different register automata $\mathcal{M}_i$ and $\mathcal{M}_j$ do not interact, i.e. their updates are independent, hence the returned values of $\mathcal{M}_i$ and $\mathcal{M}_j$ are independent. By definition of the invariant digraph, the weight of any of its arc is $e_0 + \sum_{i=1}^{k} e_i \cdot \beta_i^t$, 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 $G^v_{\mathcal{I}}$ is the constant $e + e_0 \cdot (p-1) + \sum_{i=1}^{k} e_i \cdot \beta_i^0$ (see Definition 9.1.2) plus the weight of the walk of X, which is in total $e + e_0 \cdot (p-1) + \sum_{i=1}^{k} e_i \cdot \beta_i^0 + e_0 \cdot (n-p+1) + \sum_{j=1}^{n-p+1} \sum_{i=1}^{k} e_i \cdot \beta_i^{t_j} = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot \left(\beta_i^0 + \sum_{j=1}^{n-p+1} \beta_i^{t_j} \right)$, 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 $G^v_{\mathcal{I}}$ is less than or equal to $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$.

Consider the $v_i = e_i \cdot R_i$ linear function. We show that the weight of X in $G^{v_i}_{\mathcal{I}}$, 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 $G^{v_i}_{\mathcal{I}}$ is e_i multiplied by $\alpha^t_{r_i,0}$, where t is the corresponding transition in $\mathcal{I}$, and r_i is the main register of $\mathcal{M}_i$ (see Case 9.2 of Definition 9.1.1). If, on transition t, some potential registers of $\mathcal{M}_i$ are incremented by a positive constant, the real contribution of the register updates on this transition to R_i is at least $\alpha^t_{r_i,0}$ since $e_i \geq 0$. The same reasoning applies to the contribution of the initial values of the potential registers to the final value R_i. Since this contribution is non-negative, it is ignored, and $\beta_i^0 = \alpha_j^0$ (see Case 9.2 of Definition 9.1.2). Hence $e_i \cdot (\beta_i^0 + \sum_{j=1}^{n-p+1} \beta_i^{t_j}) = e_i \cdot (\alpha^0_{r_i} + \sum_{j=1}^{n-p+1} \alpha^t_{r_i,0}) \leq e_i \cdot R_i$.

Case 2: $\mathbf{e_i < 0}$. In this case, the weight of every arc of $G^{v_i}_{\mathcal{I}}$ is e_i multiplied by the sum of the non-negative constants, which come from the updates of *every* register of $\mathcal{M}_i$ (see Case 9.5 of Definition 9.1.1). The contribution of the potential registers 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 registers to the returned value R_i. Since the initial values of the potential registers are non-negative, and $e_i < 0$, in order to obtain a lower bound on v we assume that the initial values of the potential registers always contribute to R_i (see Case 9.3 of Definition 9.1.2). 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 register automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ do not have potential registers, then for every accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ and for any linear function $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$, the weight of X in $G^v_{\mathcal{I}}$ is *equal* to v. If there is at least one potential register

for at least one register automaton $\mathcal{M}_i$, then there may exist an accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ whose weight in $G^v_{\mathcal{I}}$ is strictly less than v.

9.1.2 Finding the Relative Coefficients of the Linear Invariant

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

For any accepting sequence X wrt $\mathcal{I}$, by Theorem 9.1.1, we have that the weight w of X in $G^v_{\mathcal{I}}$ 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 $e_0, e_1, \ldots, e_k$; 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 $G^v_{\mathcal{I}}$ does not contain *any* negative cycles. Let $\mathcal{C}$ denote the set of all simple circuits of $G^v_{\mathcal{I}}$, and let w_e denote the weight of an arc e of $G^v_{\mathcal{I}}$. In order to prevent negative cycles in $G^v_{\mathcal{I}}$, we solve the following minimisation problem, parameterised by $(s_0, s_1, \ldots s_k)$, the signs of $e_0, e_1, \ldots, e_k$:

$$\text{minimise} \quad \sum_{c \in \mathcal{C}} W_c + \sum_{i=1}^{k} |e_i| \tag{9.6}$$

$$\text{subject to} \quad W_c = \sum_{e \in c} w_e \qquad \forall c \in \mathcal{C} \tag{9.7}$$

$$W_c \geq 0 \qquad \forall c \in \mathcal{C} \tag{9.8}$$

$$s_i = \text{`}-\text{'} \Rightarrow e_i \leq 0, \;\; s_i = \text{`}+\text{'} \Rightarrow e_i \geq 0 \qquad \forall i \in [0, k] \tag{9.9}$$

$$e_i \neq 0 \qquad \forall i \in [1, k] \tag{9.10}$$

In order to obtain the coefficients $e_0, e_1, \ldots, e_k$ so that $G^v_{\mathcal{I}}$ does not contain any negative cycles, it is enough to find a solution to the satisfaction problem (9.7)-(9.10). Minimisation is required to obtaining linear invariants that eliminate as many infeasible values of $(R_1, R_2, \ldots, R_k)$ as possible. Within the objective function (9.6), the term $\sum_{c \in \mathcal{C}} W_c$ is for minimising the weight of every simple circuit, while the term $\sum_{i=1}^{k} |e_i|$ is for obtaining the coefficients with the smallest absolute value. By changing the sign vector $(s_0, s_1, \ldots s_k)$ we obtain different linear invariants.

Example 9.1.2 (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 register 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 9.1.1. 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:

$$\begin{aligned}
&\text{minimise} && \sum_{j=1}^{4} W_j + \sum_{i=0}^{2} |e_i| && \\
&\text{subject to} && W_j = e_0, && \forall j \in [1, 3] \\
& && W_4 = e_0 + e_1 + e_2 && \\
& && W_j \geq 0 && \forall j \in [1, 4] \\
& && s_i = \text{`}-\text{'} \Rightarrow e_i \leq 0, \;\; s_i = \text{`}+\text{'} \Rightarrow e_i \geq 0 && \forall i \in [0, 2] \\
& && e_i \neq 0 && \forall i \in [1, 2]
\end{aligned} \tag{9.11}$$

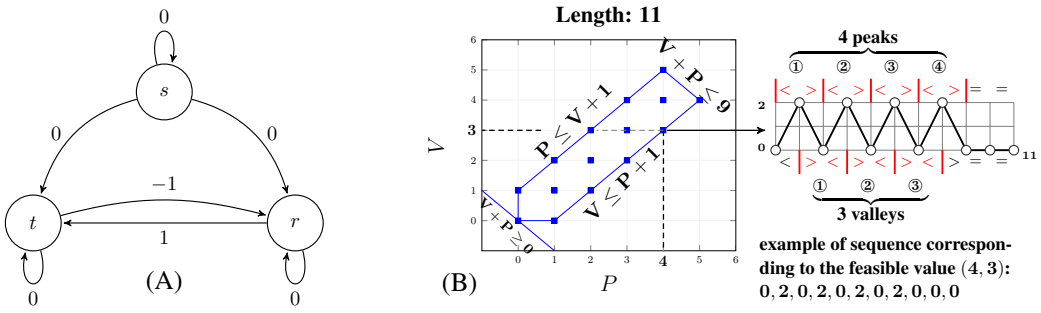


Figure 9.3 – (A) The invariant digraph of the register automata for the NB_PEAK and the NB_VALLEY time-series constraints. (B) The set of feasible values of the result variables P and V of the NB_PEAK and the NB_VALLEY time-series constraints, respectively, for sequences of length 11.

Note that the value of e_0 must be non-negative otherwise (9.11) 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 ans 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)	$(1,-1,-1)$	$(0,-1,1)$	$(0,1,-1)$	$(0,1,1)$

△

9.1.3 Finding the Constant Term of the Linear Invariant

Finally, we focus on finding the constant term e of the linear invariant $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i \geq 0$, when the coefficients $e_0, e_1, \ldots, e_k$ are known, and when the digraph of the register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ wrt v does not contain any negative cycles. By Theorem 9.1.1, 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 $e_0 \cdot (p-1) - \sum_{i=1}^{k} \beta_i^0$ (see Definition 9.1.2) 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 9.1.3 (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 9.1.2, 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 9.3. We compute the length of a shortest path from the node s, which corresponds to the initial state of the register automaton in Part (C) of Figure 9.1 to every node corresponding to the accepting state of the register automaton in Part (C) of Figure 9.1. 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 the following linear invariants: $V \leq P + 1$, $V + P \leq n - 2$, $V + P \geq 0$.

Part (B) of Figure 9.3 gives the polytope of feasible points (P, V) when n is 11. Observe that three of the four found linear invariants are facets of the convex hull of this polytope, which implies that these linear invariants are sharp. △

Example 9.1.4 (generating invariants for non-time-series constraints)**.** We illustrate how the method presented in this section can also be used for generating linear invariants for non time-series constraints.

Consider a sequence of integer variables $X = \langle X_1, X_2, \ldots, X_n \rangle$ with every X_i ranging over $[0, 3]$, four AMONG [25] constraints that restrict the variables R_0, R_1, R_2, R_3 to be the number of occurrences of values $0, 1, 2, 3$, respectively, in X, as well as the four corresponding STRETCH [108] constraints restricting the stretch length in X to be respectively in $[1, 4]$, $[2, 5]$, $[3, 5]$, and $[1, 2]$. In addition assume that value 2 (respectively 1) cannot immediately follow a 3 (respectively 2). The intersection of the corresponding register automata has 17 states and allows to generate 16 linear invariants, one of them being $2 + n + R_0 + R_1 - R_2 - 2 \cdot R_3 \geq 0$. Since the sum of all R_i is n, this linear invariant can be simplified to $2 + 2 \cdot n - 2 \cdot R_2 - 3 \cdot R_3 \geq 0$, which is equivalent to $2 \cdot (R_2 + R_3 - n) \leq 2 - R_3$. This inequality means that if X consists only of the values 2 and 3, i.e. $R_2 + R_3 - n = 0$, then $R_3 \leq 2$, which represents the conjunction of the conditions that the stretch length of $R_3 \in [1, 2]$ and $(X_i = 3) \Rightarrow (X_{i+1} \neq 2)$. △

9.2 Improving the Generated Linear Invariants

When at least one of the register automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ has at least one potential register, then there may exist an accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ such that the weight of X in the invariant digraph $G^v_{\mathcal{I}}$ is strictly less than $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$. This may lead to weaker invariants and Example 9.2.1 illustrates such a situation.

Example 9.2.1 (weak generated invariants)**.** Consider the NB_DECREASING_TERRACE(X, R_1) and the SUM_WIDTH DECREASING_TERRACE(X, R_2) constraints 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 register automata for these two constraints is given in Figure 9.4. By inspection we can derive the invariant $R_2 \geq 2 \cdot R_1$, which cannot be generated with our method, described in Section 9.1, because of the following reason: when $e_0 = 0$, $e_1 = -2$, and $e_2 = 1$, the weight of the arc from b to c is e_0, and the weight of the arcs from c to b is $e_0 + e_1 + e_2$, and thus the weight of the cycle $b - c - b$ is $2 \cdot e_0 + e_1 + e_2 = -1$.

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

To handle the issue presented in Example 9.2.1 we introduce in Section 9.2.1 a *preprocessing technique* of the intersection of register automata. It relies on the notion of *delay* of a potential register A at a state q of the intersection $\mathcal{I}$, which is a lower bound on the value of A when the register automaton arrives to state q. Intuitively, we can change the updates of some registers 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 will have larger weights.

9.2.1 Preprocessing Technique of the Intersection of Register Automata

In this section, we describe a preprocessing technique that we will allow us to obtain better linear invariants in the situation described in Example 9.2.1.

Consider register automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ whose final values are $R_1, R_2, \ldots, R_k$, respectively. In this section, we show how to modify the register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ so that the obtained register automaton $\mathcal{I}^*$ satisfies the three following conditions:

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

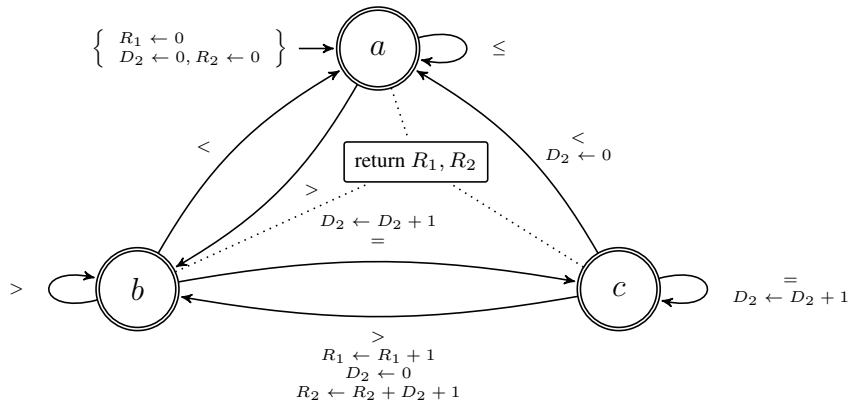


Figure 9.4 – Intersection of register automata for NB_DECREASING_TERRACE and SUM_WIDTH_DECREASING_TERRACE, for which our method does not generate sharp linear invariants.

2. For every accepting sequence X wrt $\mathcal{I}$, the register 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 $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$.

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 will lead to stronger invariants. To obtain such register automaton $\mathcal{I}^*$, we first introduce in Definition 9.2.1 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 a one dimensional matrix whose values correspond to the potential registers of $\mathcal{M}_i$. The value j of this matrix represents a lower bound on the value of the register of $\mathcal{I}$ corresponding the potential register j of $\mathcal{M}_i$ when the register automaton $\mathcal{I}$ arrives to the state q. Further, based on this notion, in Definition 9.2.2, we introduce the notion of *delayed intersection*. Finally, in Theorem 9.2.1 we show that the delayed intersection satisfies Conditions 1, 2, and 3.

Definition 9.2.1 (list of delays of a state)**.** Consider a register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$. The *list of delays* d_q of a state q is a list of one dimensional matrices, where the length of the matrix at the position i in d_q is the number of potential registers in the register automaton $\mathcal{M}_i$. 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 $d_q[i][j]$ of this matrix 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 register A_j (resp. the free term) in the update of A_j in the automaton $\mathcal{M}_i$.

Example 9.2.2 (list of delays of a state)**.** Consider two register automata $\mathcal{M}_1$ and $\mathcal{M}_2$ such that their intersection $\mathcal{I}$ is given in Figure 9.4. The register automaton $\mathcal{M}_1$ has one register R_1, and $\mathcal{M}_2$ has two registers 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 registers then for any state q of $\mathcal{I}$, the matrix $d_q[1]$ is empty. The following table gives the list of delays of every potential register of $\mathcal{I}$.

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

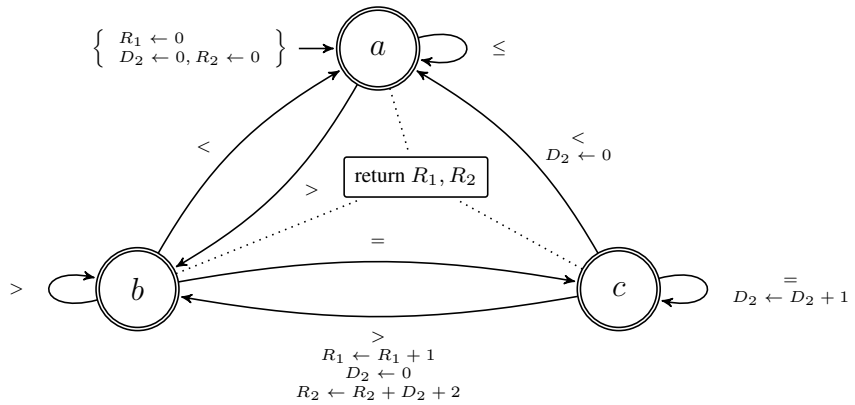


Figure 9.5 – Delayed intersection obtained from the intersection in Figure 9.4

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

Definition 9.2.2 (delayed intersection)**.** Consider the register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$. The delayed intersection $\mathcal{I}^*$ of $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ 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 registers of $\mathcal{I}^*$ is the same as for $\mathcal{I}^*$, and is denoted by r.
- The initial values of main registers of $\mathcal{I}^*$ are the same as for $\mathcal{I}^*$. For every potential register $A^*_{i,j}$ 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 register $\mathcal{M}_{i,j}$ 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 register $\mathcal{M}^*_{i,j}$ 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 register 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 registers of the register automaton $\mathcal{M}_i$.
  - If $A_{i,j}$ is a potential register 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 9.2.3 (delayed intersection)**.** Consider two register automata $\mathcal{M}_1$ and $\mathcal{M}_2$ from Example 9.2.2 and their intersection $\mathcal{I}$, which is given in Figure 9.4. The delayed intersection $\mathcal{I}^*$ constructed according to Definition 9.2.2 was given in Figure 9.5. The main difference between $\mathcal{I}^*$ and $\mathcal{I}$ is that the register 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 b. △

Theorem 9.2.1 (properties of delayed intersection)**.** Consider the register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$. 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 register 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 $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$.

Proof. We prove each of the three statements separately.

[Proof of (1)]. Since $\mathcal{I}$ have the sames 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 registers, we will show that after consuming the signature S of any accepting sequence, the main registers 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 register automaton $\mathcal{I}^*$ triggered one transition t from its initial state q to some other state q'. Then, let us consider a main register 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 register $A^*_{i,k}$ 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 register $A_{i,j}$ 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 registers of $\mathcal{I}^*$ contain the same values as the main register 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 registers 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 assumption of induction 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 register $A_{i,k}$ of $\mathcal{I}$ equals $A^*_{i,k} + d_q[i][k]$. This can be also 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 $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i$. 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 $e + e_0 \cdot (p-1) + \sum_{i=1}^{k} e_i \cdot \beta^0_i$ (see Definition 9.1.2) plus the weight of the walk of X, which is in total $e + e_0 \cdot (p-1) + \sum_{i=1}^{k} e_i \cdot \beta^0_i + e_0 \cdot (n-p+1) + \sum_{j=1}^{n-p+1} \sum_{i=1}^{k} e_i \cdot \beta^{t_j}_i = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot \left(\beta^0_i + \sum_{j=1}^{n-p+1} \beta^{t_j}_i \right)$, 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 $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot \left(\delta^0_i + \sum_{j=1}^{n-p+1} \delta^{t_j}_i \right)$, 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 9.1.1, the weight of every arc of $G^v_{\mathcal{I}}$ (respectively $G^v_{\mathcal{I}^*}$), corresponding to a transition t of $\mathcal{I}$, (respectively $\mathcal{I}^*$) is equal to $\sum_{i=1}^{k} e_i \cdot \beta^t_i$ (respectively $\sum_{i=1}^{k} e_i \cdot \delta^t_i$).

As in Theorem 9.1.1, we consider the function $v_i = e_i \cdot R_i$. Depending on the sign of e_i we consider two cases.

Case (1): $e_i \geq 0$**.** Then, the weight of X in $G^{v_i}_{\mathcal{I}}$ (respectively $G^{v_i}_{\mathcal{I}^*}$) is equal to $e_i \cdot \alpha$ (respectively

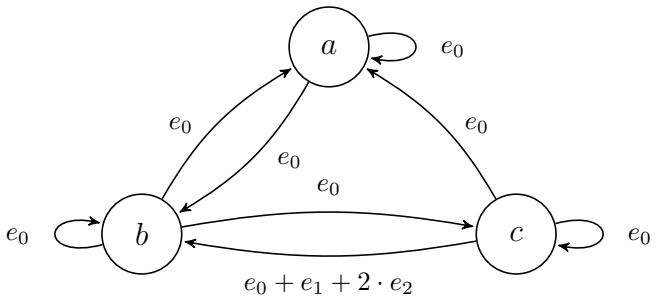


Figure 9.6 – Invariant digraph obtained from the delayed intersection in Figure 9.5

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

Note that in the register automaton $\mathcal{I}^*$, all the constants $\gamma_{i,j,0}^t$ are non-negative by definition of the delay (see Definition 9.2.1). It means that the reasoning used in the proof of Theorem 9.1.1 requiring the non-negativity of these constants remains valid for the invariant digraph $G_{\mathcal{I}^*}^v$.

Example 9.2.4 (generating stronger invariants)**.** Consider two register automata $\mathcal{M}_1$ and $\mathcal{M}_2$ from Example 9.2.2. Their intersection $\mathcal{I}$ is given in Figure 9.4, and their delayed intersection $\mathcal{I}^*$ is given in Figure 9.5. The invariant digraph $G_{\mathcal{I}^*}^v$ is given in Figure 9.6 when $e_0 > 0$, $e_1 > 0$, and $e_2 < 0$. By stating the minimisation problem from Section 9.1.2, 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 $2 \cdot R_1 \geq R_2$, which could not be found with the invariant digraph $G_{\mathcal{I}}^v$. △

9.3 Generating Additional Invariants

In Section 9.1, we presented a systematic method for generating linear invariants linking the values returned by a register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ after consuming the signature of the same accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I}$. In this section, we present several cases where the same method can be used for generating additional non-linear invariants.

Quite often a register automaton $\mathcal{M}_i$ (with i in $[1, k]$) returns its initial value only when the signature of X does not contain any occurrence of some regular expression σ_i. This may lead to a convex hull of points of coordinates $(R_1, R_2, \ldots, R_k)$ returned by $\mathcal{I}$ containing infeasible points, e.g. see Part (A) of Figure 9.7. Some of these infeasible points can be eliminated by stronger invariants subject to the condition, called the

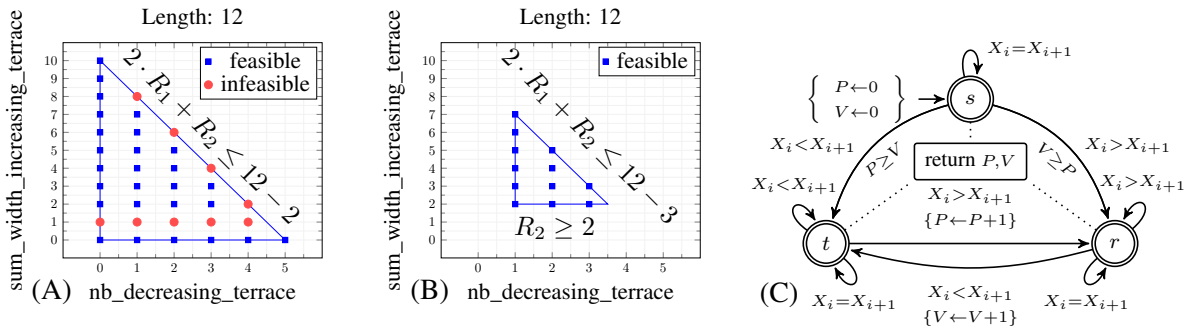


Figure 9.7 – Invariants on the result values R_1 and R_2 of NB_DECREASING_TERRACE and SUM_WIDTH_INCREASING_TERRACE for a sequence length of 12 (A) with the general linear invariants, and (B) with the Non-Default Value condition. (C) Intersection for NB_PEAK and NB_VALLEY with the guards $P \geq V$ and $V \geq P$ on transitions $s \to t$ and $s \to r$ (as for the return statement, the P and V register in the guards refer to the final values of the corresponding registers).

non-default value condition, that no variable of the returned vector is assigned to the initial value of the corresponding register. Section 9.3.1 shows how to generate such invariants. Section 9.3.2 introduces the notion of *guard* of a transition t of $\mathcal{I}$, a linear inequality of the form $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i \geq 0$, which is a *necessary condition* on the vector of values returned by $\mathcal{I}$ after consuming X for triggering the transition t upon consuming X. We call such a necessary condition a *linear guard invariant*.

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

We first illustrate the motivation for such conditional linear invariants.

Example 9.3.1 (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 9.7, 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 9.7. 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 [10, p. 2598]. △

Consider that each register automaton $\mathcal{M}_i$ (with i in $[1, k]$) returns its initial value 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 register automaton which accepts the words of the language $\Sigma^*\sigma_i\Sigma^*$, where Σ^* denotes any word over Σ. Then, using the method of Section 9.1 we generate the linear invariants for $\mathcal{M}'_1 \cap \mathcal{M}'_2 \cap \cdots \cap \mathcal{M}'_k$. These linear invariants hold when the non-default value condition is satisfied.

9.3.2 Generating Linear Guard Invariants

Consider k register automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ and let R_i (with $i \in [1, k]$) designate the value returned by $\mathcal{M}_i$. We focus on generating necessary conditions, called *guard invariants* or, simply, *guards*, introduced

in Definition 9.3.1, for enabling transitions of the register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$. Further, we give a three-step procedure for generating guards for transitions of $\mathcal{I}$.

Definition 9.3.1 (guard)**.** Consider a transition t of the register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$. A *guard* of t is a linear inequality of the form $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i \geq 0$ such that there does not exist any accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I}$ such that (1) after consuming the signature of X, the vector $(R_1, R_2, \ldots, R_k)$ returned by $\mathcal{I}$ satisfies the inequality $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot R_i < 0$, (2) and the transition t was triggered upon consuming the signature of X.

The following example illustrates Definition 9.3.1.

Example 9.3.2 (guard invariants)**.** Consider the NB_PEAK(X, P) and NB_VALLEY(X, V) time-series constraints with X being $\langle X_1, X_2, \ldots, X_n \rangle$. The intersection $\mathcal{I}$ of the register automata for NB_PEAK and NB_VALLEY was given in Part (C) of Figure 9.1. Observe that, if at the initial state s the register automaton consumes '$<$' (respectively '$>$'), then the number of peaks (respectively valleys) in X is greater than or equal to the number of valleys (respectively peaks). Hence, we can impose the guard $P \geq V$ (respectively $V \geq P$) on the transition from s to t (respectively to r). Part (C) of Figure 9.7 gives the register automaton $\mathcal{I}$ with the obtained guards. △

Guards for the transitions of a register automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ can be generated in three steps:

1. First, we identify the subset $\mathcal{T}$ of transitions of $\mathcal{I}$ such that, for any transition t in $\mathcal{T}$, upon consuming any sequence, t can be triggered at most once.
2. Second, for every transition t in $\mathcal{T}$, we obtain a new register automaton $\mathcal{I}_t$ by removing from $\mathcal{I}$ all transitions of $\mathcal{T}$ different from t that start at the same state as t.
3. Third, using the technique of Section 9.1 on the invariant digraph $G^v_{\mathcal{I}_t}$, we obtain linear invariants that are guards of transition t.

9.4 Infeasible Combinations of the Result Values not Eliminated by the Generated Linear Invariants

In this section, we give an example of a pair of time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$ imposed on the same sequence X such that the generated linear invariants do not remove all infeasible combinations of R_1 and R_2 located outside the convex hull of feasible combinations of R_1 and R_2.

Example 9.4.1 (infeasible combinations not eliminated by the generated linear invariants)**.** Consider the NB_DECREASING_SEQUENCE(X, R_1) and the SUM_WIDTH_ ZIGZAG(X, R_2) time-series constraints imposed on the same time series X of length n. With our method we generate linear and conditional invariants for this pair of constraints, from which the most interesting are ①: $R_1 > 0 \wedge R_2 > 0 \Rightarrow 3 \cdot R_1 \leq R_2 + n$ and ②: $R_2 \leq 2 \cdot R_1$. For the value of n being either 11 or 12, Figure 9.8 gives all feasible combinations (blue squares) of R_1 and R_2, all infeasible combinations (violet diamonds) of R_1 and R_2 inside the convex hull of feasible combinations of R_1 and R_2, and all infeasible combinations (red circles) outside the convex hull. All points eliminated by ① (respectively by ②) are located in the pink (respectively blue) half-space. We can see that there are some points, pictured by red circles, that are outside the convex hull of feasible points and are neither in the pink nor in the blue half-space. Such points are not eliminated by the generated invariants, and some work would still be required in that direction. △

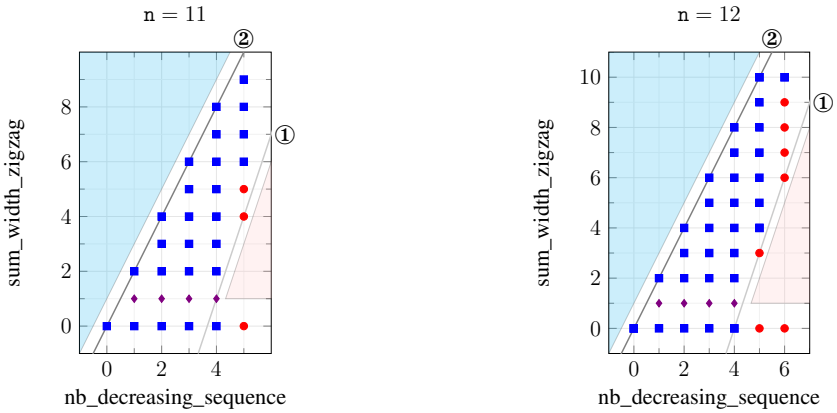


Figure 9.8 – Feasible (blue squares) and infeasible (red circles and violet diamonds) combinations of the results values R_1 and R_2 of the conjunction of constraints NB_DECREASING_SEQUENCE(X, R_1) and SUM_WIDTH_ZIGZAG(X, R_2) imposed on the same sequence X whose length n is either 11 or 12. The grey line labelled with ① (respectively ②) represents the condition $3 \cdot R_1 \geq R_2 + n$ (respectively $R_2 \geq 2 \cdot R_1$). The pink (respectively blue) half-space located to the right (respectively left) of the grey line ① (respectively ②) is the set of points eliminated by the $R_1 > 0 \wedge R_2 > 0 \Rightarrow 3 \cdot R_1 \leq R_2 + n$ (respectively $R_2 \geq 2 \cdot R_1$) invariant. Red circles are infeasible pairs of R_1 and R_2 that are outside the convex hull of feasible pairs and are eliminated by neither invariant.

9.5 Conclusion

In this chapter, we presented a systematic method for generating linear invariants linking the result variables of several AUTOMATON global constraints. Future work may look how to extend the current approach to handle register automata that also allow the $\min$ and $\max$ aggregators for register updates. It should also investigate the use of such invariants within the context of MIP. While MIP has been using linear cuts for a long time [75, 96], no off-the-shelf database of parameterised cuts in some computer readable format is currently available. Linear cuts are typically defined in papers and are then directly embedded within MIP solvers.

Going beyond our empirical evaluation, future work can also look at the quality analysis of generated cuts, i.e. verifying whether generated cuts are sharp or not. It could exploit the following idea: for a conjunction of constraints $\gamma_1(\langle X_1, X_2, \ldots, X_n \rangle, R_1)$ and $\gamma_2(\langle X_1, X_2, \ldots, X_n \rangle, R_2)$, a generated cut $e + e_0 \cdot n + e_1 \cdot R_1 + e_2 \cdot R_2 \geq 0$ is sharp for any sequence length n iff for any n, there exists two different sequences X^1 and X^2 both of length n such that

- $e + e_0 \cdot n + e_1 \cdot R_1^i + e_2 \cdot R_2^i = 0$, where R_1^i and R_2^i are restricted by $\gamma_1(X^i, R_1^i)$ and $\gamma_2(X^i, R_2^i)$, respectively, with i being either 1 or 2. In other words, both points (R_1^1, R_2^1) and (R_1^2, R_2^2) are located on the straight line $e + e_0 \cdot n + e_1 \cdot R_1 + e_2 \cdot R_2$.
- either $R_1^1 \neq R_1^2$ or $R_2^1 \neq R_2^2$. In other words, the points (R_1^1, R_2^1) and (R_1^2, R_2^2) are distinct.

If such sequences X^1 and X^2 exist for any length n, then at least two feasible points are located on the generated cut, and thus this cut is sharp.

Summary of this Chapter:

The main contribution of this chapter is a systematic method for generating linear invariants linking the result variables of several sequence constraints that have a representation by a register automaton satisfying the incremental-automaton property. In the context of time-series constraints, our method applies for all constraints of the NB_σ and the SUM_WIDTH_σ families.

Chapter 10

Synthesising Parameterised Non-Linear Invariants

This chapter is the result of a collaboration with (in alphabetic order) Nicolas Bedliceanu and Helmut Simonis. The author of this thesis was one of the main researchers and writers of this work.

While artificial intelligence has considered from its very beginning the possibility to automate the process of scientific discovery [86], relatively little work has been carried out in this area [91]. 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 [85], most computer science work has focused on the first part, namely generating conjectures both for specific domains like graph theory [77], or for more general domains [64, 87]. The main reason is that the proof part is a key bottleneck, as it is much more challenging to automate as already observed in [52], even if programs that could prove theorems in propositional or first order logic already exist since the fifties [103].

The contribution of this chapter 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 the size of the input alphabet are independent both of 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 chapter 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

$$\forall X,\ \gamma_1(X, R_1) \wedge \gamma_2(X, R_2) \Rightarrow \bigwedge_{i=1}^{m} \neg f_i(R_1, R_2, n) \tag{10.1}$$

In order to prove that (10.1) 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 (respectively R_2) as the result value of γ_1 (respectively 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 and does not explore any search space.

This invariant generation process is offline: it is done once and for all in order to build a database of generic invariants. This chapter is organised in the following way:

- Section 10.1 motivates this work with a running example, which illustrates the need for deriving non-linear invariants.
- Section 10.2 presents our method for deriving 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 10.2. Its goal is to generate a dataset, from which we will extract invariants.
  2. A *mining phase* is detailed in Section 10.2.1. 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 10.2.2. 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* 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 the goal of our work is not to discover statistical properties of time-series constraints, but rather to extract mathematical theorems, which are always true.

The impact of this theoretical contribution is estimated in Chapter 16 of Part III, which first evaluates the capability of our method to capture most infeasible points by using the data mining phase only on small time-series lengths from 7 to 12, and by checking on the unseen data set of time-series lengths from 13 to 24, whether there are uncovered infeasible combinations of R_1 and R_2. Second, it evaluates the effect of adding the obtained non-linear invariants to the state of the art, which contains linear invariants, synthesised by the method of Chapter 9.

10.1 Motivation 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 Chapter 9, using the representation of γ_1 and γ_2 as *register automata*, we presented a method for deriving parameterised linear invariants 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 combinations. The following example illustrates such a situation.

Example 10.1.1 (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 serie X of length n. For the values of n in the interval $[9, 12]$, Figure 10.1 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. △

This work develops a systematic approach for generating invariants characterising infeasible combinations of R_1 and R_2 located within the convex hull of feasible combinations. Section 10.2 describes our method.

10.2 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 of feasible combinations of R_1 and R_2. Our approach uses three sequential phases.

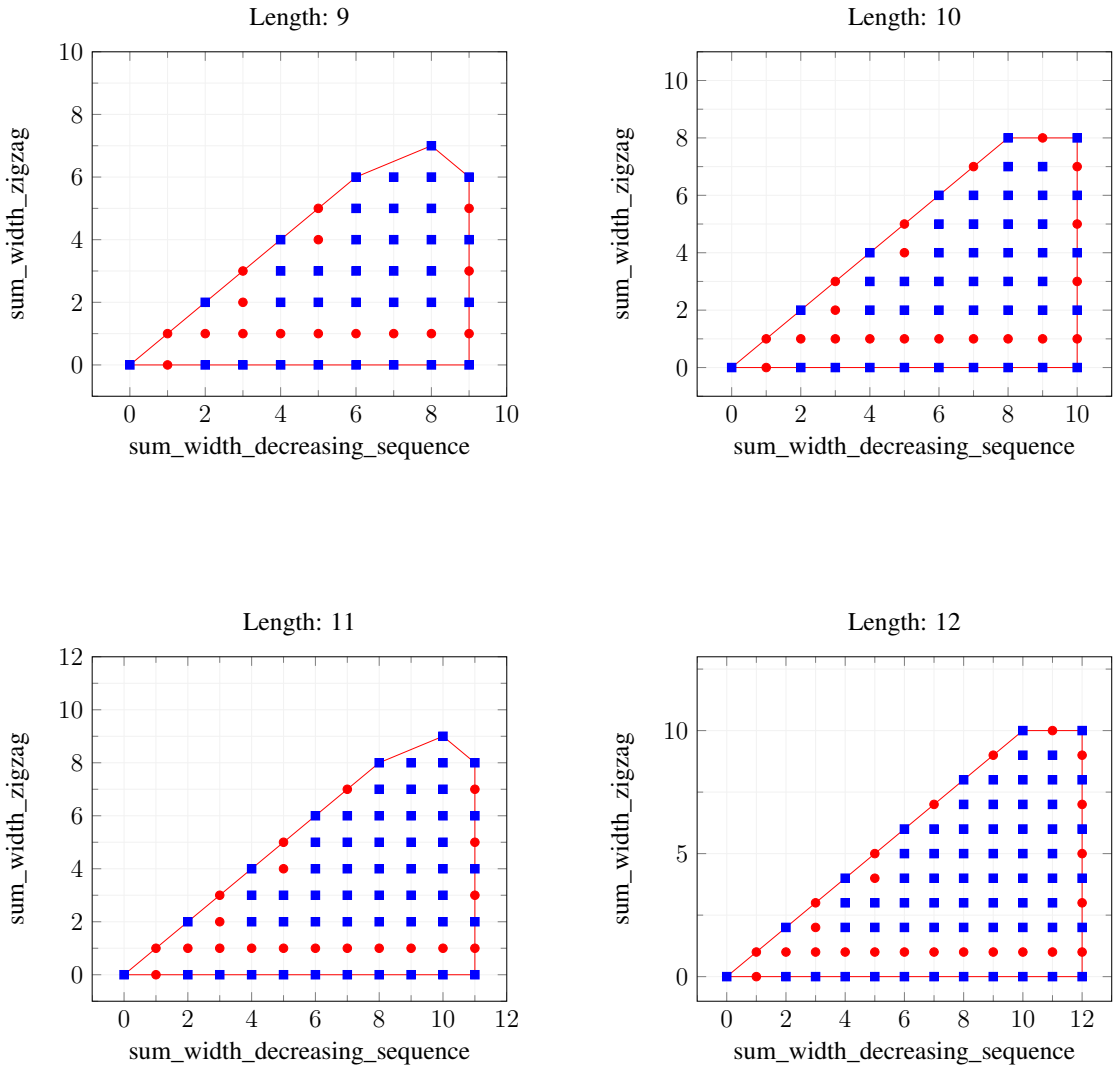


Figure 10.1 – Feasible points, shown as blue squares, for the result variables R_1, R_2 of the conjunction of SUM_WIDTH_DECREASING_SEQUENCE(X, R_1) and SUM_WIDTH_ZIGZAG(X, R_2) on the same time series $X = \langle X_1, X_2, \ldots, X_n \rangle$ for the values of n in $\{9, 10, 11, 12\}$; red circles represent infeasible points inside the convex hull of feasible points, while red straight lines depict the facets of the convex hull of feasible points.

[GENERATING DATA PHASE] The first phase of our method is a preparatory work, namely *generating data*. For each time-series 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 $\mathcal{I}$ of infeasible combinations of R_1 and R_2 from the generated data. We represent this hypothesis as a disjunction of Boolean functions $f_i(R_1, R_2, n)$. The mining phase is described in Section 10.2.1.

[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 $\mathcal{I}$. The proof phase is described in Section 10.2.2.

10.2.1 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 is a number of works on learning a disjunction of predicates [49], and some special case, where disjunction corresponds to a geometric concept [50, 53]. 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 in Section 10.2.1.1 our dataset, which consists of *feasible* and *infeasible* pairs of the result values R_1, R_2.
- Second, we define in Section 10.2.1.2 the space of concepts, *hypotheses*, we can potentially extract from our dataset.
- Third, we outline in Section 10.2.1.3 the *target hypothesis* for time-series constraints, i.e. what we are searching for.
- Finally, we briefly describe in Section 10.2.1.4 the algorithm used for finding the target hypothesis.

10.2.1.1 Input Dataset

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

- For every (k, p_1, p_2) (with $k \in [7, 12]$) 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.
- For every (k, p_1, p_2) (with $k \in [7, 12]$) in $\mathcal{D}^-$,
  1. there does not exist any time series of length k that would yield p_1 and p_2 as the values of R_1 and R_2, respectively.
  2. (p_1, p_2) is located within the convex hull of feasible combinations of R_1 and R_2.

10.2.1.2 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 being a predicate, 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) $R_j \geq d$,
(v) $R_j \leq d$,
(vi) $R_j = c$,
(vii) $R_j = \text{up}(R_j, n) - c$,
(viii) $R_j = c \cdot R_k + d$,

with c and d being natural numbers, and $\text{up}(R_k, n)$ 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) stands 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 [14] 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.

10.2.1.3 Target Hypothesis

Definition 10.2.1 (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 10.1, but the two Boolean functions $R_1 = 13$ and $R_1 = R_2$ are not.

Definition 10.2.2 (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 10.2.3 (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.

10.2.1.4 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) $n > c$, 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 data set generated from time series of length in $[7, 12]$.

10.2.2 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, $R_1 = \text{up}(R_1, n) \wedge R_2 \bmod 2 = 1$ 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. In Section 10.2.2.1 (respectively Section 10.2.2.2), we show how to prove that an independent (respectively dependent) Boolean function is universally true.

10.2.2.1 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 10.2.1, provided that there exists constant-size automata associated with the atomic relations in f.

Definition 10.2.4 (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 10.2.5 (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 10.2.1 (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 10.2.1 follows from Definitions 10.2.4 and 10.2.5.

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.

Chapter 11 will further show how to generate automata for independent atomic relations. Every such automaton is called a *conditional automaton*.

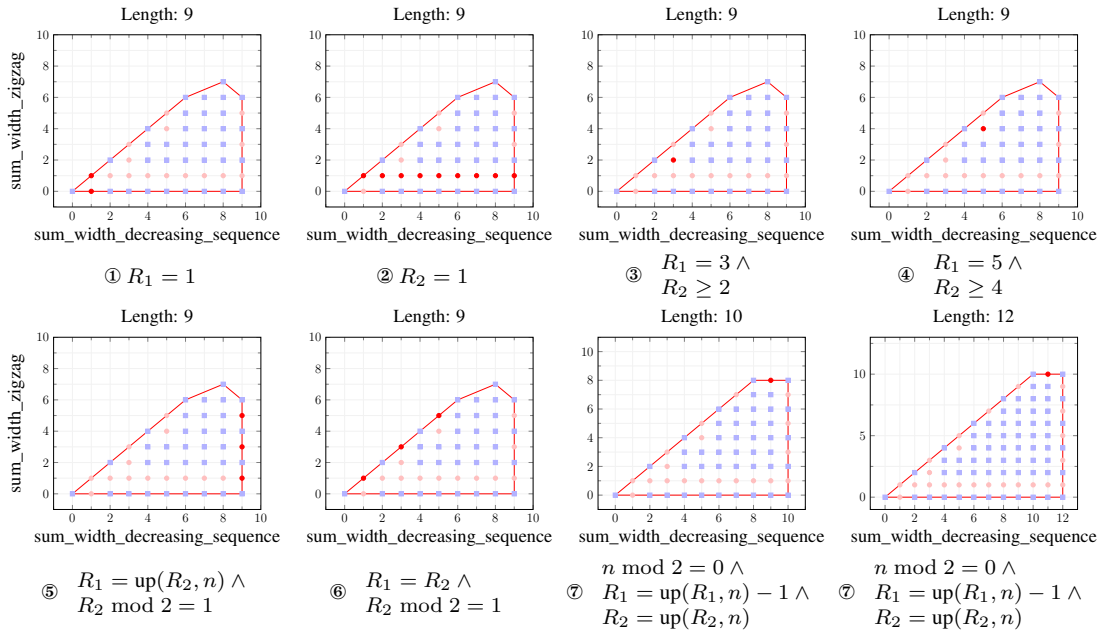


Figure 10.2 – Seven groups of infeasible combinations of R_1 and R_2, where R_1 and R_2 are, respectively, constrained by SUM_WIDTH_DECREASING_SEQUENCE(X, R_1) and SUM_WIDTH_ZIGZAG(X, R_2) on the same sequence X of length 9 (all plots on top and the two plots on bottom left) and of lengths 10 and 12 (the two plots on bottom right).

10.2.2.2 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 Chapter 9.

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 three 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 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 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 Chapter 9 produces a linear 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 [13] did indeed generate the desired linear invariant, which proved that the considered Boolean function is universally true.

Example 10.2.1 (mining, proving and filtering non-linear invariants)**.** Consider the conjunction of the SUM_WIDTH_DECREASING_SEQUENCE(X, R_1) and the SUM_WIDTH_ZIGZAG(X, R_2) time-series con-

straints on the same time series X, introduced in Example 10.1.1. 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 $R_1 = \text{up}(R_1, n) \ \wedge \ R_2 \bmod 2 = 1$ ⑥ $R_1 \bmod 2 = 1 \ \wedge \ R_1 = R_2$
  ⑦ $n \bmod 2 = 0 \ \wedge \ \ R_1 = \text{up}(R_1, n) - 1 \ \wedge \ R_2 = \text{up}(R_2, n)$

All four upper plots and the two lower plots on the left of Figure 10.2 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 10.2 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. For example, the automata for both atomic relations of the ⑦ are given in Figure 10.3. One can take their intersection to check it is 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 presented by an automaton or a register automaton satisfying the required properties of the method of Chapter 9, 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. △

10.3 Infeasible Combinations not Eliminated by our Non-Linear Invariants

In this section, we give an example of a pair of time-series constraints $\gamma_1(X, R_1)$ and $\gamma_2(X, R_2)$ imposed on the same sequence X such that the generated non-linear invariants do not remove all infeasible combinations of R_1 and R_2 located within the convex hull of feasible combinations of R_1 and R_2.

Example 10.3.1 (infeasible combinations not eliminated by the generated non-linear invariants)**.** Consider SUM_WIDTH_PLAIN$(\langle X_1, X_2, \ldots, X_n \rangle, R_1)$ and SUM_WIDTH_ ZIGZAG$(\langle X_1, X_2, \ldots, X_n \rangle, R_2)$ imposed

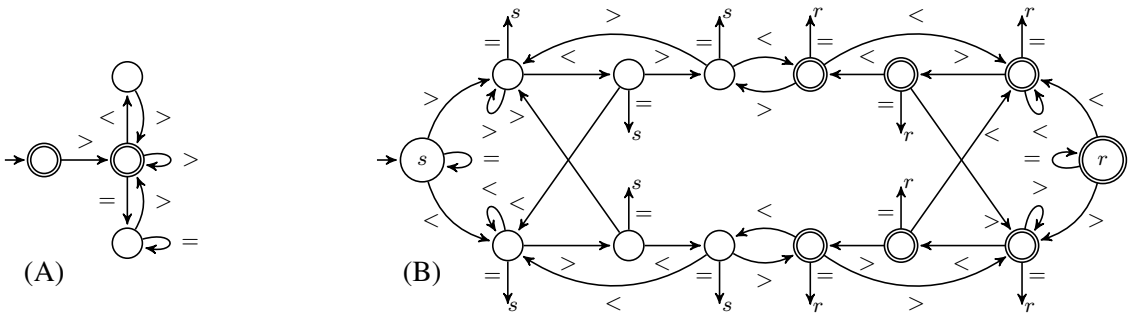


Figure 10.3 – (A) Automaton for the $R_1 = \text{up}(R_1, n)$ atomic relation, where R_1 is constrained by SUM_WIDTH_DECREASING_SEQUENCE(X, R_1). (B) Automaton for the $R_2 \bmod 2 = 1$ atomic relation, where R_2 is constrained by SUM_WIDTH_ZIGZAG(X, R_2).

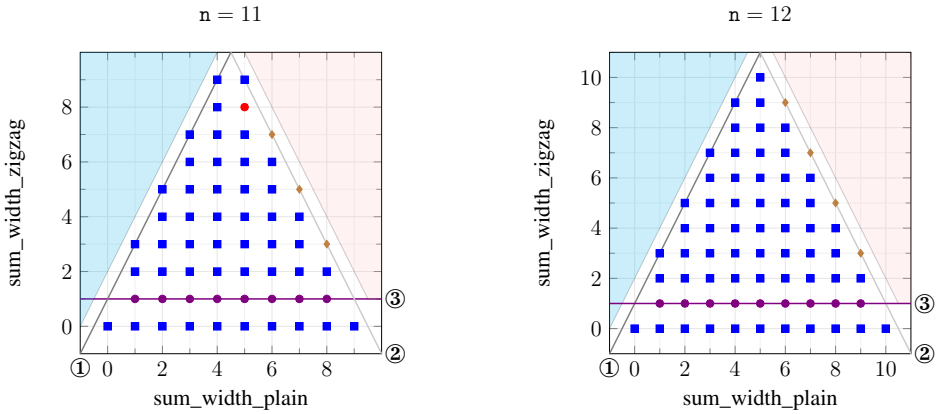


Figure 10.4 – Feasible (blue squares) and infeasible (red and violet circles and brown diamonds) combinations of the results values R_1 and R_2 of the conjunction of constraints SUM_WIDTH_PLAIN(X, R_1) and SUM_WIDTH_ZIGZAG(X, R_2) imposed on the same sequence X whose length n is either 11 or 12. The grey straight line labelled with ① (respectively ②) is represented by the condition $R_2 = 2 \cdot R_1 + 1$ (respectively $2 \cdot R_1 + R_2 = 2 \cdot n - 3$). The violet straight line, labelled with ③, is represented by the equation $R_2 = 1$. The blue (respectively pink) half-space located to the left (respectively right) of the grey line ① (respectively ②) is the set of points eliminated by the $R_2 \leq 2 \cdot R_1 + 1$ (respectively $R_1 > 0 \wedge R_2 > 0 \Rightarrow 2 \cdot R_1 + R_2 \leq 2 \cdot n - 3$) invariant. The red circle (respectively brown diamonds) is an infeasible pair of R_1 and R_2 that is outside (respectively within) the convex hull of feasible pairs and is not eliminated by the generated invariants. The points denoted by violet circles are infeasible and eliminated by the $R_2 \neq 1$ invariant.

on the same time series. Using the methods of Chapter 9 and of this chapter we generate linear and non-linear invariants for this pair of constraints, from which the most interesting are ①: $R_2 \geq 2 \cdot R_1 + 1$, ②: $R_1 > 0 \wedge R_2 > 0 \Rightarrow 2 \cdot R_1 + R_2 \geq 2 \cdot n - 3$, and ③: $R_2 \neq 1$. For the value of n being either 11 or 12, Figure 10.4 gives all feasible combinations (blue squares) of R_1 and R_2, all infeasible combinations (red and violet circles) of R_1 and R_2 inside the convex hull of feasible combinations of R_1 and R_2, and all infeasible combinations (brown diamonds) outside the convex hull. All points eliminated by ① (respectively by ②) are located in the pink (respectively blue) half-space. The points denoted by violet circles are eliminated by ③. The point denoted by a red circle is not eliminated by our non-linear invariants since its x-coordinate is $\lfloor \frac{n}{2} \rfloor$, and it cannot be represented as some constant d_1, or $\text{up}(R_1, n) - d_2$, where d_2 is an integer constant. Hence, in order to express the coordinates of this point in terms of atomic relations we need to extend our set of atomic relations, which may also require a different proof scheme that uses parameterised automata. △

10.4 Conclusion

This chapter proposes a systematic approach to extract and prove non-linear invariants denoting infeasible combinations of the result values of two different time-series constraints imposed on the same time series. To avoid being instance specific these invariants are parameterised by the time-series length. The approach relies on the fact that infeasible pairs are quite often located at a small distance from the convex hull of all feasible pairs, and can therefore be described by intersecting constant-size automata.

Summary of this Chapter:

The main contribution of this chapter is a systematic method for extracting and proving non-linear invariants for conjunctions of time-series constraints of the NB_σ and SUM_WIDTH_σ families. Such invariants characterise infeasible combinations of the result variables of time-series constraints in the conjunction that are located inside the convex hull of feasible combinations. The main idea of the proof part is to represent an infinite set of sequences as the intersection of constant-size automata.

Chapter 11

Synthesising Constant-Size Conditional Automata

For two families of time-series constraints, considered in Chapter 10, we need to generate constant-size finite automata satisfying certain restrictions, e.g. an automaton recognising the signatures of all and only all time series with an odd number of peaks. Such automata are required for proving generic non-linear invariants parameterised by the time-series length, described in Chapter 10. This chapter 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 10.2.1.2. In particular, one of the most interesting atomic relations we consider is $R = \text{up}(R, n) - d$, where R is constrained by some time-series constraint $\gamma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ with γ being either 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. All the conditional automata are obtained by either using the register automaton for γ or the seed transducer for the regular expression associated with γ.

This chapter is organised as follows:

- First, in Section 11.1, we explain how to generate a constant-size automaton for the $R = d$ atomic relation, called a *constant* atomic relation.
- Second, in Section 11.2, we explain how to generate a constant-size automaton for the $R \bmod d = b$ atomic relation with d being a positive integer number and b being in the interval $[0, d-1]$, called a *modulo* atomic relation.
- Third, in Section 11.3, we explain how to generate a constant-size automaton for the $R = \text{up}(R, n) - d$ atomic relation with d being a natural number, called a *gap* atomic relation.
- Finally, in Section 11.4, we explain how to generate a constant-size automata for the atomic relations of the type $R \geq d$ (respectively $R \leq \text{up}(R, n) - d$), with d being a natural constant, called a *not-less* (respectively a *not-greater*) atomic relation.

11.1 Generation of Constant-Size Automata for Constant Atomic Relations

Consider a time-series constraint $\gamma(X, R)$ with its register automaton $\mathcal{M}$ and a constant atomic relation C of the form $R = d$. In this section, we focus on the generation of a constant-size automaton for C using $\mathcal{M}$.

We introduce in Definition 11.1.1 the *non-negativity conditions* on $\mathcal{M}$, which are a set of 3 conditions restricting the initial values of the register, the register updates, and the acceptance function of $\mathcal{M}$. Further,

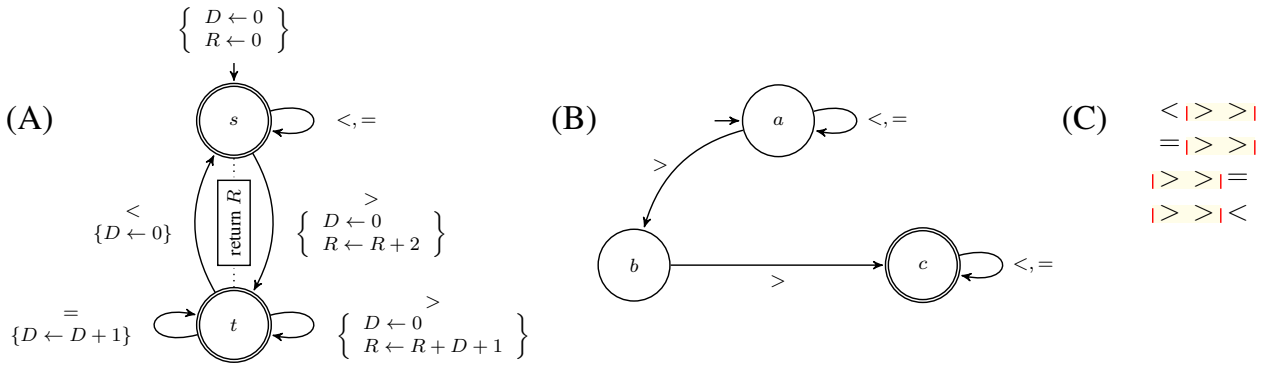


Figure 11.1 – (A) Register automaton for SUM_WIDTH_DECREASING_SEQUENCE(X, R). (B) Automaton for the $R = 3$ constant atomic relation. (C) All signatures of length 3 accepted by the automaton in Part (B); all occurrences of DECREASING_SEQUENCE are highlighted in yellow, and the red bars designate the borders of maximal occurrences.

Theorem 11.1.1 constructively proves that when the non-negativity conditions are satisfied for $\mathcal{M}$, the automaton for C exists and can be obtained from $\mathcal{M}$. We consider first an illustrating example.

Example 11.1.1 (automaton for a constant atomic relation)**.** Consider the SUM_WIDTH_DECREASING_SEQUENCE(X, R) constraint, whose register automaton is given in Part (A) of Figure 11.1, and the atomic relation C defined by $R = 3$. The minimal automaton representing the atomic relation C is given in Part (B) of Figure 11.1. Part (C) of Figure 11.1 gives all signatures of length 3 recognised by the automaton in Part (B). △

Definition 11.1.1 (non-negativity conditions)**.** Consider a register automaton $\mathcal{M}$ over an input alphabet Σ recognising any input signature over Σ. The *non-negativity* conditions on $\mathcal{M}$ are defined as follows:

1. Every register update of $\mathcal{M}$ has one of the following forms:
   (a) The register is incremented by a natural number, or by the value of another register.
   (b) The value of the register is reset to a natural number.
2. The initial values of the registers of $\mathcal{M}$ are natural numbers.
3. The acceptance function of $\mathcal{M}$ is a weighted sum with natural number coefficients of the last values of the registers of $\mathcal{M}$ after having consumed an input signature.

If a regular expression σ and an integer constant b_σ form a recognisable pattern, i.e. a seed transducer for σ exists [68], then the register automata [11] for the SUM_WIDTH_σ and NB_σ time-series constraints satisfy the non-negativity conditions.

Theorem 11.1.1 (existence of the automaton for a constant atomic relation)**.** Consider a $\gamma(X, R)$ time-series constraint whose register automaton $\mathcal{M}$ satisfies the non-negativity conditions, and a natural number d. Then there exists an automaton representing the $R = d$ constant atomic relation, denoted by C.

Proof. We prove the theorem by explicitly constructing a constant-size automaton $\mathcal{M}_C$ representing C using $\mathcal{M}$.

[Construction of $\mathcal{M}_C$] Let $\langle A_1, A_2, \ldots, A_p \rangle$ be the registers of $\mathcal{M}$, 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 $\mathcal{M}$. Then, the states, the initial state, the accepting states, and the transitions of $\mathcal{M}_C$ are defined as follows:

- **States.** For every state q of $\mathcal{M}$, there are $(d+2)^p$ states in $\mathcal{M}_C$, each of which is labelled with $q_{i_1,i_2,\ldots,i_p}$ with every i_j (with $1 \leq j \leq p$) being in $[0, d+1]$.

- **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 $\mathcal{M}_C$.
- **Accepting states.** A state $q_{i_1,i_2,\ldots,i_p}$ of $\mathcal{M}_C$ is accepting iff $\alpha(i_1, i_2, \ldots, i_p)$ is equal to d.
- **Transitions.** There is a transition from state $q_{i_1,i_2,\ldots,i_p}$ (with $i_1, i_2, \ldots, i_p \in [0, d+1]$) 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 $\min(d+1, i^*_j)$, with j in $[1, p]$.

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

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

- the number of states of $\mathcal{M}$,
- the number of registers of $\mathcal{M}$,
- the parameter of the considered constant atomic relation.

We explain why $\mathcal{M}_C$ needs only $m \cdot (d+2)^p$ states to represent the set of supporting signatures of the $R = d$ constant atomic relation. We show that if, when consuming the signature of some ground time series, the value of some register of $\mathcal{M}$ becomes greater than d, 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 registers of $\mathcal{M}$. If for a register 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 d, the value of α also exceeds d. By the non-negativity conditions, if the value of A_j exceeds d it can either increase even more, or it can be reset to a natural number. In either case, the exact value of A_j is irrelevant, and it is enough to know a lower bound, $d+1$, on its value.

[Correctness of $\mathcal{M}_C$] We now prove that the constructed automaton $\mathcal{M}_C$ is sound, i.e. it recognises the signatures of *only* ground time series yielding d as the value of R, and complete, i.e. it recognises the signatures of *all* ground time series yielding d as the value of R.

- **Soundness of $\mathcal{M}_C$.** We prove the soundness of $\mathcal{M}_C$ by contradiction. Assume there exists a ground time series X recognised by $\mathcal{M}_C$ and that yields $d' \neq d$ as the value of R. 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 d this means that, after consuming S, the register automaton $\mathcal{M}$ finishes in the state q of $\mathcal{M}$, and for every $j \in [1, p]$, if $i_j \leq d$ (respectively $i_j = d+1$), then the register A_j has value i_j (respectively is strictly greater than d). Since $q_{i_1,i_2,\ldots,i_p}$ is an accepting state of $\mathcal{M}_C$, then $\alpha(i_1, i_2, \ldots, i_p)$ is equal to d, and we obtain the contradiction.
- **Completeness of $\mathcal{M}_C$.** We prove the completeness of $\mathcal{M}_C$ by assuming that there exists a ground time series X that yields d as the value of R, but its signature S is not recognised by $\mathcal{M}_C$. Then,
  1. either the final state $q_{i_1,i_2,\ldots,i_p}$ of $\mathcal{M}_C$ after consuming S is not accepting,
  2. or the automaton $\mathcal{M}_C$ 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 $\mathcal{M}_C$, after having consumed the signature of X, the automaton $\mathcal{M}$ finishes 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 X yields d as the value of R the state $q_{i_1,i_2,\ldots,i_p}$ of $\mathcal{M}_C$ 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 $\mathcal{M}_C$, there does not exist a transition labelled with some input symbol s, and that (2) $\mathcal{M}_C$ needs to trigger this transition when consuming the signature of X. Then, at state q of $\mathcal{M}$, there does not exist the transition

labelled with s. This contradicts the nature of the register automaton $\mathcal{M}$ since it must compute the value of R for any time series. Hence, Situation 2 is also impossible.

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

Since $\mathcal{M}_C$ is sound and complete, then $\mathcal{M}_C$ is indeed an automaton representing the constant atomic relation C. □

11.2 Generation of Constant-Size Automata for Modulo Atomic Relations

Consider a time-series constraint $\gamma(X, R)$ with its register automaton $\mathcal{M}$ and a modulo atomic relation C of the form $R \bmod d = b$ with d being a positive natural number, and b being in the interval $[0, d-1]$. In this section, we focus on the generation of constant-size automaton for C using $\mathcal{M}$.

We introduce in Definition 11.2.1 the *modulo conditions* on $\mathcal{M}$, which are a relaxed version of the non-negativity conditions, introduced in Definition 11.1.1. The main difference is that in the modulo conditions there are no restrictions on the sign of the integer constants in the register updates. Further, Theorem 11.2.1 constructively proves that when the modulo conditions are satisfied for $\mathcal{M}$, the automaton for C exists and can be obtained from $\mathcal{M}$. We start with an illustrating example.

Example 11.2.1 (automaton for a modulo atomic relation)**.** Consider the NB_DECREASING_SEQUENCE(X, R) time-series constraint, and the $R \bmod 2 = 1$ atomic relation, denoted by C. The automaton for C is given in Part (B) of Figure 11.2. It is obtained from the register automaton for NB_DECREASING_SEQUENCE, given in Part (A). Part (C) of Figure 11.2 gives all signatures of length 2 recognised by the automaton in Part (B). △

Definition 11.2.1 (modulo conditions)**.** Consider a register automaton $\mathcal{M}$ over an input alphabet Σ recognising any input signature over Σ. The *modulo* conditions on $\mathcal{M}$ are defined as following.

1. Every register update of $\mathcal{M}$ has one of the following forms:
   (a) The register is changed by an integer constant, or by an integer constant plus the value of another register.
   (b) The value of the register is reset to an integer constant.
2. The initial values of the registers of $\mathcal{M}$ are integer constants.

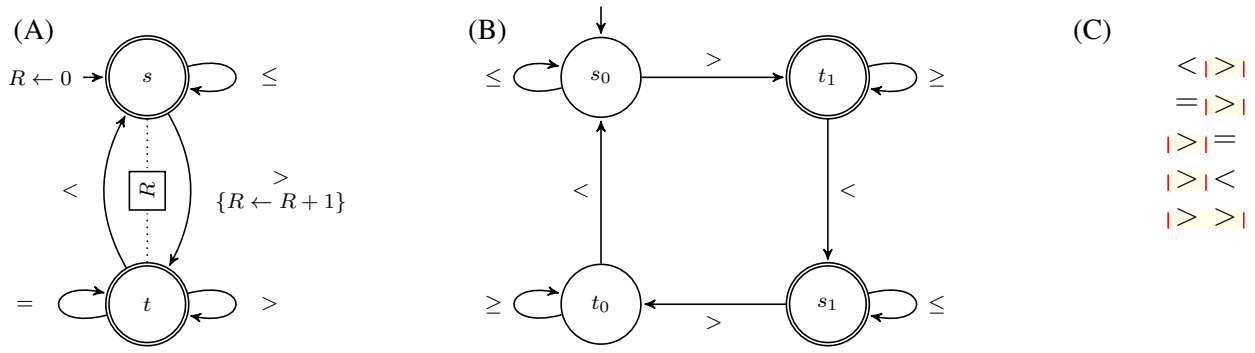


Figure 11.2 – (A) Register automaton for NB_DECREASING_SEQUENCE(X, R). (B) Automaton for the $R \bmod 2 = 1$ atomic relation, i.e. accepting the signatures of all and only time series yielding an odd value of R. (C) All signatures of length 2 accepted by the automaton in Part (B); all occurrences of DECREASING_SEQUENCE are highlighted in yellow, and the red bars designate the borders of maximal occurrences.

3. The accepting function of $\mathcal{M}$ is a weighted sum with integer coefficients of the last values of the registers of $\mathcal{M}$ after having consumed an input signature.

If a regular expression σ and an integer constant b_σ form a recognisable pattern, i.e. a seed transducer for σ exists [68], then the register automata [11] for the SUM_WIDTH_σ and NB_σ time-series constraints satisfy the modulo conditions.

Theorem 11.2.1 (existence of the automaton for a modulo atomic relation)**.** Consider a $\gamma(X, R)$ time-series constraint whose register automaton $\mathcal{M}$ satisfies the modulo conditions of Definition 11.2.1, a positive natural number d, and a number b in the interval $[0, d-1]$. Then there exists an automaton representing the $R \bmod d = b$ modulo atomic relation, denoted by C.

Proof. We prove the theorem by explicitly constructing a constant-size automaton $\mathcal{M}_C$ representing the modulo atomic relation C using the register automaton $\mathcal{M}$.

Let $\langle A_1, A_2, \ldots, A_p\rangle$ be the registers of $\mathcal{M}$, whose initial values are $\langle v_1, v_2, \ldots, v_p\rangle$, and let $\alpha(A_1, A_2, \ldots, A_p)$ denote the acceptance function of $\mathcal{M}$. Then, the states, the initial state, the accepting states, and the transitions of $\mathcal{M}_C$ are defined as follows:

- **States.** For every state q of $\mathcal{M}$, there are d^p states in $\mathcal{M}_C$, each of which is labelled with $q_{i_1,i_2,\ldots,i_p}$ with every i_j (with $1 \leq j \leq p$) being in $[0, d-1]$.
- **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 $\mathcal{M}_C$ with every v'_i being $v_i \bmod d$.
- **Accepting states.** A state $q_{i_1,i_2,\ldots,i_p}$ of $\mathcal{M}_C$ is accepting iff $\alpha(i_1, i_2, \ldots, i_p) \bmod d$ is equal to b.
- **Transitions.** There is a transition from state $q_{i_1,i_2,\ldots,i_p}$ (with $i_1, i_2, \ldots, i_p \in [0, d-1]$) 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 $i^*_j \bmod d$, with j in $[1, p]$.

A similar scheme to the proof of Theorem 11.1.1 can be used to prove that the constructed automaton indeed represents the set of supporting signatures of C. □

11.3 Generation of Constant-Size Automata for Gap Atomic Relations

Consider a time-series constraint $\gamma(\langle X_1, X_2, \ldots, X_n\rangle, R)$ and a gap atomic relation C of the form $R = \text{up}(R, n) - d$ with d being a natural number. In this section, we focus on the generation of constant-size automaton for C. We consider first an illustrative example.

Example 11.3.1 (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 $R = \text{up}(R, n)$. In Chapter 7, we showed 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 11.3 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. △

This section is organised as follows:

- Section 11.3.1 first introduces the notion of *gap of a time series* X, which indicates how far apart the 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 automaton. Third, it describes how to synthesise such δ-gap automaton.

1. Section 11.3.1.1 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 a shortest time series yielding the same result value of a time-series constraint. For example, all words of length 4 (respectively 5) in Part (B) of Figure 11.3 are the signatures of time series whose gap is 0 and whose loss is 0 (respectively 1). Part (C) of Figure 11.3 gives two signatures of time series with gap (respectively loss) 1 and 2 (respectively 3 and 5).

   It shows how to compute the loss with a register automaton, called *loss automaton*, and exploits the connection between the gap and the loss of the same time series as the basis for deriving the sought δ-gap automaton from the loss automaton.
2. Section 11.3.1.2 introduces a sufficient condition in the form of a conjunction of sufficient conditions on a time-series constraint, called *principal conditions*, that, when satisfied, guarantee the existence of the δ-gap automaton.
   - When the three first 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 registers 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 11.3.1.3 constructively proves the existence of the δ-gap automaton.

- Section 11.3.2 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 11.3.1.2. It also shows how to obtain a loss automaton for a NB_σ time-series constraint from the seed transducer for σ. The main idea is to compute the *regret* of every transition of the seed transducer as a special case of *minimax regret* [69, 120] from the 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 [44].
- Section 11.3.3 introduces a sufficient condition on a regular expression σ such that, when σ satisfies this condition, the SUM_WIDTH_σ family satisfies the first three principal conditions of Section 11.3.1.2.

11.3.1 Deriving a δ-gap Automaton for a Time-Series Constraint

We present the main contribution of this chapter namely a systematic method for deriving a δ-gap automaton for a time-series constraint, see Definition 11.3.2, satisfying certain conditions that will be given in Definition 11.3.6. We first introduce the *gap of a ground time series* in Definition 11.3.1, and the *δ-gap automaton for a time-series constraint* in Definition 11.3.2. Let $\mathbb{S}$ denote the set of time-series constraints of the NB_σ and SUM_WIDTH_σ families.

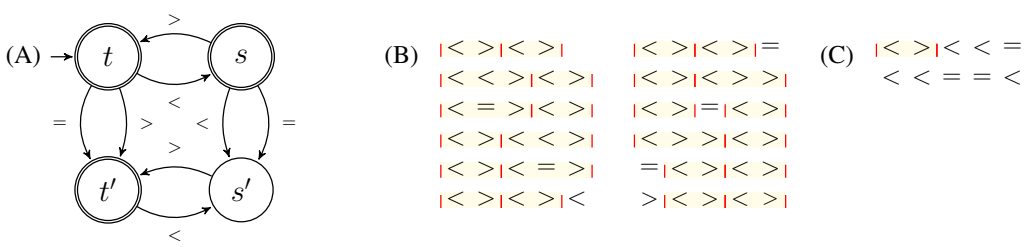


Figure 11.3 – (A) Automaton achieving the maximum number of peaks in a time series of length n, i.e. $\max(0, \lfloor \frac{n-1}{2} \rfloor)$, and (B) all corresponding accepted words for $n-1 \in \{4, 5\}$, where each peak is surrounded by two vertical bars, and is highlighted in yellow. (C) The signatures of time series with gap 1 and 2, respectively, and with loss 3 and 5, respectively.

Definition 11.3.1 (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 $\mathrm{gap}_\gamma(X)$, 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 11.3.3 will illustrate the notion of gap for different time series.

Definition 11.3.2 (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 11.3.6 will further give a sufficient condition on a time-series constraint γ for the existence of a δ-gap automaton for γ.

Example 11.3.2 (0-gap automaton)**.** The 0-gap automaton for NB_PEAK was given in Part (A) of Figure 11.3. 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.

- First, Section 11.3.1.1 defines the *loss* of a ground time series wrt γ, as well as a *loss automaton* for γ as a register automaton computing the loss.
- Second, Section 11.3.1.2 gives a sufficient condition in the form of conjunction of four conditions on the gap, the loss and the loss automaton such that, when this conjunction is satisfied, the δ-gap automaton for γ exists.
- Third, Section 11.3.1.3 constructively proves that the δ-gap automaton for γ exists when the conjunction of conditions of Section 11.3.1.2 is satisfied.

11.3.1.1 Loss and Loss Automaton

Consider a time-series constraint γ and a natural number δ. Definition 11.3.3 introduces the *loss of a time series* wrt γ, and Definition 11.3.4 presents the notion of *loss automaton* for γ.

Definition 11.3.3 (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 $\mathrm{loss}_\gamma(X)$, 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 11.3.3 (gap and loss of a time series)**.** We illustrate now the computation of the gap and the loss on two examples.

- Consider the NB_PEAK time-series constraint. From [14], the maximum number of peaks in a time series of length n is $\max\left(0, \left\lfloor \frac{n-1}{2} \right\rfloor\right)$.
  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.

- Consider the SUM_WIDTH_PEAK(X, R) time-series constraint. From [14], the maximum value of R for a time series of length n is $\max(0, n-2)$.
  The time series $X^4 = \langle 1, 2, 2, 3, 2, 1, 0\rangle$ has a of gap 0 since it yields 5 as the value of R, and a loss of 0 since any shorter time series yields a smaller value of R. The time series $X^5 = \langle 1, 2, 3, 2, 1, 2, 1\rangle$ has a gap of 1 since it yields $3+1 = 4$ as the value of R, when 5 is the maximum, and a loss of 1 since a shortest time series yielding 4 as the value of R is of length 6. The time series $X^6 = \langle 1, 1, 1, 0, 3\rangle$ has a gap of 3 since it yields 0 as the value of R, when 3 is the maximum, and a loss of 4 since a shortest time series yielding 0 as the value of R is of length 1. △

Definition 11.3.4 (loss automaton for a time-series constraint)**.** Consider a time-series constraint γ. A *loss automaton* for γ is a register automaton over the alphabet $\{<, =, >\}$ with a constant number of registers such that, for any ground time series X, it returns $\mathrm{loss}_\gamma(X)$ after having consumed the signature of X.

For the NB_σ and SUM_WIDTH_σ families of time-series constraints, a loss automaton can be synthesised from the seed transducer for the regular expression σ. For the NB_σ family, this will be explained in Section 11.3.2.3.

11.3.1.2 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 11.3.6 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 registers 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 11.3.5 introduces the notions of before-found and after-found state of a loss automaton.

Definition 11.3.5 (before-found and after-found states)**.** Consider a loss automaton $\mathcal{M}$ for a g_f_σ time-series constraint. An accepting state q of $\mathcal{M}$ is a *before-found* (respectively *after-found*) state, if there exists a time series X without any σ-patterns (respectively 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 11.3.6 (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 $h_\gamma \colon \mathbb{S} \times \mathbb{N} \times \{0, 1\} \times \mathbb{N} \to \mathbb{N}$, 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 $\mathrm{loss}_\gamma(X)$ being equal to $h_\gamma(\mathrm{gap}_\gamma(X), \mathrm{sgn}(R), n)$, where sgn is the signum function. Hence, in order to compute the loss of a ground time series it is enough to know its gap, whether it has at least one σ-pattern or not, and the length of this time series.
2. **Boundedness condition.** For given values of $\mathrm{gap}_\gamma(X)$ and $\mathrm{sgn}(R)$, and for any n in $\mathbb{N}$, the value of the gap-to-loss function $h_\gamma(\mathrm{gap}_\gamma(X), \mathrm{sgn}(R), n)$ belongs to a bounded integer interval, called the *loss interval wrt* $\langle \mathrm{gap}_\gamma(X), \mathrm{sgn}(R)\rangle$.
3. **Disjointness condition.** For a given value of $\mathrm{sgn}(R)$, and two different values of gap, δ_1 and δ_2, the loss intervals wrt $\langle \delta_1, \mathrm{sgn}(R)\rangle$ and wrt $\langle \delta_2, \mathrm{sgn}(R)\rangle$ are disjoint.
4. **Loss-automaton condition.** There exists a loss automaton $\mathcal{M}$ for γ satisfying all the following conditions:

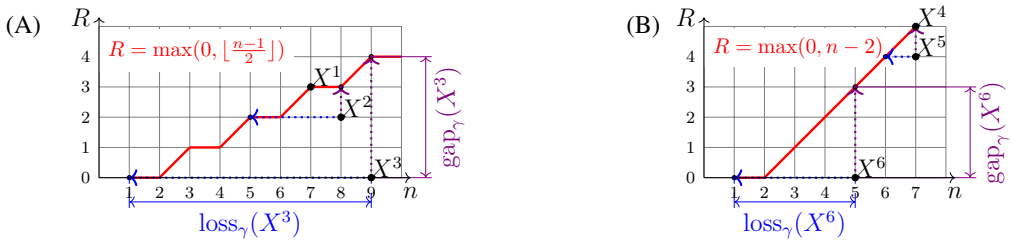


Figure 11.4 – In both figures the horizontal (respectively vertical) axis represents the time-series length n (respectively the result value R of (A) NB_PEAK and (B) SUM_WIDTH_PEAK). The red curves show the maximum value of R for a given n; any point X^i with coordinates (n_i, R_i) denotes all time series of length n_i yielding R_i as the value of R. The length of the blue (respectively violet) dotted line-segments starting from X^i equals the loss (respectively gap) of X^i.

(a) Every register update of $\mathcal{M}$ has one of the following forms:

   i. The register is incremented by a natural number, or by the value of another register.

   ii. The value of the register is reset to a natural number.

(b) The initial values of the registers 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 registers of $\mathcal{M}$ after having consumed an input signature.

Conditions (4a), (4b), and (4c) are the *non-negativity conditions*, introduced in Definition 11.1.1.

(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. This condition is called the *separation condition* on $\mathcal{M}$.

Conditions (1), (2) and (3) are called the *gap-loss-relation conditions*.

Example 11.3.4 (principal conditions)**.** Consider a $\gamma(X, R)$ time-series constraint. For the time series X^1, X^2, X^3, X^4, X^5 and X^6 of Example 11.3.3, Part (A) (respectively Part (B)) of Figure 11.4 shows the relation between the gap, the loss, the time-series lengths, and R when γ is NB_PEAK (respectively SUM_WIDTH_PEAK). For any time series X^i (with i in $[1, 6]$) of length n_i yielding R_i as the value of R, its gap (respectively loss) is equal to the length of the violet (respectively blue) dotted line-segment starting from the point X^i with coordinates (n_i, R_i). Note that the boundedness condition is satisfied for both NB_PEAK and SUM_WIDTH_PEAK. It is easy to see that the disjointness condition is also satisfied for both NB_PEAK and SUM_WIDTH_PEAK. △

11.3.1.3 Deriving the δ-Gap Automaton

Consider a γ time-series constraint satisfying all four principal conditions of Section 11.3.1.2, and a natural number δ. We prove that the δ-gap automaton for γ exists. First, Lemma 11.3.1 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 to describe in terms of loss the set of ground time series whose gap is δ. Then using the result of Lemma 11.3.1, Theorem 11.3.1 constructively proves that the δ-gap automaton for γ exists.

Lemma 11.3.1 (gap-loss relation)**.** Consider a $\gamma(X, R)$ time-series constraint such that the gap-loss-relation conditions, see Definition 11.3.6, 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 disjointness condition, see Condition 3. □

Theorem 11.3.1 (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 11.3.6, 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 11.3.6, 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 11.3.1, 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 $\mathrm{loss}_\gamma(X)$ is in $\mathcal{L}_{\delta,0}$.
- **Type 2.** The time series X has at least one σ-pattern and the value of $\mathrm{loss}_\gamma(X)$ 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 $\langle A_1, A_2, \ldots, A_p\rangle$ be the registers of $\mathcal{M}$, 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 $\mathcal{M}$, 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 $1 \leq j \leq p$) 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 $i_j \leq \phi$ (respectively $i_j = \phi + 1$), the register A_j has value i_j (respectively is strictly greater than ϕ). Hence, the states of $\mathcal{A}_{\mathcal{M}}$ encode the register 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 of the time-series length, respectively defined as:

- the number of states of $\mathcal{M}$,
- the number of registers 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 11.3.6) 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 register 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 registers of $\mathcal{M}$. If for a register 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, $\phi + 1$, on 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 register automaton $\mathcal{M}$ finishes in the state q of $\mathcal{M}$, and for every $j \in [1, p]$, if $i_j \leq \phi$ (respectively $i_j = \phi+1$), then the register A_j has value i_j (respectively 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 (respectively latter) case, X belongs to Type 1 (respectively Type 2), and by Lemma 11.3.1, 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 $\mathcal{M}$ finishes 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 11.3.1 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 the transition labelled with s. This contradicts the nature of the loss automaton $\mathcal{M}$ since it must compute the loss for 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. □

11.3.2 Deriving the δ-gap 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 11.3.6 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 11.3.1 can be used to derive the δ-gap automaton.

1. Section 11.3.2.1 introduces the HOMOGENEITY property. Sections 11.3.2.2 and 11.3.2.3 both assume the HOMOGENEITY property.

2. Section 11.3.2.2 proves three theorems stating that, the gap-to-loss, the boundedness, and the disjointness conditions are satisfied for NB_σ.
3. Section 11.3.2.3 gives a systematic method for constructing a loss automaton $\mathcal{M}$ satisfying the non-negativity and the separation conditions.

11.3.2.1 The HOMOGENEITY Property

Before giving the HOMOGENEITY property in Property 11.3.1, we introduce the notions of `found`-transition and `found`-path in Definition 11.3.5, which will use in one of the conditions of Property 11.3.1, and further in Section 11.3.2.3.

Definition 11.3.7 (`found`-transition, `found`-path)**.** Consider a seed transducer $\mathcal{T}$. A `found`-transition of $\mathcal{T}$ is any transition whose output symbol is either `found` or $\texttt{found}_\texttt{e}$. A `found`-path in $\mathcal{T}$ is any sequence of consecutive transition containing at least one `found`-transition.

Example 11.3.5 (`found`-transition, `found`-path)**.** Consider the seed transducer $\mathcal{T}$ in Part (A) of Figure 11.5. The transition from r to t is a single `found`-transition of $\mathcal{T}$. The sequence of transitions from s to r, from r to t and from t to r is a `found`-path in $\mathcal{T}$. The sequence of transitions from r to t, and from t to t is also a `found`-path in $\mathcal{T}$. △

Property 11.3.1 (HOMOGENEITY property)**.** A regular expression σ has the HOMOGENEITY property if the following conditions are all satisfied:

1. The pair $\langle \sigma, b_\sigma \rangle$ is a *recognisable pattern* [68], see Definition 5.2.4. This implies that the seed transducer for σ exists and can be constructed by the method of [68].
2. The regular expression σ has either the $\overline{\text{NB}}$-overlapping property or the $\overline{\text{NB}}$-non-overlapping property, see Properties 7.2.2 and 7.2.3, when there are no restrictions on the domains of time-series variables. By Theorem 7.2.2, this implies that for the NB_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint, the maximum value of R is $\max\left(0, \left\lfloor \frac{n-c_\sigma}{d_\sigma} \right\rfloor\right)$, where c_σ and d_σ are constants depending on σ.
3. For every state q that is the destination of a `found`-transition, the length of the shortest `found`-path starting in q is d_σ.

11.3.2.2 Verifying the Gap-Loss-Relation Conditions

This section shows that the gap-loss-relation conditions, see Definition 11.3.6, for a NB_σ time-series constraint are satisfied, assuming σ has the HOMOGENEITY property. Theorem 11.3.2 proves the gap-to-loss condition and derives the formula for the gap-to-loss function; Theorem 11.3.3 proves the boundedness condition and derives the formula of loss interval for a given gap and sign of the result value, and, finally, Theorem 11.3.4 proves the disjointness condition.

Theorem 11.3.2 (gap-to-loss condition)**.** Consider a NB_$\sigma(X, R)$ time-series constraint, denoted by $\gamma(X, R)$, such that σ has 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:

$$\text{loss}_\gamma(X) = \text{gap}_\gamma(X) \cdot d_\sigma + (1 - \text{sgn}(R)) \cdot (\min(n, c_\sigma) - 1) + \max(0, n - c_\sigma) \bmod d_\sigma, \qquad (11.1)$$

where sgn is the signum function, and c_σ and d_σ are the constants from the maximum value of R given in Property 11.3.1.

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:

$$\text{gap}_\gamma(X) = \max\left(0, \left\lfloor \frac{n - c_\sigma}{d_\sigma} \right\rfloor\right) - R = \max\left(0, \left\lfloor \frac{n - c_\sigma}{d_\sigma} \right\rfloor\right). \tag{11.2}$$

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

- $\text{gap}_\gamma(X)$ is zero. By (11.2), $n < c_\sigma + d_\sigma$, and the value of the right-hand side of (11.1) 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 (11.2), $n \geq c_\sigma + d_\sigma$, and we have the following equality:

$$\text{gap}_\gamma(X) = \left\lfloor \frac{n - c_\sigma}{d_\sigma} \right\rfloor = \frac{n - c_\sigma - (n - c_\sigma) \bmod d_\sigma}{d_\sigma} \tag{11.3}$$

From (11.3) 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:

$$\text{gap}_\gamma(X) = \left\lfloor \frac{n - c_\sigma}{d_\sigma} \right\rfloor - R = \frac{n - c_\sigma - (n - c_\sigma) \bmod d_\sigma}{d_\sigma} - \frac{(n - \text{loss}_\gamma(X)) - c_\sigma}{d_\sigma} \tag{11.4}$$

Since in the right-hand side of (11.4), both divisions are integer divisions we obtain:

$$\text{gap}_\gamma(X) = \frac{\text{loss}_\gamma(X) - (n - c_\sigma) \bmod d_\sigma}{d_\sigma}. \tag{11.5}$$

By isolating $\text{loss}_\gamma(X)$ from (11.5) we obtain the formula of the theorem. □

Example 11.3.6 (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 11.3.2 for computing the gap-to-loss function for NB_σ. By Theorem 7.2.2, 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 11.3.3 (boundedness condition)**.** Consider a NB_$\sigma(X, R)$ time-series constraint, denoted by $\gamma(X, R)$, such that σ has 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:

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

Proof. Let X be a ground time series of length n whose gap is δ. From Theorem 11.3.2, 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} (respectively ℓ_{max}) is a lower (respectively upper) bound on $\text{loss}_\gamma(X)$ we need to find the minimum (respectively maximum) of the function $z(n) = \min(n, c_\sigma) + \max(0, n - c_\sigma) \bmod d_\sigma$.

- $\boldsymbol{\ell_{min} \leq \text{loss}_\gamma(X)}$. We prove that $\text{loss}_\gamma(X) = \delta \cdot d_\sigma + z(n) \geq \ell_{min}$ by case analysis on δ:

1. **[$\mathbf{sgn(\delta)}$ is zero]** As shown in the proof of Theorem 11.3.2, $n < c_\sigma + d_\sigma$ and the minimum value of the function $z(n)$ is 1, and is reached for n being 1.
2. **[$\mathbf{sgn(\delta)}$ 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, $\delta \cdot d_\sigma + \mathrm{sgn}(\delta) \cdot (c_\sigma - 1)$ is indeed a lower bound on $\mathrm{loss}_\gamma(X)$ when $\mathrm{sgn}(R)$ is zero.

- $\boldsymbol{\ell_{max} \geq \mathbf{loss}_\gamma(X)}$. We prove that $\mathrm{loss}_\gamma(X) \leq \ell_{max}$. 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 $\mathrm{loss}_\gamma(X)$.

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

Example 11.3.7 (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 11.3.3 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 $\mathrm{sgn}(R)$, the loss interval wrt $\langle \delta, \mathrm{sgn}(R) \rangle$ is $[2 \cdot \delta, 2 \cdot \delta + 1]$. △

Theorem 11.3.4 (disjointness 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 disjointness condition is satisfied for NB_σ.

Proof. The disjointness condition can be proved using the formula of the loss interval of Theorem 11.3.3. 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 disjointness condition. □

11.3.2.3 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 [68]. In order to satisfy the separation condition for the loss automaton for NB_σ, we require the seed transducer for σ be of a specific form that we introduce in Definition 11.3.8.

Definition 11.3.8 (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.

Example 11.3.8 (separated seed transducer)**.** Part (B) of Figure 11.5 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 [68] 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 register automata for time-series constraints [22], namely:

1. Section 11.3.2.3.1 identifies the required registers of $\mathcal{M}$ and their role.
2. With each phase letter of the output alphabet of the seed transducer for σ, Section 11.3.2.3.2 associates a set of instructions, i.e. register 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.

11.3.2.3.1 Identifying the Required Registers 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 11.3.9 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 11.3.9 (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 11.3.9 (regret of a transition)**.** Consider the PEAK regular expression, whose seed transducer is given in Part (B) of Figure 11.5. 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 $\{s \xrightarrow{<} r, r \xrightarrow{>} t, t \xrightarrow{>} r', r' \xrightarrow{<} t\}$ 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 11.3.2 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 11.3.2 (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:

$\text{loss}_\gamma(X) = n - 1 - t^* + \sum_{i=1}^{t^*} \rho(t_i)$, where $\rho(t_i)$ 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 $\sum_{i=1}^{t^*} \rho(t_i)$ and $n - 1 - t^*$. □

Example 11.3.10 (regret-loss transition)**.** Consider the PEAK regular expression, whose separated seed transducer $\mathcal{T}_{\text{PEAK}}$ is given in Part (B) of Figure 11.5. 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 11.3.2, we obtain $\text{loss}_\gamma(X) = 10-1-8+(1+0+0+0+0+1+0+0+0) = 3$. △

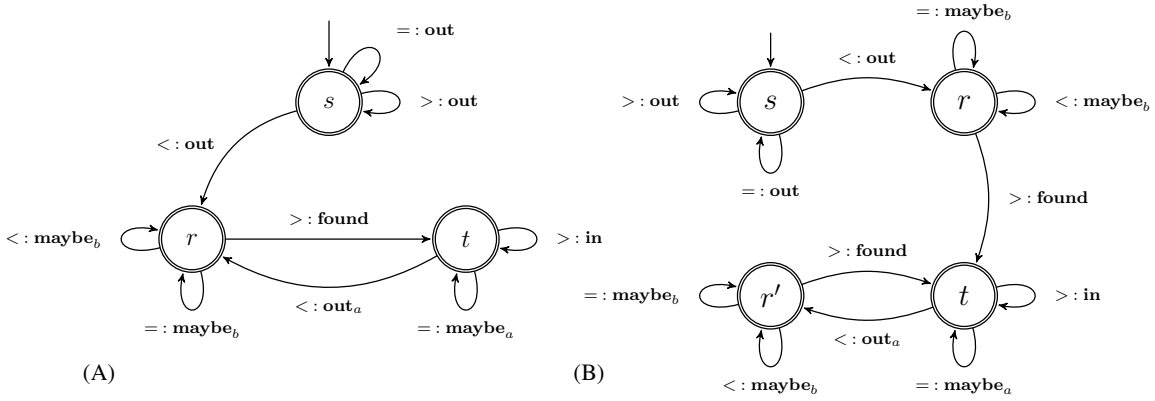


Figure 11.5 – Seed transducer (A) and separated seed transducer (B) for the PEAK regular expression.

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
$\texttt{out}_\texttt{r}$	$C \leftarrow C + 1$	$D \leftarrow D + \rho(t)$	
$\texttt{out}_\texttt{a}$	$C \leftarrow C + 1$	$D \leftarrow D + \rho(t)$	
$\texttt{maybe}_\texttt{b}$	$C \leftarrow C + 1$	$D \leftarrow D + \rho(t)$	
$\texttt{maybe}_\texttt{a}$	$C \leftarrow C + 1$	$D \leftarrow D + \rho(t)$	
found	$C \leftarrow 0$	$D \leftarrow 0$	$R \leftarrow R + D$
$\texttt{found}_\texttt{e}$	$C \leftarrow 0$	$D \leftarrow 0$	$R \leftarrow R + D$
in	$C \leftarrow C + 1$	$D \leftarrow D + \rho(t)$	
out	$C \leftarrow C + 1$	$D \leftarrow D + \rho(t)$	

Table 11.1 – 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.

From Lemma 11.3.2, three registers 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:

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

The initial value of these three registers is zero. The decoration table, given in the next section, follows from Lemma 11.3.2.

11.3.2.3.2 Decoration Table of a Loss Automaton As stated in Section 11.3.2.3.1, a loss automaton for NB_σ has three registers 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 register C must be increased by 1. Further, the value of D should be increased by the regret of t. Finally, register 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, registers C and D must both be reset to 0. Register 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 11.3.2, 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. Table 11.1 summarises how registers are updated.

In order to obtain a 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 register updates according to the decoration table in Table 11.1. The initial value of all the three registers is zero, and the acceptance function is $C + R$.

Example 11.3.11 (loss automaton)**.** A loss automaton for NB_PEAK, obtained from the seed transducer in Part (B) of Figure 11.5 and the decoration table in Table 11.1, is given in Figure 11.6. △

11.3.3 Deriving the δ-gap Automaton for the SUM_WIDTH_σ Family

In the context of the SUM_WIDTH_σ family, we show that, when the regular expression σ has a property, named the CONTINUITY *property,* the first three principal conditions of Definition 11.3.6 are satisfied.

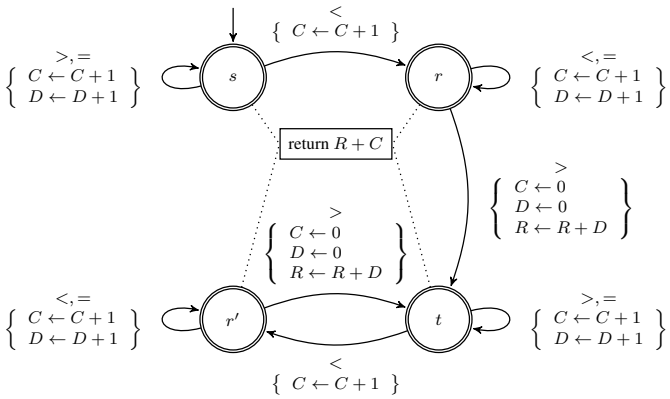


Figure 11.6 – Loss automaton for NB_PEAK. The initial value of the registers C, D, and R is zero.

In order to derive a loss automaton satisfying the non-negativity and the separation conditions a similar approach as for the NB_σ family can be applied. We do not detail it here, but it allowed to obtain a loss automata satisfying the non-negativity and the separation conditions for the 17 SUM_WIDTH_σ time-series constraints of [10]. The main problem for generating loss automata for the SUM_WIDTH_σ family is that we cannot use the seed transducers of [22] because of the mix of the quantitative and the qualitative aspects in those seed transducers, i.e. the parameter b_σ is encoded into the seed transducers of [22]. One of the consequences of such mix is that when a seed transducer consumes the signature of a time series it cannot detect whether a current signature symbol belongs to an occurrence of a regular expression or not. For example, in the seed transducer given in Part (A) of Figure 11.5, the transition from s to r is labelled with `out` whereas the consumed signature symbol '$<$' may belong to an occurrence of the regular expression, and this `out` is different from the one on the transitions from s to s. The new seed transducers, which will be presented in Chapter 12 does not have this problem, and can be used for generating the loss automata for the constraints of the SUM_WIDTH_σ family.

1. Section 11.3.3.1 introduces the CONTINUITY property.
2. Assuming the CONTINUITY property, Section 11.3.3.2 proves three theorems stating that, the gap-to-loss, the boundedness, and the disjointness conditions are satisfied for SUM_WIDTH_σ.

11.3.3.1 The CONTINUITY Property

Property 11.3.2 introduces the CONTINUITY *property* of a regular expression.

Property 11.3.2 (CONTINUITY property)**.** A regular expression σ has the CONTINUITY property iff σ has the $\overline{\text{WIDTH}}$-*sum* property, see Property 7.3.2, when there are no restrictions on the domains of time-series variables. It implies that, for the SUM_WIDTH_$\sigma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint, the maximum value of R is equal to $n - b_\sigma - a_\sigma$ if $n > \omega_\sigma$, and is 0, otherwise.

11.3.3.2 Verifying the Gap-Loss-Relation Conditions

This section shows that the gap-loss-relation conditions, see Definition 11.3.6, for a SUM_WIDTH_σ time-series constraint are satisfied, assuming σ has the CONTINUITY property. Lemma 11.3.3 first proves that, when the result of SUM_WIDTH_σ is zero, some gaps are infeasible. Then Theorem 11.3.5 proves the gap-to-loss condition and derives the formula for the gap-to-loss function. Theorem 11.3.6 proves the boundedness condition and derives the formula of the loss interval for a given gap and sign of the result value, and finally, Theorem 11.3.7 proves the disjointness condition.

Lemma 11.3.3 (infeasible gap values)**.** Consider a $\gamma(X, R)$ time-series constraint such that γ belongs to the SUM_WIDTH_σ family and σ has the CONTINUITY property. Then there does not exist a ground time series X such that

- it yields zero as the value of R, and
- $\mathrm{gap}_\gamma(X)$ belongs to the interval $[1, \omega_\sigma - b_\sigma - a_\sigma]$, where ω_σ is the size of σ.

Proof. We prove the lemma by contradiction. Assume that there exists a ground time series X of length n yielding zero as the value of R whose gap belongs to the interval $[1, \omega_\sigma - b_\sigma - a_\sigma]$. Then, by Definition 11.3.1, $\mathrm{gap}_\gamma(X)$ is the maximum value of R for a time series of length n. However, this is not possible since the smallest width of a σ-pattern is $\omega_\sigma - b_\sigma - a_\sigma + 1$, which is strictly greater than $\mathrm{gap}_\gamma(X)$. □

Theorem 11.3.5 (gap-to-loss condition)**.** Consider a SUM_WIDTH_$\sigma(X, R)$ time-series constraint, denoted by $\gamma(X, R)$, such that σ has the CONTINUITY 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:

$$\mathrm{loss}_\gamma(X) = \begin{cases} n - 1 & \text{if } \mathrm{sgn}(R) = 0 \text{ and } \mathrm{gap}_\gamma(X) = 0, \quad (11.6) \\ \mathrm{gap}_\gamma(X) + a_\sigma + b_\sigma - 1 & \text{if } \mathrm{sgn}(R) = 0 \text{ and } \mathrm{gap}_\gamma(X) > \omega_\sigma - b_\sigma - a_\sigma, \quad (11.7) \\ \mathrm{gap}_\gamma(X) & \text{if } \mathrm{sgn}(R) = 1. \quad (11.8) \end{cases}$$

Proof. By Lemma 11.3.3, when $\mathrm{sgn}(R)$ is zero there does not exist a ground time series whose gap would be in $[1, \omega_\sigma - b_\sigma - a_\sigma]$. Hence, we do not need to define our gap-to-loss function for these values. We successively consider two disjoint cases wrt $\mathrm{sgn}(R)$.

[$\mathrm{sgn}(R)$ is zero] Since a shortest ground time series yielding zero as the value of R is of length 1, the loss of X is $n - 1$. This gives the formula in (11.6).

When $\mathrm{gap}_\gamma(X)$ is positive, n is strictly greater than ω_σ, and from the CONTINUITY property, the maximum value of R is equal to $n - b_\sigma - a_\sigma$. By Definition 11.3.1, $\mathrm{gap}_\gamma(X)$ is equal to the difference between the maximum value of R for a ground time series of length n, and the value of R on X. Hence, $\mathrm{gap}_\gamma(X)$ is equal to $n - b_\sigma - a_\sigma$, and is equal to $\mathrm{loss}_\gamma(X) + 1 - b_\sigma - a_\sigma$. From this equality, we can isolate $\mathrm{loss}_\gamma(X)$, which is $\mathrm{gap}_\gamma(X) + a_\sigma + b_\sigma - 1$, namely formula in (11.7).

[$\mathrm{sgn}(R)$ is one] By definition of gap and from the CONTINUITY property, the value of $\mathrm{gap}_\gamma(X)$ is equal to $n - b_\sigma - a_\sigma - R$. Let k denote the length of a shortest time series such that there exists a ground time series of length k yielding the same value of R as X. Then the loss of X is equal to $n - k$. Further, the value of R is equal to $k - b_\sigma - a_\sigma$, and thus $\mathrm{gap}_\gamma(X)$ is equal to $n - k$. Hence, $\mathrm{gap}_\gamma(X)$ is equal to $\mathrm{loss}_\gamma(X)$, i.e. formula in (11.8). □

Theorem 11.3.6 (boundedness condition)**.** Consider a SUM_WIDTH_$\sigma(X, R))$ time-series constraint, denoted by $\gamma(X, R)$, such that σ has the CONTINUITY property. First, the boundedness condition is satisfied for γ; second, for any given gap δ and any value of $\mathrm{sgn}(R)$, the loss interval $[\ell_{min}, \ell_{max}]$ wrt $\langle \delta, \mathrm{sgn}(R) \rangle$ is defined by:

$$\ell_{min} = \begin{cases} 0 & \text{if } \mathrm{sgn}(R) = 0 \text{ and } \delta = 0, \\ \delta + a_\sigma + b_\sigma - 1 & \text{if } \mathrm{sgn}(R) = 0 \text{ and } \delta > \omega_\sigma - b_\sigma - a_\sigma, \\ \delta & \text{if } \mathrm{sgn}(R) = 1, \end{cases}$$

$$\ell_{max} = \begin{cases} \omega_\sigma - 1 & \text{if } \mathrm{sgn}(R) = 0 \text{ and } \delta = 0, \\ \delta + a_\sigma + b_\sigma - 1 & \text{if } \mathrm{sgn}(R) = 0 \text{ and } \delta > \omega_\sigma - b_\sigma - a_\sigma, \\ \delta & \text{if } \mathrm{sgn}(R) = 1. \end{cases}$$

Proof. **[$\mathrm{sgn}(R) = 0$ and $\delta = 0$]** Consider a ground time series X of length n yielding zero as the value of $\mathrm{sgn}(R)$ and whose gap is 0. In this case, n is less than or equal to ω_σ, the size of σ. By Theorem 11.3.5, the loss of X is equal to its length minus 1. Hence, it gives us the loss interval $[0, \omega_\sigma - 1]$.

[$\mathrm{sgn}(R) = 1$ or $\delta > \omega_\sigma - b_\sigma - a_\sigma$] When $\mathrm{sgn}(R) = 1$ or $\mathrm{sgn}(R) = 0$ and $\delta > \omega_\sigma - b_\sigma - a_\sigma$, by Theorem 11.3.5, there is a bijection between the values of $\mathrm{loss}_\gamma(X)$ and $\mathrm{gap}_\gamma(X)$. Hence, the loss interval contains a single value, which is the value of the gap-to-loss function for given values of $\mathrm{sgn}(R)$ and $\mathrm{gap}_\gamma(X)$. □

Theorem 11.3.7 (disjointness condition)**.** Consider a SUM_WIDTH_σ time-series constraint such that σ has the CONTINUITY property. The disjointness condition holds for SUM_WIDTH_σ.

Proof. It follows from the formulae of the loss interval of Theorem 11.3.6. □

11.3.4 Conclusion

We presented a systematic approach for generating δ-gap automata for time-series constraints, and demonstrated its applicability for the NB_σ and the SUM_WIDTH_σ families. We used the obtained automata for creating a database of invariants on conjunctions of time-series constraints.

Although, we did this work in the context of time-series constraints, 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$(\langle X_1, X_2, \ldots, X_n\rangle, R, P)$ constraint, where $\langle X_1, X_2, \ldots, X_n\rangle$ is a sequence of integer variables, R is an integer variable, and P is a finite subset of integer numbers. This constraint restricts R to be the number of maximal subsequences of X whose elements are in P. For example, the NB_GROUP$(\langle 1, 3, 4, 1, 0, 9, 0\rangle, 3, \langle 0, 1\rangle)$ constraint holds. If P is not empty, 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 11.3.1 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 11.3.1.

11.4 Generation of Constant-Size Automata for Not-Less ant Not-Greater Atomic Relations

Consider a time-series constraint $\gamma(X, R)$ and two atomic relations C_1 and C_2 of the form $R \geq d_1$ and $R \leq \mathrm{up}(R, n) - d_2$, respectively, with d_1 and d_2 being non-negative integer constants. In this section, we focus on the generation of constant-size automata $\mathcal{M}_{C_1}$ and $\mathcal{M}_{C_2}$ for C_1 and C_2, respectively. The main idea of our method for generation of $\mathcal{M}_{C_2}$ (respectively $\mathcal{M}_{C_2}$) is to use automata for constant (respectively gap) atomic relations, and automaton operations such as union, intersection, and complement, see Section 3.2.

Our algorithms for obtaining the automata $\mathcal{M}_{C_1}$ and $\mathcal{M}_{C_2}$ are very similar and both have 3 steps. Hence we now give both algorithms in parallel by making explicit in the brackets before a step for which automaton this step applies.

1. [Constructing $\mathcal{M}_{C_1}$] Construct the automaton $\mathcal{M}_i$ for every $R = i$ constant atomic relation (with i in $[0, d_1 - 1]$) using the method of Section 11.1.

   [Constructing $\mathcal{M}_{C_2}$] Construct the automaton $\mathcal{M}_{\mathrm{up}-i}$ for every $R = \mathrm{up}(R, n) - i$ gap atomic relation (with i in $[0, d_2 - 1]$) using the method of Section 11.3.

2. [Constructing $\mathcal{M}_{C_1}$] Take the union $\mathcal{M}_{<d_1}$ of all $\mathcal{M}_i$, which is an automaton for the $R \leq d_1$ atomic relation. If the interval $[0, d_1 - 1]$ is empty, i.e. d_1 is 0, then the automaton $\mathcal{M}_{<d_1}$ is empty.

   [Constructing $\mathcal{M}_{C_2}$] Take the union $\mathcal{M}_{>\mathrm{up}-d_2}$ of all $\mathcal{M}_{\mathrm{up}-i}$, which is an automaton for the $R > \mathrm{up}(R, n) - d_2$ relation. If the interval $[0, d_2 - 1]$ is empty, i.e. d_2 is 0, then the automaton $\mathcal{M}_{>\mathrm{up}-d_2}$ is empty.

3. [Constructing $\mathcal{M}_{C_1}$] Take the complement $\mathcal{M}_{\geq d_1}$ of $\mathcal{M}_{<d_1}$, which is the desired automaton for the $R \geq d_1$ not-less atomic relation.
   [Constructing $\mathcal{M}_{C_2}$] Take the complement $\mathcal{M}_{\leq \mathrm{up}-d_2}$ of $\mathcal{M}_{>\mathrm{up}-d_2}$, which is the desired automaton for the $R \leq \mathrm{up} - d_2$ not-greater atomic relation.

11.5 Conclusion

In this chapter, we presented systematic methods for generating conditional automata for five different types of atomic relations, namely constant atomic relations, modulo atomic relations, gap atomic relations, and not-less and not-greater atomic relations. Since all these automata have a number of states and an input alphabet that do not depend on the length of an input sequence these automata will allow us to prove the validity of synthesised non-linear invariants, described in Chapter 10, that are valid for any sequence length.

Summary of this Chapter:

The main contribution of this chapter is a systematic method for generating constant-size automata for different atomic relations of Chapter 10. Most of the construction schemes are straightforward, the only exception is automata for gap atomic relations, which required the introduction of the notions of loss of an time series and loss automata for a time-series constraints as well as principal conditions. The principal conditions define a class of constraints, for which our method for generating gap automata is applicable.

Chapter 12

Extended Transducer-Based Model

This chapter is the result of a collaboration with (in alphabetic order) Nicolas Bedliceanu, Mats Carlsson, Rémi Douence, Pierre Flener, María Andreína Francisco Rodríguez, Justin Pearson, and Helmut Simonis. The author of this thesis was one of the main researchers and writers of this work.

Motivated by representing a large number of sequence constraints, such as [25, 108], we extend an initial work [22, 10] that uses a manually designed transducer [119] on the way to automatically inducing a decomposition of a time-series constraint. Our aim is a concise normalised representation that is expressive enough to capture declaratively many sequence constraints, namely those constraining an aggregation of integer features of all maximal occurrences of a regular expression within an integer sequence, such as the minimal width of all its plateaus.

Our contribution is a regular-expression-based representation for such constraints. From such a representation, a seed transducer can be generated automatically. We not only extend the set of time-series constraints of [22, 10], but also cover most sequence constraints of the Global Constraint Catalogue [21], such as [25, 108].

From a transducer, a register automaton describing the computation of the function can be synthesised [22]. From a register automaton, a decomposition of the represented constraint in terms of basic constraints can be induced [20].

The chapter is organised as follows:

- Section 12.1 defines our regular-expression-based representation of the considered sequence constraints.
- Section 12.2 gives an operational view of such constraints, using for the regular expression a transducer whose output alphabet is a set of instructions denoting the computations for each phase of recognising all maximal occurrences of the regular expression within a sequence. The instructions use registers for recording information about past maximal occurrences of the regular expression, the current possibly unfinished maximal occurrence, and the hypothetical next maximal occurrence.
- Finally, Section 12.3 summarises our contributions and concludes.

12.1 Defining Functions over Integer Sequences

To define a function over integer sequences, we introduce the notion of an abstract pattern, which is a regular expression defined over an abstract alphabet. Further, we present the notion of a concrete pattern that can be associated with an abstract pattern. Then, we describe the parameters of a function over integer sequences, one of them being a concrete pattern. Afterwards, we restrict the values of the parameters describing a function wrt the considered pattern, in order to locate unambiguously each pattern occurrence and to avoid overlapping pattern occurrences. Finally, we define the evaluation of a function over integer sequences.

Definition 12.1.1 (abstract/concrete alphabets and pattern)**.** The *abstract alphabet* $\mathcal{A}$ of k letters is the set $\{0, 1, \ldots, k-1\}$. A *concretisation of* $\mathcal{A}$ is a bijection from $\mathcal{A}$ to a set $\{a_0, a_1, \ldots, a_{k-1}\}$, called the *concrete alphabet*. An *abstract pattern* over a finite abstract alphabet $\mathcal{A}$ is a regular expression over $\mathcal{A}$. A *concrete pattern* is obtained from an abstract pattern over an abstract alphabet $\mathcal{A}$ by applying a concretisation of $\mathcal{A}$.

Example 12.1.1 (abstract/concrete alphabets and pattern)**.** Consider the abstract alphabet $\mathcal{A} = \{0, 1\}$ and its concretisation C mapping 0 to '$\notin$' and 1 to '$\in$'. The concrete pattern '$\in\in^*$' is obtained by applying C to the abstract pattern '11^*'.

Definition 12.1.2 (parameterised function over integer sequences)**.** A *function over integer sequences* $\mathcal{F}$ is parameterised by $\langle \psi, \mathcal{S} \rangle$, $\langle f, g, h \rangle$, $\langle b, a \rangle$, $\langle balance \rangle$, and $\langle skip_0, skip_1 \rangle$, where:

- ψ is a concrete pattern over a concrete alphabet Σ, and $\mathcal{S}$ is a total surjective function of arity $p \in \mathbb{N}$, called the *signature function*, mapping $\mathbb{Z}^p$ to Σ;
- f, g, and h are respectively one of the features `one`, `width`, `surf`, `max`, `min`, `range`, one of the primary aggregators `sum`, `max`, `min`, `SumIf`, `CountIf`, and one of the secondary aggregators `Id`, `max`, `min` defined in Table 12.1;
- b and a are non-negative integers, whose role is to trim the left and right borders of maximal occurrences of the pattern ψ in an integer sequence;
- $balance \in \{0, 1\}$ indicates whether, for computing the feature value, we use only f (value 0) or both f and $-f$ (value 1);
- $skip_1$ (respectively $skip_0$) with $skip_0 \cup skip_1 \subset \Sigma$ and $skip_0 \cap skip_1 = \emptyset$ is a subset of skipped (respectively possibly skipped) symbols when computing the value of f depending on the phase of recognising ψ when computing function $\mathcal{F}$ on an integer sequence.

Example 12.1.2 (parameterised function over integer sequences)**.** Consider a function over integer sequences with the following parameters:

- The concrete pattern ψ is '$\in\in^*$' over the alphabet $\Sigma = \{\in, \notin\}$. For any integer sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$, the signature function $\mathcal{S}$ of arity 1 is defined as follows:

$$\mathcal{S}(X_i) = \begin{cases} \text{'}\in\text{'} & \text{if } X_i \in \{1\}, \\ \text{'}\notin\text{'} & \text{if } X_i \notin \{1\}. \end{cases}$$

- The feature f is `width`, the aggregator g is `max`, and the secondary aggregator is `Id`.
- The value of b and a is zero.
- The value of $balance$ is 0.
- The sets $skip_0$ and $skip_1$ are empty. △

Before defining the result value of a function over integer sequences, we extend the notions of *e-occurrence*, see Definition 5.1.1, which now depends on the new parameters $skip_0$ and $skip_1$, which allow us to skip some positions inside a pattern occurrence.

Definition 12.1.3 (s-occurrence)**.** Consider a concrete pattern ψ over a concrete alphabet Σ, a sequence $S = \langle S_1, S_2, \ldots, S_m \rangle$ over Σ, and a subsequence $\langle S_i, S_{i+1}, \ldots, S_j \rangle$, with $1 \leq i \leq j \leq m$, forming a maximal word in S that matches ψ. The *s-occurrence* of S is the index sequence $\langle i, i+1, \ldots, j \rangle$, denoted by $(i..j)$.

Example 12.1.3 (s-occurrence)**.** We give examples of s-occurrences of two different concrete patterns ψ_1 and ψ_2.

- Consider the ψ_1 = '$\in\in^*$' concrete pattern over the $\{\in, \notin\}$ concrete alphabet. Then the sequence $\langle \in, \notin, \notin, \in, \in, \notin, \notin, \notin \rangle$ contains two s-occurrences of ψ_1, namely $(1..1)$ and $(4..5)$.
- Consider the ψ_2 = '$< (< \mid =)^*(= \mid >)^* >$' concrete pattern over the $\{<, =, >\}$ concrete alphabet. Then the sequence $\langle =, =, <, =, >, >, =, <, >, <, =, <, >, = \rangle$ contains three s-occurrences of ψ_2, namely $(3..6)$, $(8..9)$, and $(10..13)$. △

Definition 12.1.4 (found index, e-occurrence, i-occurrence)**.** Consider a function $\mathcal{F}$ over integer sequences whose signature function $\mathcal{S}$ is of arity p, a concrete pattern ψ over an alphabet Σ, two integer constants 'b' and 'a', a subset $skip_0$ of Σ, and an integer sequence $X = \langle X_1, X_2, \ldots, X_n\rangle$ whose signature sequence is $S = \langle S_1, S_2, \ldots, S_{n-p+1}\rangle$ wrt $\mathcal{S}$. For any s-occurrence $(i..j)$ of ψ in S:

- the *found index* is the smallest index k in the interval $[i, j]$ such that the word $S_iS_{i+1}\ldots S_k$ belongs to $\mathcal{L}_\psi$;
- the *e-occurrence* is a set of indices in $\{i+b, \ldots, j+p-1-a\}$ such that an index m belongs to the e-occurrence iff both conditions hold:
  1. The signature symbol S_m is not in $skip_1$.
  2. If $m < k$ and S_m is in $skip_0$, then there exists a signature symbol $S_t \notin skip_0$ with $m < t < k$.
- the *i-occurrence* is the index sequence $\langle i{+}b, i{+}b{+}1, \ldots, j{+}p{-}1\rangle$, denoted by $[(i + b)..(j + p - 1)]$.

Example 12.1.4 (found index, e-occurrence, i-occurrence)**.** Consider the function over integer sequences $\mathcal{F}$ whose signature function has its arity p being 2, concrete pattern is ψ_2 from Example 12.1.3, the integer constants b and a both are zero, the set $skip_0$ is $\{=\}$, and the set $skip_1$ is empty. As we have shown in Example 12.1.3, the $\langle =, =, <, =, >, >, =, <, >, <, =, <, >, =\rangle$ signature sequence contains three s-occurrences of ψ, namely $(3..6)$, $(8..9)$, and $(10..13)$. Table 12.2 gives the found index, the e-occurrence and the i-occurrence corresponding to each of the three s-occurrences. △

Definition 12.1.5 (well-formed function)**.** A function $\mathcal{F}$ parameterised by $\langle \psi, \mathcal{S}\rangle$, $\langle f, g, h\rangle$, $\langle b, a\rangle$, $\langle balance\rangle$, and $\langle skip_0, skip_1\rangle$ is *well-formed* iff:

$$f = \texttt{range} \Rightarrow \forall X \in \mathbb{Z}^*,\ \forall \text{ e-occurrence } \{i_1, i_2, \ldots, i_\ell\} \text{ of } X \quad (X_{i_1} \le X_{i_2} \le \cdots \le X_{i_\ell} \ \vee\ X_{i_1} \ge X_{i_2} \ge \cdots \ge X_{i_\ell}) \tag{12.1}$$

$$balance = 1 \Rightarrow (f = \texttt{width} \vee f = \texttt{surf}) \ \wedge\ a = 0 \tag{12.2}$$

$$b < \min_{w\in\mathcal{L}_\psi} |w| \ \wedge\ b + a < \min_{w\in\mathcal{L}_\psi} |w| + p - 1 \tag{12.3}$$

$$b \ge \mathrm{o}_\psi \tag{12.4}$$

$$a \le p \ \wedge\ skip_1 \ne \emptyset \Rightarrow a \ge p - 1 \tag{12.5}$$

$$\forall p \in \overrightarrow{\psi}, \forall w \in \mathcal{L}_\psi, \exists v_1, v_2 \in \Sigma^* \ (p = v_1wv_2 \Rightarrow v_1w \in \mathcal{L}_\psi) \tag{12.6}$$

$$\exists c \in \mathbb{Z} \ (\mu_\psi \le c) \tag{12.7}$$

f	value	$\texttt{id}_f$	$\min_f$	$\max_f$
`one`	1	0	n/a	n/a
`width`	$j - i + 1$	0	0	$n + 1$
`surf`	$\sum_{k=i}^{j} X_k$	0	$-\infty$	$+\infty$
`max`	$\max_{k\in[i,j]} X_k$	$-\infty$	$-\infty$	$+\infty$
`min`	$\min_{k\in[i,j]} X_k$	$+\infty$	$-\infty$	$+\infty$
`range`	$\begin{pmatrix} \max_{k\in[i,j]} X_k \\ - \\ \min_{k\in[i,j]} X_k \end{pmatrix}$	0	0	$+\infty$

g	value	$\texttt{id}_{g,f}$
`sum`	$\sum_{k=1}^{m} f_k$	0
`max`	$\max_{k\in[1,m]} f_k$	$\min_f$
`min`	$\min_{k\in[1,m]} f_k$	$\max_f$
`SumIf`	$\sum_{k=1}^{m} (f_k \circ q) \cdot f_k$	0
`CountIf`	$\sum_{k=1}^{m} (f_k \circ q)$	0

h	value	$\mathrm{id}^h_{f,g}$
`Id`	$\texttt{id}_{g,f}$	$\texttt{id}_{g,f}$
`max`	$\max(\max_{k\in[1,m]} g_k, \texttt{id}_{g,f})$	$\texttt{id}_{g,f}$
`min`	$\min(\min_{k\in[1,m]} g_k, \texttt{id}_{g,f})$	$\texttt{id}_{g,f}$

Table 12.1 – Consider a sequence $X = \langle X_1, X_2, \ldots, X_n\rangle$. Left: features, their values computed from a subsequence $\langle X_i, X_{i+1}, \ldots, X_j\rangle$, their neutral, minimum and maximum values. Centre (respectively right): primary aggregators (respectively secondary aggregators), their values computed from a sequence $\langle f_1, f_2, \ldots, f_m\rangle$ (respectively $\langle g_1, g_2, \ldots, g_m\rangle$), and their identity values. Here, q is an integer parameter, and $\circ$ is a comparison operator.

s-occurrence	$(3..6)$	$(8..9)$	$(10..13)$
found index	5	9	13
e-occurrence	$\{3,6,7\}$	$\{8,9,10\}$	$\{10,11,12,13,14\}$
i-occurrence	$[3..7]$	$[8..10]$	$[10..14]$

Table 12.2 – The three s-occurrences of the '$< (< \mid =)^*(= \mid >)^* >$' concrete pattern in the $\langle =, =, <, =, >, >, =, <, >, <, =, <, >, = \rangle$ signature, and their corresponding found indices, e-occurrences and i-occurrences.

$$\forall s \in \Sigma\ (s \in \mathit{skip}_0 \cup \mathit{skip}_1) \Rightarrow$$
$$\forall w \in \mathcal{L}_\psi\ (s \notin \overrightarrow{w}) \wedge \tag{12.8}$$
$$\forall w \in \mathcal{L}_\psi\ (s \notin \overleftarrow{w}) \wedge \tag{12.9}$$
$$\forall w_1, w_2 \in \mathcal{L}_\psi, \forall z \in \Sigma^*\ (z \in \overrightarrow{w_1} \wedge z \in \overleftarrow{w_2} \Rightarrow s \notin \overleftrightarrow{z}) \wedge \tag{12.10}$$
$$\forall w \in \overrightarrow{\psi}, \forall e \in \Sigma, \forall z \in \Sigma^*((we \notin \overrightarrow{\psi} \wedge z \in \overleftarrow{we} \wedge z \in \overrightarrow{\psi}) \Rightarrow s \notin \overleftrightarrow{z}) \tag{12.11}$$

$$\varepsilon \in \mathcal{L}_\psi \ \Rightarrow\ \forall e \in \Sigma\ (e \in \mathcal{L}_\psi) \wedge a = p - 1 \tag{12.12}$$

To compute incrementally the value of the `range` feature Condition 12.1 enforces monotonous patterns. Condition 12.3 enforces every i-occurrence of ψ to be non-empty. Condition 12.4 imposes disjointness of any two i-occurrences of ψ. As we will see in Section 12.2.2.3, Condition 12.5 is required to compute the value of $\mathcal{F}$ from an integer sequence X by reading the signature sequence of X from left to right and by accessing one signature symbol at a time. By Condition 12.6, there is discontinuity in recognition of ψ, and it allows us to avoid any regular expression ψ whose language contains words v and w such that v is a proper factor of w, and after having read a prefix of w whose suffix is v we cannot decide whether v or w is a maximal occurrence of ψ. While extracting an occurrence of the pattern ψ for any possible mismatch, we need to know in advance the suffix length to keep, which is ensured by Condition 12.7. Condition 12.8 (respectively Condition 12.9) states that the first (respectively last) letter of any word in the language of ψ cannot be skipped. By Condition 12.10 symbols simultaneously belonging to two maximal occurrences of ψ cannot be skipped, and by Condition 12.11 symbols in a mismatch cannot be skipped either. Condition 12.12 is motivated by the fact that, when ε belongs to $\mathcal{L}_\psi$, then any sequence contains at least one occurrence of a pattern, even if its length is smaller than p. Definition 12.1.6 shows how this condition is used.

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

Definition 12.1.6 (function evaluation)**.** Consider a function $\mathcal{F}$ parameterised by $\langle \psi, \mathcal{S} \rangle$, $\langle f, g, h \rangle$, $\langle b, a \rangle$, $\langle \mathit{balance} \rangle$, $\langle \mathit{skip}_0, \mathit{skip}_1 \rangle$. For any integer sequence X, the result of $\mathcal{F}$ from X is (R_1, R_2) if $h \neq \mathtt{Id}$, R_1 otherwise, where R_1 (respectively R_2) is obtained by applying the aggregator g (respectively h) to the list $\langle f_1, f_2, \ldots, f_t \rangle$ (respectively $\langle g_1, g_2, \ldots, g_t \rangle$), where every g_i is equal to $g(f_1, f_2, \ldots, f_i)$, and every f_i is computed from to the e-occurrence i $\{i_1, \ldots, i_\ell\}$ as follows:

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

If the signature of X does not contain any s-occurrences of ψ, then R_1 (respectively R_2) is equal to $\mathtt{id}_{g,f}$ (respectively $\mathrm{id}^h_{f,g}$) according to Table 12.1. Note that when $\varepsilon \in \mathcal{L}_\psi$, we add a sequence of $p-1$ arbitrary integers at the end of the input sequence.

Example 12.1.5 (function evaluation)**.** Table 12.4 provides seven examples of well-formed functions. In examples ① and ②, the same abstract alphabet is associated with several concrete alphabets, and even with

f	ϕ_f	δ_f^i
`one`	1	1
`width`	$\lambda y, x.x + y$	1
`surf`	$\lambda y, x.x + y$	X_i
`max`	$\lambda y, x. \max(x, y)$	X_i
`min`	$\lambda y, x. \min(x, y)$	X_i
`range`	$\lambda y, x.x + y$	$\lvert X_i - X_{i+1} \rvert$

g	ϕ_g
`max`	$\lambda y, x. \max(x, y)$
`min`	$\lambda y, x. \min(x, y)$
`sum`	$\lambda y, x.x + y$
`SumIf`	$\lambda y, x.(x \circ q?y + x : y)$
`CountIf`	$\lambda y, x.(x \circ q?y + 1 : y)$

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

Table 12.3 – (Left) Features and their operators ϕ_f and δ_f^i. (Center) (respectively Right) Aggregators (respectively secondary aggregators) and their operators ϕ_g (respectively ϕ_h), where $\circ$ stands for a comparison operator and q for an integer.

Abstract alphabet	Abstract pattern	Arity	Concrete alphabet	Concrete pattern	Concrete function	...
$\langle 0, 1\rangle$	11^*	2	$\langle \leq, > \rangle$	$>>^*$	$\langle$`one, sum, Id`$, 0, 0, 0, \emptyset, \emptyset\rangle$	①
		1	$\langle \notin, \in \rangle$	$\in\in^*$	$\langle$`width, max, Id`$, 0, 0, 0, \emptyset, \emptyset\rangle$	②
$\langle 0, 1, 2\rangle$	$0(1\lvert 0)^*(2\lvert 1)^*2$	2	$\langle <, =, > \rangle$	$< (= \mid <)^*(> \mid =)^* >$	$\langle$`surf, max, Id`$, 0, 0, 1, \{=\}, \emptyset\rangle$	③
		2	$\langle >, =, < \rangle$	$> (= \mid >)^*(< \mid =)^* <$	$\langle$`width, sum, Id`$, 1, 1, 0, \emptyset, \emptyset\rangle$	④
$\langle 0, 1\rangle$	$1^*0\lvert 1^*$	2	$\langle =, \neq \rangle$	$=^* \neq \mid =^*$	$\langle$`one, sum, Id`$, 0, 1, 0, \emptyset, \emptyset\rangle$	⑤
$\langle 0\rangle$	0	1	$\langle \top \rangle$	$\top$	$\langle$`surf, sum, min`$, 0, 0, 0, \emptyset, \emptyset\rangle$	⑥
$\langle 0, 1\rangle$	1	k	$\langle \notin, \in \rangle$	$\in$	$\langle$`one, sum, Id`$, 0, 0, 0, \emptyset, \emptyset\rangle$	⑦

Table 12.4 – Examples of functions, where $\mathcal{F}_①, \mathcal{F}_②, \ldots, \mathcal{F}_⑦$ stand for NB_STRICTLY_DE- CREASING_SEQUENCE, MAX_WIDTH_GROUP, MAX_SURF_BALANCE_PEAK, SUM_WIDTH_VALLEY, NB_STRETCH, MIN_SUM_SURF_TRUE and NB_IN.

signatures of different arities: in ①, $sig_2(X_i, X_{i+1}) = \text{`}\leq\text{'} \Leftrightarrow X_i \leq X_{i+1} \wedge sig_2(X_i, X_{i+1}) = \text{`}>\text{'} \Leftrightarrow X_i > X_{i+1}$, while in ②, $sig_1(X_i) = \text{`}\notin\text{'} \Leftrightarrow X_i \notin \mathcal{V} \wedge sig_1(X_i) = \text{`}\in\text{'} \Leftrightarrow X_i \in \mathcal{V}$ where $\mathcal{V}$ is a set of integers. $\mathcal{F}_① \left(\left\langle 1, \boxed{1,0}, 0, \boxed{1,0}, 0, 1 \right\rangle\right) = 2$ since we have two maximal occurrences of '$>>^*$' (highlighted in grey), and $\mathcal{F}_② \left(\left\langle 0, \boxed{1}, 0, \boxed{1,1} \right\rangle\right)$ with $\mathcal{V} = \{1\}$ is equal to 2 since the maximum number of consecutive ones is 2 (also highlighted). The patterns associated with ③ and ④ correspond to the PEAK and VALLEY regular expressions. $\mathcal{F}_③ \left(\left\langle \boxed{0,1,1,1}, 2, \boxed{1,0}, \boxed{0,1}, 2, 2, \boxed{1,1,0} \right\rangle\right) = 2$, i.e. the maximum difference $\max(\lvert \boxed{3} - \boxed{1} \rvert, \lvert \boxed{1} - \boxed{2} \rvert)$ of the surface located before/after each peak with 5 (respectively 11) being the found index of the first (respectively second) s-occurrence. $\mathcal{F}_④ \left(\left\langle 0, 1, \boxed{0}, 1, 1, 1, \boxed{0,0,0}, 1 \right\rangle\right) = 4$, the sum of the widths $\boxed{1} + \boxed{3}$ of the 2 valleys.

$\mathcal{F}_⑤ \left(\left\langle \boxed{0}, \boxed{1,1,1}, \boxed{0}, \boxed{1}, \boxed{0}, \boxed{1} \right\rangle\right) = 6$ since we have 6 maximal groups of consecutive identical values. Note that $\mathcal{F}_⑤(\langle 0\rangle) = 1$ since, from Condition 12.12 of Definition 12.1.6, when $\varepsilon \in \mathcal{L}_{=^*\neq\mid=^*}$ and the arity of the signature is 2, we add one integer value at the end of the input sequence.

In ⑥, $sig_1(X_i) = \text{`}\top\text{'}$ means that every index i of the signature sequence of any input sequence X is an e-occurrence. $\mathcal{F}_⑥(X) = \langle 0, 0\rangle$, where $X_i = 1$ (respectively $X_i = -1$) represents an opening (respectively closing) parenthesis models well-formed expressions with parentheses.

In ⑦, given low, up in $\mathbb{Z}$, $sig_{\text{k}}(X_i, X_{i+1}, \ldots, X_{i+k-1}) = \text{`}\in\text{'} \Leftrightarrow \sum_{\alpha=i}^{i+k-1} X_\alpha \in [low, up]$. $\mathcal{F}_⑦(X)$ returns the number of sliding sequences of k consecutive values of X, whose sum is located in the interval $[low, up]$. △

12.2 Operational View of Functions Over Integer Sequences

To evaluate a function $\mathcal{F}$ wrt an integer sequence X, i.e. see Definition 12.1.6, we need to 1) find all s-occurrences of the pattern ψ of $\mathcal{F}$ in the signature sequence of X, and 2) obtain the corresponding e-occurrences to compute the feature values and aggregate them. The qualitative (respectively quantitative)

part 1) (respectively part 2)) is called the *recognition* (respectively *computational*) aspect of $\mathcal{F}$. Note that the recognition aspect of $\mathcal{F}$ is only related to its pattern ψ and its alphabet Σ.

We describe in Section 12.2.1 an extended *seed transducer*, for dealing with the recognition aspect of $\mathcal{F}$. Then we show in Section 12.2.2 how the computational aspect of $\mathcal{F}$ is handled by a *reduced instruction set* based on the output alphabet of the seed transducer. This set of instructions is parameterised by all the parameters of $\mathcal{F}$, except the pattern ψ. It allows one to synthesise a register automaton with a constant number of registers, which returns the value of $\mathcal{F}$ from an integer sequence X after having consumed the signature sequence of X. Hence it takes linear time in the length fo X to compute the value of $\mathcal{F}$ from X.

12.2.1 Handling the Recognition Aspect: Seed Transducer

To find all s-occurrences of a pattern in an integer sequence, in the corresponding signature sequence, we extend the notion of *seed transducer* that was recalled in Section 5.2.1.

First, we describe a seed transducer for an abstract pattern, and show how to obtain the seed transducer for any concrete pattern from the seed transducer for the corresponding abstract pattern. Second, we give the conditions of *well-formedness* of a seed transducer wrt any given pattern, as well as wrt a given abstract pattern.

12.2.1.1 Describing the Seed Transducer of an Abstract Pattern

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

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

Besides the phase letters $\texttt{in}$ and $\texttt{maybe}_{\texttt{a}}$ whose meaning was left unchanged compared to the seed transducers described in Section 5.2.1, we have the following modifications:

- Some transitions that were labelled with $\texttt{out}$ are now labelled with $\texttt{maybe}_{\texttt{b}}$. For example, in [22], given the pattern '>><>>' this was the case for the two transitions recognising the first two occurrences of '>'; but to make the seed transducer independent of b they are now labelled with $\texttt{maybe}_{\texttt{b}}$.
- The letter $\mathbf{maybe}_{\texttt{r}}^{k}$ was not in the output alphabet of [22]. It has been added in order to capture patterns that require to restart from a small fixed suffix after a mismatch. It also replaces the first occurrence of $\texttt{maybe}_{\texttt{b}}$.
- Furthermore, since in [22], any seed transducer could only produce a single phase letter per transition, the authors had to introduce the letter $\texttt{found}_{\texttt{e}}$, which was a combination of $\texttt{found}$ and $\texttt{out}_{\texttt{a}}$. In our new model, this phase letter has disappeared since it is no longer needed. In fact, the same transition may now be labelled with more than one phase letter. For example, given the pattern σ = '>><>>', the transition associated with the recognition of σ is labelled by the input symbol '>', i.e. the last symbol of σ, and the output word '$\texttt{found}\,\texttt{out}_{\texttt{a}}\,\mathbf{maybe}_{\texttt{r}}^{2}$': $\texttt{found}$ indicates

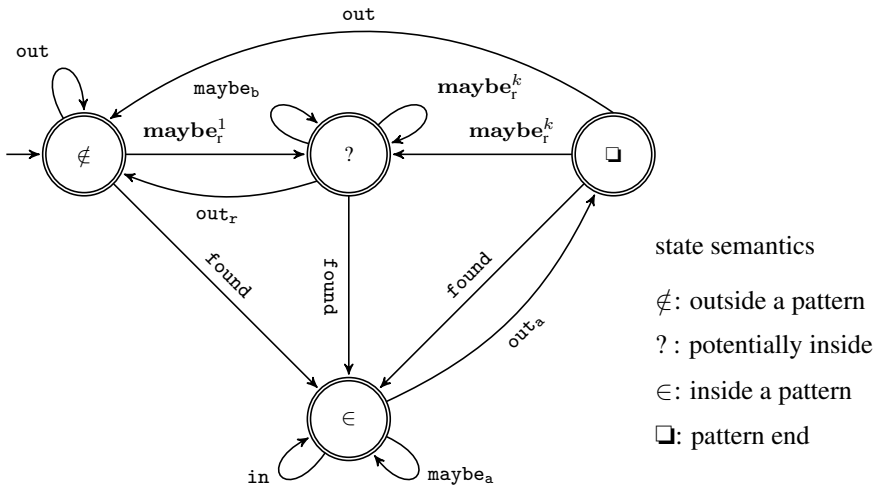


Figure 12.1 – Well-formed output language

that a new occurrence of σ was found, $\mathtt{out_a}$ denotes that this new occurrence ended, and $\mathbf{maybe}_{\mathrm{r}}^{2}$ indicates that potentially there is a next occurrence of σ whose prefix corresponds to the last two encountered input symbols, i.e. '$>>$'.

New seed transducers are free of the quantitative aspects of $\mathcal{F}$, i.e. the parameters b and a, and they are capable of handling a larger class of regular expressions.

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

12.2.1.2 Well-Formed Seed Transducer

We describe the structural properties a seed transducer must have. Condition 1 of Definition 12.2.1 implies that it is always possible in the future to have an occurrence of pattern σ, Condition 2 defines a partial order between the different phase letters of the same pattern occurrence, Condition 3 forbids the sequence $\mathbf{maybe}_{\mathrm{r}}^{k}\mathbf{maybe}_{\mathrm{r}}^{k+1}$, which can be replaced by $\mathbf{maybe}_{\mathrm{r}}^{k}\mathtt{maybe_b}$.

Definition 12.2.1 (necessary conditions for a well-formed seed transducer)**.** A seed transducer $\mathcal{S}$ is *well-formed* if all the following conditions hold:

1. There is a path from each transition to each transition labelled by a `found`.
2. The output language is accepted by the automaton in Figure 12.1.
3. For any state q we cannot have simultaneously a transition labelled by $\mathbf{maybe}_{\mathrm{r}}^{k}$ entering q, and a transition labelled by $\mathbf{maybe}_{\mathrm{r}}^{k+1}$ exiting q.

12.2.1.3 Well-Formed Seed Transducer wrt an Abstract Pattern

We introduce the notion of a *well-formed seed transducer wrt an abstract pattern* σ, which guarantees that a seed transducer recognises all maximal occurrences of an abstract pattern. We first present the notion of t-occurrence as an interval of indices of specific words in the output sequence of the seed transducer. Further, we state that, for any path v leading to a state q, the length of the longest suffix in $\overrightarrow{\sigma}$ of the sequence of input symbols of the transitions of v is either 1) a constant and is smaller than $\overline{b}_\sigma + 1$, or 2) is greater than or equal to $\overline{b}_\sigma + 1$, where $\overline{b}_\sigma$ is the largest value of b of a well-formed function whose concrete pattern is obtained from the abstract pattern σ. Note that, by Definition 12.1.5, such $\overline{b}_\sigma$ always exists and depends only on σ.

Definition 12.2.2 (t-occurrence)**.** Given a seed transducer $\mathcal{S}$ for some abstract pattern over an abstract alphabet $\mathcal{A}$ and an input sequence of symbols of $\mathcal{A}$, the *t-occurrence* of $\mathcal{S}$ for s consists of the indices of the phase letters of a maximal word within the transduction of s that matches the regular expression ‘ $(\varepsilon|\mathbf{maybe}_{\mathrm{r}}^{k}\mathtt{maybe_b}^{*})\mathtt{found}(\mathtt{maybe_a}^{*}\mathtt{in})^{*}$ ’.

Definition 12.2.3 ($\mathtt{maybe_b}$-degree of a path)**.** Consider an abstract pattern σ, and a *path* v, a sequence of consecutive transitions, wrt its seed transducer $\mathcal{T}_\sigma$.

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

Definition 12.2.4 ($\mathtt{maybe_b}$-degree of a state)**.** Consider an abstract pattern σ and its seed transducer $\mathcal{T}_\sigma$. For every state q of $\mathcal{T}_\sigma$, if every path from the initial state of $\mathcal{T}_\sigma$ to the state q has the same $\mathtt{maybe_b}$-degree d, then the $\mathtt{maybe_b}$-*degree of* q is equal to d; otherwise, the $\mathtt{maybe_b}$-degree of q is undefined.

Definition 12.2.5 (necessary conditions for a well-formed seed transducer wrt a pattern)**.** A seed transducer $\mathcal{S}$ is *well-formed wrt an abstract pattern* σ over an alphabet $\mathcal{A}$ if all the following conditions hold:

1. It is well-formed in the sense of Definition 12.2.1.
2. For any state of $\mathcal{S}$, its $\mathtt{maybe_b}$-degree is defined.
3. For any input sequence S of symbols of $\mathcal{A}$, for any t-occurrence $[[i..j]]$ of $\mathcal{S}$, there exists an s-occurrence $(i-k+1..j)$ of σ in S, where k is the degree of $\mathbf{maybe}_{\mathrm{r}}^{k}$, if the t-occurrence $[[i..j]]$ has one, and is 1, otherwise.

Example 12.2.1 (seed transducers well-formed wrt a pattern)**.** Parts (A) – (E) of Figure 12.2 respectively give the seed transducer for ‘ $>=^{+}>$ ’, ‘ $>><<>>$ ’, ‘ $=^{*}\neq \mid =^{*}$ ’ (the STRETCH pattern in ⑤), ‘ $\in^{+}$ ’ (the GROUP pattern in ②) and ‘ $<^{+} \mid >^{+}$ ’ patterns. The minimum and maximum values of b and a are set up according to Conditions 12.3 and 12.4 of Definition 12.1.5. In Part (A) of Figure 12.2, the $\mathtt{maybe_b}$-degree of states s, r, t and t' is respectively equal to 0, 1, 2 and 3. Note that states t and t' cannot be merged since, according to Definition 12.2.4, the $\mathtt{maybe_b}$-degree of the merged state would be undefined. In Part (B) of Figure 12.2, the $\mathtt{maybe_b}$-degree of states s, r, t, u and v is respectively equal to 0, 1, 2, 3 and 4. In Parts (C) – (E) of Figure 12.2 the $\mathtt{maybe_b}$-degree of all states is 0, since the corresponding seed transducers do no mention neither $\mathtt{maybe_b}$, nor $\mathbf{maybe}_{\mathrm{r}}^{k}$. △

12.2.2 Handling the Computational Aspect: Reduced Instruction Set

For a well-formed function $\mathcal{F}$ whose concrete pattern is ψ, and for an integer sequence X, knowing where are located the s-occurrences of σ in X, we can compute the value of $\mathcal{F}$. Our aim is to perform a single pass to both 1) detect all s-occurrences of ψ in X, and 2) compute $\mathcal{F}$ on the fly from the subsequences of X corresponding to e-occurrences. To do so, we describe a *reduced instruction set* for computing $\mathcal{F}$ that is associated with the phase letters. The reduced instruction set manipulates registers, whose values can be updated by performing *micro instructions*. When the seed transducer for ψ consumes the next symbol of an input signature sequence, a sequence of micro instructions is executed, which is called a *macro instruction*.

- In our model, we consider 5 registers described in Section 12.2.2.1.
- The reduced instruction set has 4 micro instructions described in Section 12.2.2.2.
- The macro instructions corresponding to the phase letters of the seed transducer for ψ are described in Section 12.2.2.3.

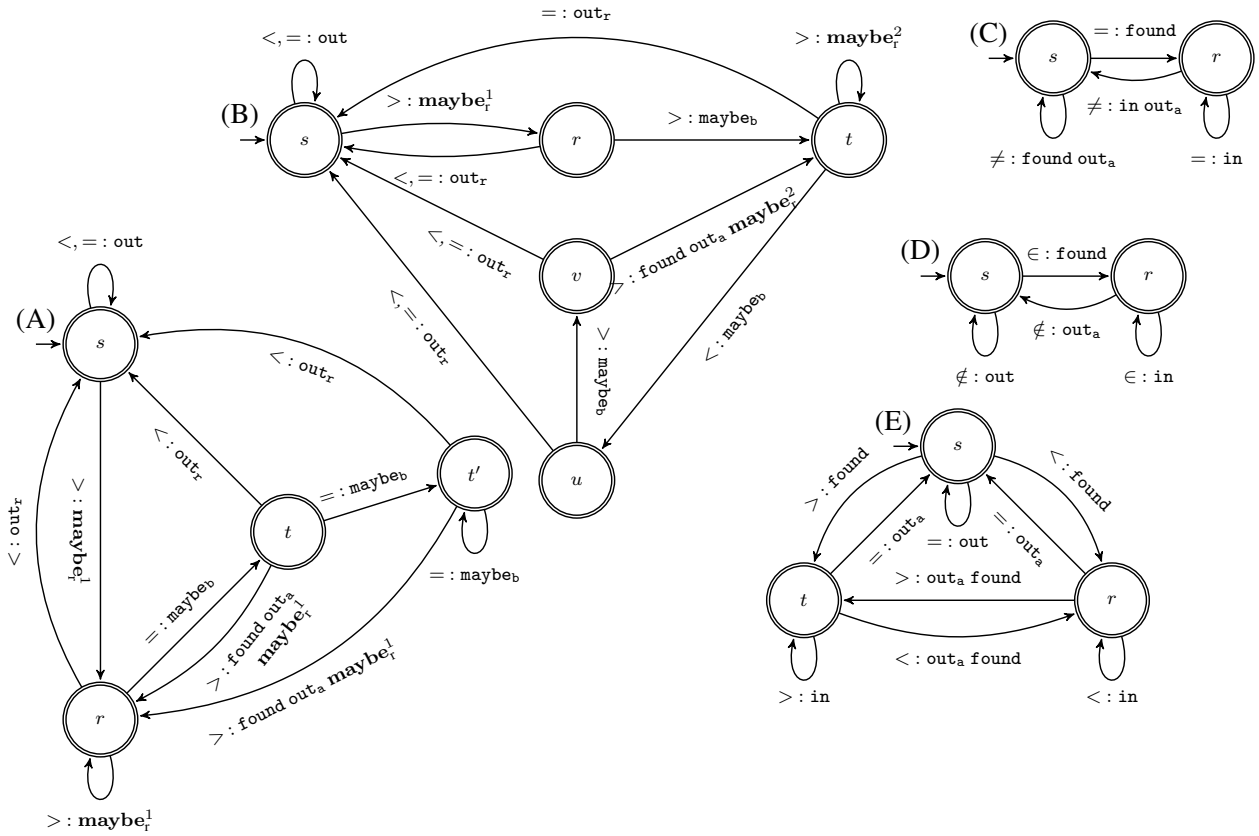


Figure 12.2 – Seed transducers for (A) ‘ $>=^+>$ ’ with $b \in [1,2]$, $a \in [0,2]$ and the alphabet $\{<,=,>\}$, (B) ‘ $>><<>>$ ’ with $b \in [0,4]$, $a \in [0,2]$ and the alphabet $\{<,=,>\}$, (C) ‘ $=^*\neq \mid =^*$ ’ with $b = 0$, $a = 1$ and the alphabet $\{=,\neq\}$, (D) ‘ $\in^+$ ’ with $b = 0 = a = 0$ and the alphabet $\{\in,\notin\}$, (E) ‘ $<^+ \mid >^+$ ’ with $b = 0$, $a \in [0,1]$ and the alphabet $\{<,=,>\}$.

12.2.2.1 Registers of the Reduced Instruction Set

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

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

With each level ℓ (with $\ell \in [0, 4]$) we associate a register V_ℓ and a function ϕ_ℓ defined according to Table 12.3 as follows:

- ϕ_4 is ϕ_h (with $h \in \{\texttt{max}, \texttt{min}, \texttt{Id}\}$) and the initial value of V_4 is $\mathrm{id}^h_{f,g}$.
- ϕ_3 is ϕ_g (with $g \in \{\texttt{max}, \texttt{min}, \texttt{sum}\}$) and the initial value of V_3 is $\mathtt{id}_{g,f}$.
- ϕ_0, ϕ_1 and ϕ_2 correspond all to ϕ_f (with $f \in \{\texttt{max}, \texttt{min}, \texttt{one}, \texttt{surf}, \texttt{width}\}$), and the initial value of V_0, V_1 is $\mathtt{id}_f$, while the initial value of V_2 is $\mathtt{id}_{g,f}$.

12.2.2.2 Micro Instructions of the Reduced Instruction Set

The next table describes the available micro instructions for modifying register values:

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

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

Note that all quantities $\phi_\ell(\delta^{i-k+1+b}_f, \ldots, \delta^i_f)$, where k is an integer value occurring in the $\mathbf{maybe}^k_{\mathrm{r}}$ phase letter of a seed transducer and $i \in [k - b, n - p + 1]$ is the index of the current signature symbol we are processing, used in the 'set' micro instruction, are computed in advance in an initialisation phase in linear time wrt the sequence length so that they are directly available. Note also that, similarly to [22], each micro instruction can be turned into a constraint in order to induce a reformulation of the original sequence constraint.

12.2.2.3 Macro Instructions of the Reduced Instruction Set

Consider a well-formed function $\mathcal{F}$ and its concrete pattern ψ. We describe the macro instructions associated with each phase letter of the seed transducer for ψ. A macro instruction may sometimes depend on the $\texttt{maybe}_\texttt{b}$-degree, denoted d in the next table, of the destination state of a transition labelled by the corresponding phase letter. The next table defines the macro instructions where the functions κ and ξ are defined just after. Some of the macro instructions have a precondition which must hold in order to execute its corresponding code.

letter		precondition	macro instruction code
`out`	:		
$\texttt{maybe}_\texttt{b}$	:	$\left(\begin{array}{l} s \notin \mathit{skip}_0 \wedge \\ s \notin \mathit{skip}_1 \wedge \\ d > b \end{array}\right)$	$\texttt{compute}\left(1, 0, \delta_f^i\right), \texttt{transmit}(0,0,0), \texttt{reset}(0)$
		$\left(\begin{array}{l} s \in \mathit{skip}_0 \wedge \\ d > b \end{array}\right)$	$\texttt{compute}\left(0, 0, \delta_f^i\right)$
$\mathbf{maybe}_\texttt{r}^k$	:	$s \notin \mathit{skip}_1$	$\texttt{reset}(1), \texttt{set}(1, k)$
$\texttt{out}_\texttt{r}$	:		$\texttt{reset}(1)$
`found`	:		$\texttt{compute}(1, \mathit{balance}, \kappa), \texttt{transmit}(1,0,1), \texttt{reset}(1)$
$\texttt{maybe}_\texttt{a}$	:	$s \notin \mathit{skip}_1$	$\texttt{compute}\,(1, \mathit{balance}, \xi)$
`in`	:		$\texttt{compute}\,(1, \mathit{balance}, \xi), \texttt{transmit}(0,0,1), \texttt{reset}(1)$
$\texttt{out}_\texttt{a}$	:		$\texttt{transmit}(0, \mathit{balance}, 2), \texttt{transmit}(0,0,3), \texttt{reset}(2)$

The functions ξ and κ of the macro instructions are defined as follows, where i is the index of the current input symbol we are processing:

precondition	κ
$f = \texttt{range} \wedge p - 1 - a < 1$	$\texttt{id}_f$
$f = \texttt{range} \wedge p - 1 - a = 1$	δ_f^i
$f = \texttt{range} \wedge p - 1 - a > 1$	$\phi_f(\delta_f^i, \ldots, \delta_f^{i+p-1-a})$
$f \neq \texttt{range} \wedge p - 1 - a < 0 \wedge \mathit{balance} = 0$	$\texttt{id}_f$
$f \neq \texttt{range} \wedge p - 1 - a = 0 \wedge \mathit{balance} = 0$	δ_f^i
$f \neq \texttt{range} \wedge p - 1 - a > 0 \wedge \mathit{balance} = 0$	$\phi_f(\delta_f^i, \ldots, \delta_f^{i+p-1-a})$
$f \neq \texttt{range} \wedge \mathit{balance} = 1$	δ_f^{i+p-1}

precondition	ξ
$f = \texttt{range}$	δ_f^{i-a}
$f \neq \texttt{range}$	$\delta_f^{i+p-1-a}$

12.2.2.4 Value Returned by the Function

After having consumed an input signature the function performs the macro instruction of the $\texttt{out}_\texttt{a}$ phase letter. If h is not `Id`, then the function returns a pair of values consisting of the last values of the registers V_3 and V_4. If h is `Id`, then the function only returns the last value of the register V_3.

Figure 12.3 illustrates the evaluation of the function described by Example ② of Table 12.4. It provides the phase letters produced by transducer (D) of Figure 12.2, as well as the corresponding sequence of micro-instructions updating the three registers V_1, V_2 and V_3.

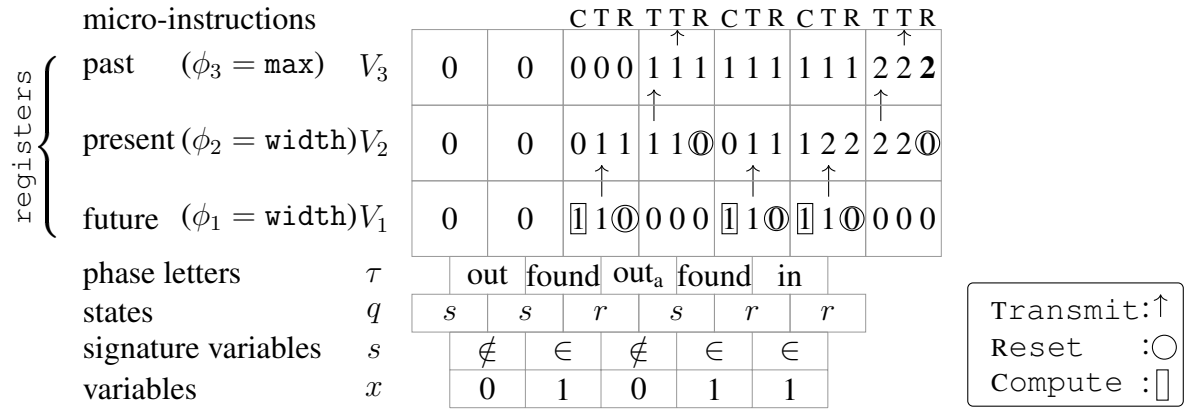


Figure 12.3 – Trace for the MAX_WIDTH_GROUP constraint, i.e. Example ② of Table 12.4 on the sequence $\langle 0, 1, 0, 1, 1\rangle$: evolution of the register values V_1, V_2, V_3 while executing the micro-instructions `Compute`, `Reset` and `Transmit` leading to the result **2** shown in bold on the right upper corner (since they are not relevant for this example, registers V_0 and V_4 are not shown)

12.3 Conclusion, Related Work, and Future Work

Our contributions over related work can be summarised as follows:

1. First, we have extended the *qualitative* aspect of the transducer-based computational model of [22]. The input alphabet of transducers is not fixed to $\{<, =, >\}$, that is the binary topological comparison operators that are useful for time-series constraints, but can be any set of operators, including unary ones (such as $\{\in, \notin\}$ with fixed sets, used in [5]) and k-ary ones (as frequently used in the Global Constraint Catalogue [21]). The output alphabet of transducers is augmented by $\mathbf{maybe}_{\mathrm{r}}$ and simplified, since transducers can output more than one letter for each input symbol.
2. Second, we have parameterised the *quantitative* aspect of the computation:
   - The model of [22] had parameters for trimming the borders of a maximal occurrence of a regular expression, with the major drawback that transducers were dependent on these parameters devoted to the quantitative aspect of the computation. In the new model, transducers are independent of such trimming parameters.
   - Within a maximal occurrence of a regular expression, depending on the current recognition phase, a function f or its opposite $-f$ may now be used for computing the contribution of an input letter to the feature value.

   While regular expressions and transducers are already used in the context of frequent sequence mining [17], they are focussed on the qualitative aspect, i.e. they do not compute a value for each pattern occurrence.
3. Third, the small number of phase letters and the very small set of micro instructions allow a compact implementation of checkers and reformulation.

While learning from a large collection of examples can be done with neural networks without assuming any bias, learning from very few examples still require having a proper bias. Consequently, future work may exploit the canonical form introduced in this paper to acquire constraint models involving functional constraints on integer sequences both from very few samples [32] and with a limited number of queries [38].

Summary of this Chapter:

The main contribution of this chapter is an extended transducer-based model describing sequence constraints. In this model we introduce new features, and new aggregators and allow an arbitrary signature function, which enlarges the class of global constraints that can be described using our approach. In addition, the extended model does not suffer from the mix of the qualitative (recognition of a regular expression) and the quantitative (feature and aggregator computation) aspects.

Part III

Practical Evaluation of our Contributions

In this part, we evaluate the impact of the obtained combinatorial objects on the propagation of time-series constraints.

We do it by comparing the previous state of the art to the state of the art with our new synthesised combinatorial objects. In every presented benchmark, we use *glue constraints* [8], recalled in Section 5.2.3. For every contribution, we do a systematic benchmark where for every time-series constraint (respectively a pair of time-series constraints), we try all assignments of its result variable (respectively their result variables) from some finite set and either find a feasible solution or prove infeasibility. For linear invariants, we also do a benchmark related to generation of time series for an electricity provider.

Although most of our implied constraints can be used straight away in the context of linear programming, we could not evaluate them in the LP context because we do not have linear glue constraints.

We now give the comparison made in every chapter of this part together with citations of the papers in which this comparison was made, and the contribution of the author of this thesis:

1. Chapter 13 compares register automata alone and register automata with bounds and glue constraints [8, 14]. We do a systematic benchmark for every time-series constraint. The author of this thesis provided bound formulae and their symbolic representation in Prolog.
2. Chapter 14 compares on the one hand register automata with bounds and glue constraints and on the other hand register automata with bounds, glue constraints, and AMONG implied constraints [12]. We do a systematic benchmark for every time-series constraint belonging either to the MAX_SURF_σ or to the SUM_SURF_σ family. The author of this thesis provided AMONG implied constraints, their implementation in Prolog, and also run the code to obtain the comparison.
3. Chapter 15 compares on the one hand register automata with bounds and glue constraints and on the other hand register automata with bounds, glue constraints, and linear invariants [13]. We do a systematic benchmark for every pair time-series constraint belonging either to the SUM_WIDTH_σ or to the NB_σ family, and also one practical benchmark. The author of this thesis developed and implemented in Prolog a method for synthesising linear invariants.
4. Chapter 16 compares on the one hand register automata with bounds, glue constraints and linear invariants and on the other hand register automata with bounds, glue constraints, linear invariants, and non-linear invariants. We do a systematic benchmark for every pair of time-series constraints belonging either to the SUM_WIDTH_σ or to the NB_σ family. The author of this thesis developed and implemented in Prolog a method for synthesising conditional automata, and also the method for proving non-linear invariants.

Chapter 13

Evaluation of the Impact of Bounds

This chapter is adapted from an article published in the *Constraints* journal [14]. The final authenticated version of this article is available online at: `http://dx.doi.org/10.1007/s10601-017-9276-z`.

We evaluate the impact of bounds on the result values of time-series constraints, described in Chapter 7, on both execution time and the number of backtracks (failures) for all the 200 time-series constraints for which the glue constraint, see Section 5.2.3, exists.

In our first experiment, we consider a single $\gamma(\langle X_1, X_2, \ldots, X_n \rangle, R)$ time-series constraint for which we first enumerate R and then either find a solution by assigning the X_i or prove infeasibility of the chosen R. For each constraint, we compare five variants: 1) the *Automaton* version just states the constraint, using the register automaton of [11]; 2) the *Glue* version adds to *Automaton* the glue constraints [8, 23] for all prefixes and corresponding reversed suffixes by just posting a single additional constraint γ' such that the equivalence $\gamma(\langle X_1, X_2, \ldots, X_n \rangle, R) \Leftrightarrow \gamma'(\langle X_n, X_{n-1}, \ldots, X_1 \rangle, R)$ is always true; 3) the *Bounds* version adds to *Automaton* the bound restrictions; 4) the *Bounds+Glue* version uses both the glue constraints and the bounds; and the *Combined* version adds to *Bounds+Glue* the bounds for each prefix and corresponding reversed suffix.

In Figure 13.1, we show results for two problems that are small enough to perform all computations for *Automaton* and all variants within a reasonable time. In the first problem (first row of plots), we use time series of length 10 over the domain $[1, 5]$, and find, for each value of R, the first solution or prove infeasibility. This would be typical for satisfaction or optimisation problems, where one has to detect infeasibility quickly. Our static search routine enumerates the time-series variables X_i from left to right, starting with the smallest value in the domain. In the case of the initial domains being of the same size, this heuristic typically works best. In the second problem (second row of plots), we consider time series of length 8 over the domain $[1, 5]$, and find all solutions for each value of R. This allows us to verify that no solutions are incorrectly eliminated by any of the variants, and provides a worst-case scenario exploring the complete search tree. Results for the backtrack count are on the left, results for the execution time on the right. We use log scales on both axes, replacing a zero value by one in order to allow plotting. All experiments were run with SICStus Prolog $4.2.3$ on a 2011 MacBook Pro 2.2 GHz quadcore Intel Core i7-950 machine with 6 MB cache and 16 GB memory using a single core.

We see that *Bounds* and *Glue* on their own bring good reductions of the search space, but their combinations *Bounds+Glue* and *Combined* in many cases reduce the number of backtracks by more than three orders of magnitude. Indeed, for many constraints, finding the first solution requires no backtracks. On the other hand, there are a few constraints for which the number of backtracks is not reduced significantly. These are constraints for which values of R in the middle of the domain are infeasible, but this is not detected by any of our variants.

The time for finding the first solution or proving infeasibility is also significantly reduced by the combinations *Bounds+Glue* and *Combined*, even though the glue constraints require posting two time-series constraints. When finding all solutions, this overhead shows in the total time taken for the three variants

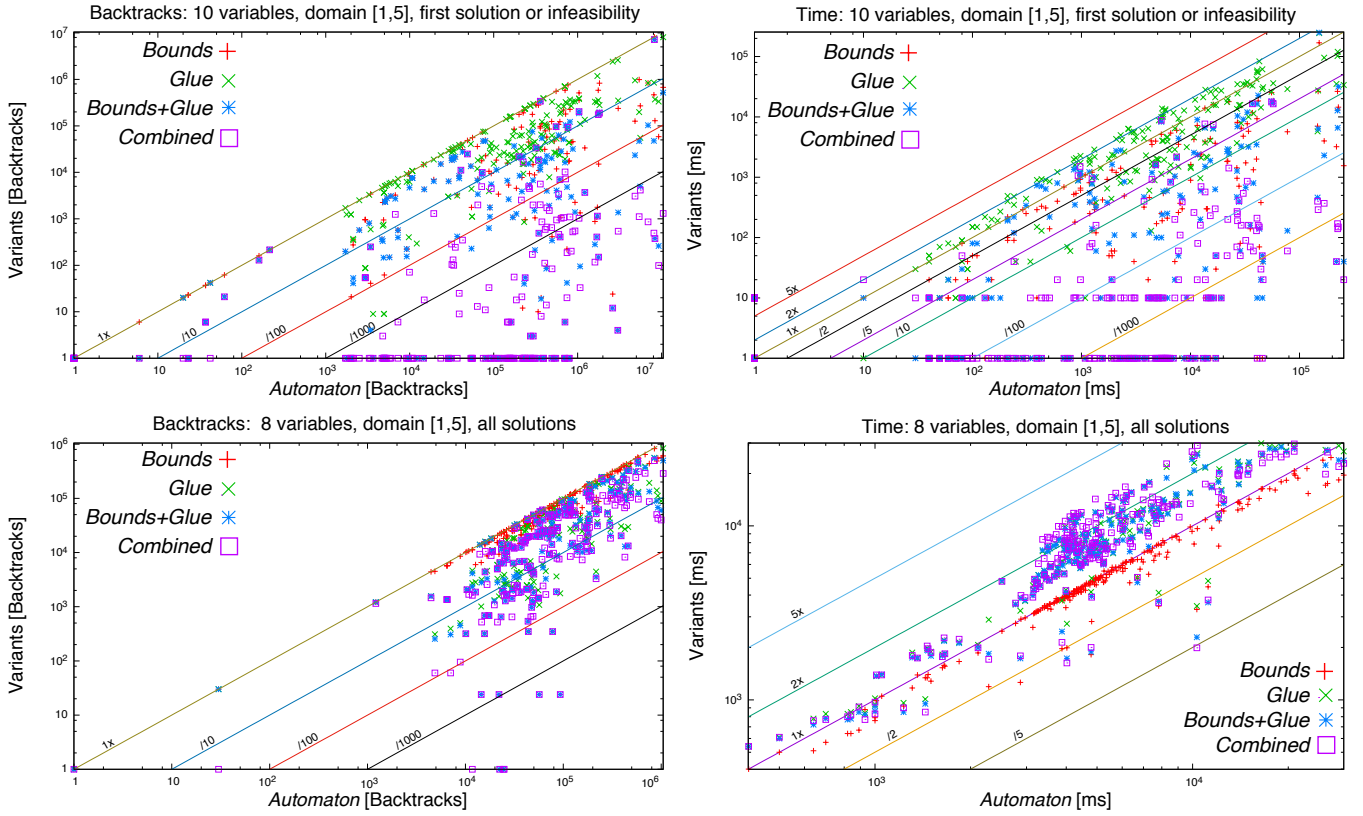


Figure 13.1 – Comparing backtrack count and runtime for *Automaton* and its variants for (left plots) the first solution or infeasibility for time series of length 10 and (right plots) all solutions for time series of length 8.

using the glue constraints. The bounds on their own reduce the time for many constraints, but rarely by more than a factor of ten.

In our second experiment, shown in Figure 13.2, we want to see whether the *Combined* variant is scalable. For this, we increase the length of the time series from 10 to 120 over the domain $[1, 5]$. We enumerate all possible values of R and find a first solution or prove infeasibility. For each time-series constraint and value of R, we impose a timeout of 20 seconds, and we do not consider the constraint if there is a timeout on some value of R. We plot the percentage of all constraints for which the average runtime is less than or equal to the value on the horizontal axis. For small time values, there are some quantisation effects due to the SICStus time resolution of 10 milliseconds.

For length 10, we find solutions for all values of R within the timeout, and our plots for *Automaton* (dashed) and *Combined* (solid) reach 100%, but the average time of *Combined* is much smaller. For *Automaton*, the percentage of constraints that are solved within the timeout drops to less than 20% for length 20, and less than 10% for length 40. For *Combined*, we solve over 75% of all constraints within the time limit, even for lengths 100 and 120.

The constraints that are not solved by *Combined* use the feature `surf` or the aggregator `sum`. The worst performance is observed for constraints combining both `surf` and `sum`. This is not surprising, as we know that achieving domain consistency for many of those constraints is NP-hard (encoding of *subset-sum*).

As a final experiment, we look at the search trees generated by four solution variants for a single constraint MAX_SURF_INCREASING_TERRACE. We only display some of the values for the parameter R, to make the trees more legible. Figure 13.3 shows the search tree produced with the help of CP-Viz [125]. Each tree shows the branches explored to find a first solution or proving infeasibility for each parameter value, with the initial choice of the value R at the top, and then the assignment of ten variables with a standard left-to-right labeling. Failed subtrees are abstracted as red triangles containing two numbers, the one above is the number of internal nodes in the tree, the one below the number of failed leaf nodes. Success nodes are colored in green, while failure nodes are colored red. Internal nodes are labeled by the variable

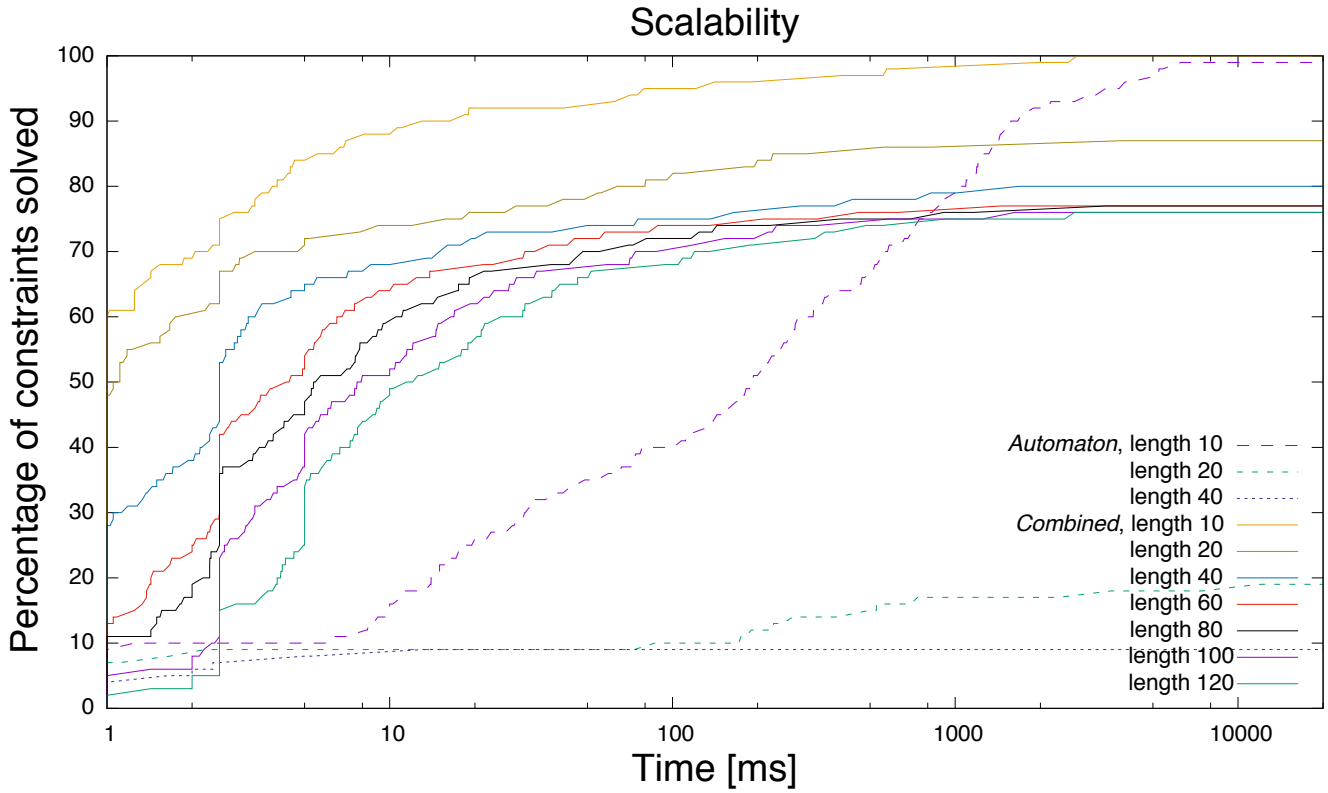
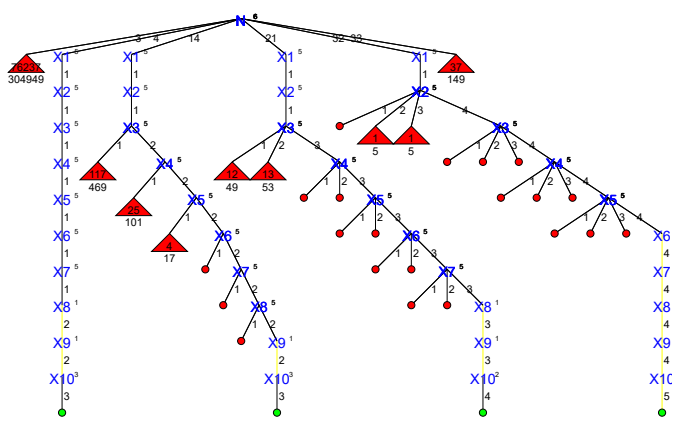


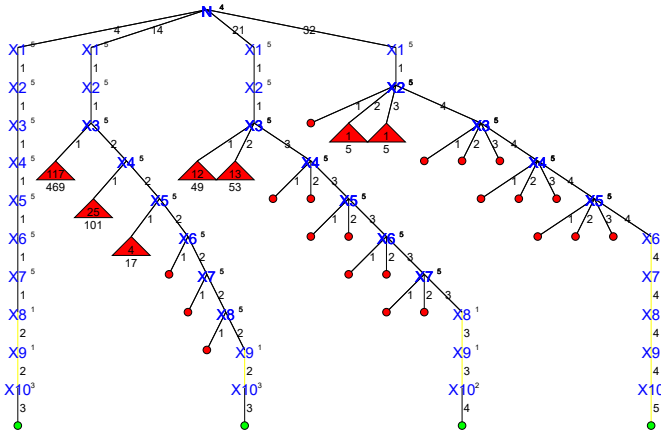
Figure 13.2 – Scalability results comparing time for *Automaton* and *Combined* on problems of increasing length.

name currently being assigned, and a superscript indicating the number of values in the domain of that variable. Edges indicate choices that are explored, the number indicates the value that is assigned to the selected variable, while a yellow edge color indicates that the value had been fixed by propagation.

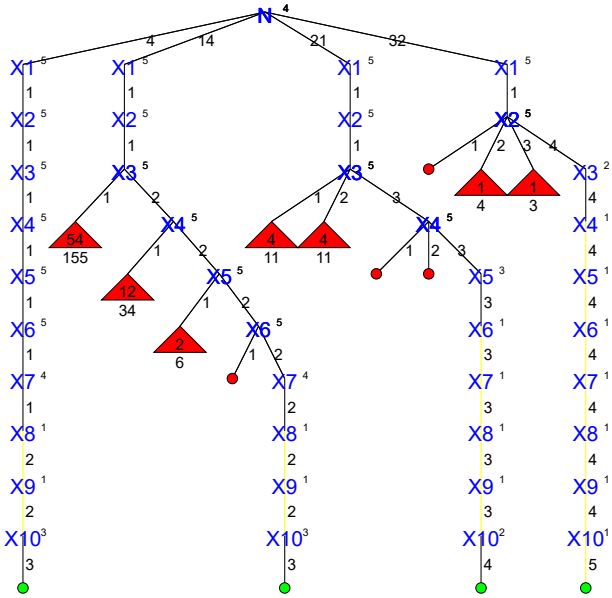
In all trees, a first solution for parameter value 4, the smallest feasible value, is found without backtracking. The solution chooses value 1 for X_1 to X_7, then value 2 for X_8 and X_9, and finally value 3 for variable X_{10}. On the other hand, in the initial register automaton, a very large failed subtree is shown for the left-most parameter value 3, and a much smaller failed tree for the right-most value 33. Both of those values are infeasible, and are removed by the bounds for this constraint. The *Bounds* version therefore avoids these failed sub-trees, but there are no changes for the other, feasible values. When we consider the *Bounds+Glue* version, the search for feasible solutions is reduced, with a further reduction for the *Combined* variant. But we still need search to find the initial solution for some of the parameter values. This occurs since the bounds and the glue constraint reasoning only consider lower and upper bounds, and we do not detect holes in the domain of variable R. To get the best use of the generated bounds, we have to use the incremental combination of *Bounds* with the *Glue* constraint, as the bounds are then applied for each suffix of unassigned variables to maximise the information extracted.



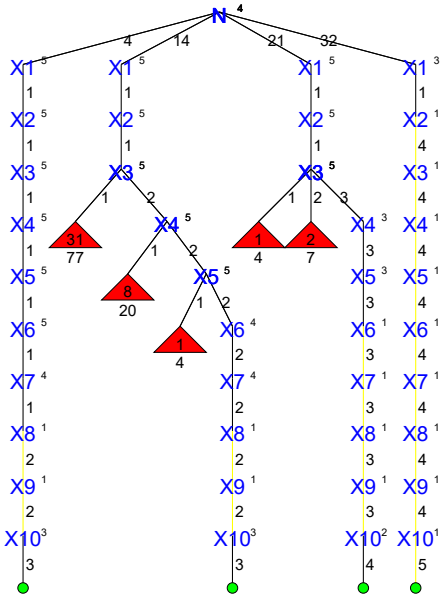
(A) *Automaton*



(B) *Bounds*



(C) *Bounds+Glue*



(D) *Combined*

Figure 13.3 – Comparing parts of the search tree for MAX_SURF_INCREASING_TERRACE, finding the first solution or proving infeasibility for the manually selected values 3, 4, 14, 21, 32, and 33 of variable R and 10 variables $X_1, X_2, \ldots, X_{10}$, each with domain $[1, 5]$. *Automaton* is the register automaton alone; *Bounds* adds to Automaton the bound restrictions; *Bounds+Glue* uses both the glue constraints and the bounds; and *Combined* adds to *Bounds+Glue* the bounds for each prefix and corresponding reversed suffix.

Chapter 14

Evaluation of the Impact of AMONG Implied Constraints

This chapter is adapted from an article published in the proceedings of the *CP'17* conference [12]. The final authenticated version of this article is available online at: `http://dx.doi.org/10.1007/978-3-319-66158-2_3`.

In this chapter, we evaluate the impact of AMONG implied constraints, presented in Chapter 8, on both execution time and the number of backtracks for time-series constraints of the MAX_SURF_σ and the SUM_SURF_σ families. The intended use case for such constraints is a problem where we learn parameters for a conjunction of time-series constraints from data, and use this conjunction to create new time series that are "similar" to the existing ones. An example would be electricity production data for a day [28], in half hour periods (48 values), or manpower levels per week over a year (52 values). To solve the conjunction, we need strong propagation for each individual constraint. We therefore evaluate the impact of the AMONG implied constraint on both execution time and the number of backtracks for the time-series constraints of the MAX_SURF_σ and the SUM_SURF_σ families for which a glue constraint [8] exists, which are 38 out of 44 time-series constraints of the two families. These families of constraints were the most difficult to solve in the experiments reported in [8] and in the previous chapter.

In the experiments for both families, we consider a single g_SURF_$\sigma(X, R)$ time-series constraint with g being either `sum` or `max`, for which we first systematically try out all potential values of the parameter R, and then either find a solution by assigning the X_i or prove infeasibility. We compare the best (Combined) approach from the previous chapter to the new method, adding the AMONG implied constraint on every suffix of $X = \langle X_1, X_2, \ldots, X_n \rangle$, and also a *preprocessing procedure*. The preprocessing procedure is a useful, if minor, contribution of the paper for 8 out of 38 of the constraints in the families studied. The purpose of this procedure is to find all feasible values of R, when σ is such that any σ-pattern has all values being the same. Such values of R must satisfy the following constraint:

$$R = \mathtt{id}_{g,f} \;\; \vee \;\; \left(\exists V \in [\ell', u'] \; \beta_{\sigma}^{\langle \ell, u, n \rangle} \cdot V \geq R \;\; \wedge \;\; R \bmod V = 0\right),$$

where ℓ' and u' are the smallest and the largest value, respectively, that can occur in a σ-pattern over $[\ell, u]$.

Since the implied constraints are precomputed offline, posting one AMONG implied constraint takes a *constant time*, and the time and space complexity of the preprocessing procedure does not exceed the size of the domain of R, which is $O(n \cdot (u - \ell))$.

Figure 14.1 presents the results for the SUM_SURF_σ (upper plots) and the MAX_SURF_σ (lower plots) time-series constraints, where X is a time series of length 50 over the domain $[0, 5]$, when the goal is to find, for each value of R, the first solution or prove infeasibility. This corresponds to our main use case, where we want to construct time series with fixed R values. Our static search routine enumerates the time-series variables X_i from left to right, starting with the smallest value in the domain. Results for the backtrack count are on the left, results for the execution time on the right. We use log scales on both axes, replacing a zero value by one in order to allow plotting. A timeout of 60 seconds was imposed. We see that the AMONG

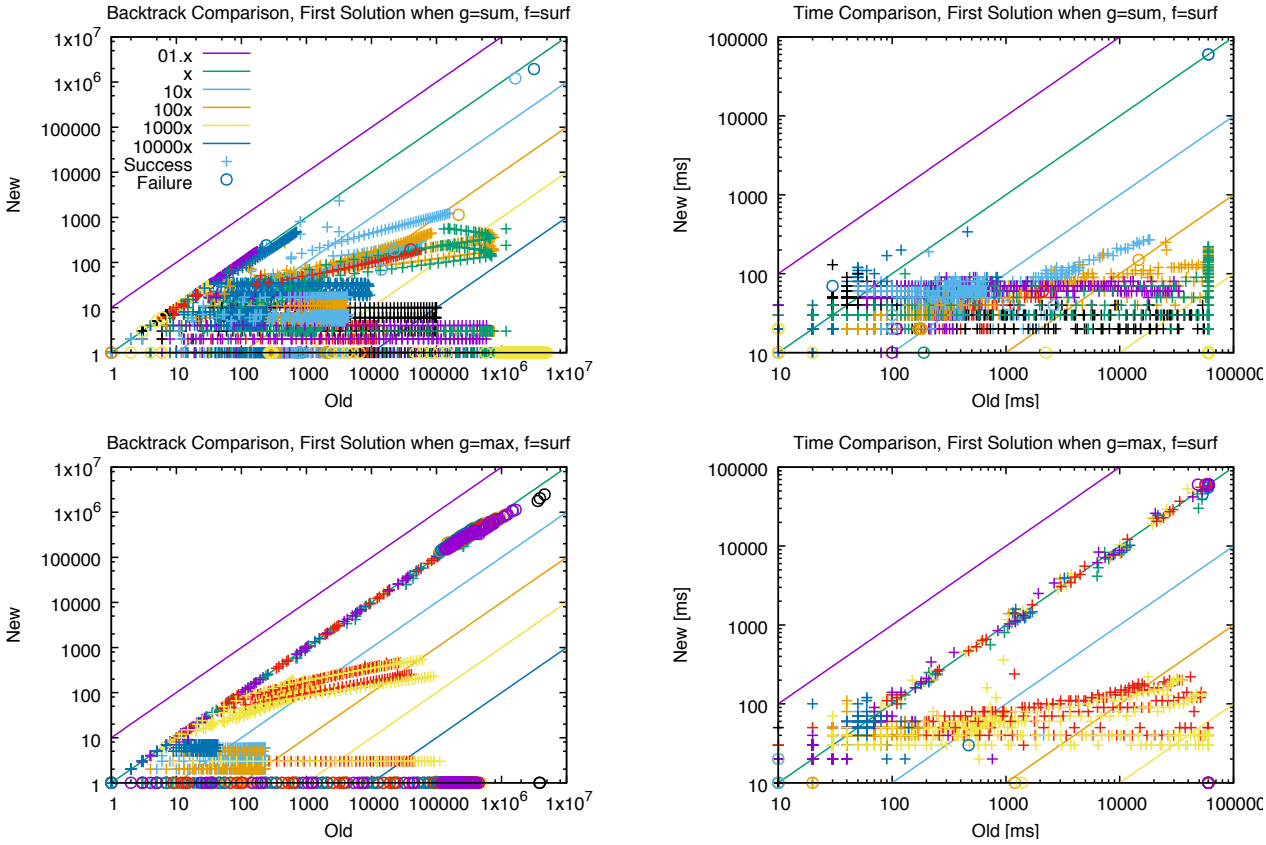


Figure 14.1 – Comparing backtrack count and runtime of the g_f_σ time-series constraint for previous best results (old) and new method for finding the first solution or proving infeasibility for time series of length 50 and domain $[0, 5]$. Colours of markers indicate the regular expression, the cross (respectively circle) marker type indicates success (respectively failure/timeout).

implied constraints reduce number of backtracks by up to a factor exceeding 10,000 and runtime by up to a factor of 1,000, and they divide the total execution time of terminated instances by a factor of 5 and 45 times when g is `max` and `sum`, respectively. All experiments were run on a 2014 iMac 4 GHz $i7$ using SICStus Prolog.

The results for the case $g = \texttt{sum}$ are better than for the case $g = \texttt{max}$ because the aggregator `sum` allows summing the surfaces of several σ-patterns, whereas for the `max` aggregator, R is the surface of a single σ-pattern, the surfaces of other σ-patterns, if any, are absorbed.

Chapter 15

Evaluation of the Impact of Linear Invariants

This chapter is adapted from an article published in the proceedings of the *CP'17* conference [13]. The final authenticated version of this article is available online at: `http://dx.doi.org/10.1007/978-3-319-66158-2_2`.

In this chapter, we evaluate the impact of linear invariants, generated by the method of Chapter 9.

To test the effectiveness of the generated invariants, we first try systematic tests on the conjunction of pairs of the 35 time-series constraints [22] of the NB_σ and SUM_WIDTH_σ families for which the glue constraints exist [8]. Recall that NB_σ constraints the number of σ-patterns in a time series, while SUM_WIDTH_σ constrains the sum of the widths of σ-patterns. Our intended use case is similar to [28], where constraints and parameter ranges of the problem are learned from real-world data, and are used to produce solutions that are similar to the previously observed data. It is important both to remove infeasible parameter combinations quickly, as well as helping to find solutions for feasible problems. Real world datasets often will only show a tiny subset of all possible parameter combinations, but as we do not know the data a priori, a systematic evaluation seems the most conservative approach.

For the experiments we use a database of generated invariants in a format compatible with the Global Constraint Catalogue [10]. Invariants are generated as Prolog facts, from which executable code, and other formats are then produced automatically. The time required to produce the invariants (5 min) is insignificant compared to the overall runtime of the experiments. For the 595 combinations of the 35 constraints we produce over 4100 linear invariants, over 3500 conditional linear invariants, and 86 guard invariants. In the test, we try each pair of constraints and try to find solutions for all possible pairs of parameter values.

We compare four different versions of our methods: The *pure* baseline version is the best approach (Combined) of Chapter 13. This version represents the state of the art for the considered families of time-series constraints before the current work. In the *invariant* version we add the generated invariants for the parameters of the complete time series. In the *incremental* version, we not only state the invariants for the complete time series, but also apply them for each suffix. The required variables are already available as part of the glue constraint setup, we only need to add the linear inequalities for each suffix length. In the *all* version, we add the intersection register automaton of the conjunction of the two constraints, if it contains guard invariants, and also state some additional, manually derived invariants.

The test program uses a labeling routine that first assigns the signature variables, and only afterwards assigns values for the X_i time-series variables. The variables in each case are assigned from left to right, i.e. the lexicographic order. For each pair of parameters values, defined by the product of the bounds from Chapter 7, we try to find a first solution with a timeout of 60s.

We have tested the results for different time-series length, Figure 15.1 shows the result for length 18 and domain size $0..18$, the largest problem size where we find solutions for each case within the timeout. All experiments were run on a laptop with Intel i7 CPU (2.9GHz), 64Gb main memory and Windows 10 64bit OS using SICStus Prolog 4.3.5 utilising a single core. For our four problem variants, we plot the percentage of undecided problem instances as a function of computation time. The plot uses log-log scales to more clearly show the values for short runtimes and for low number of undecided problems. The baseline *pure*

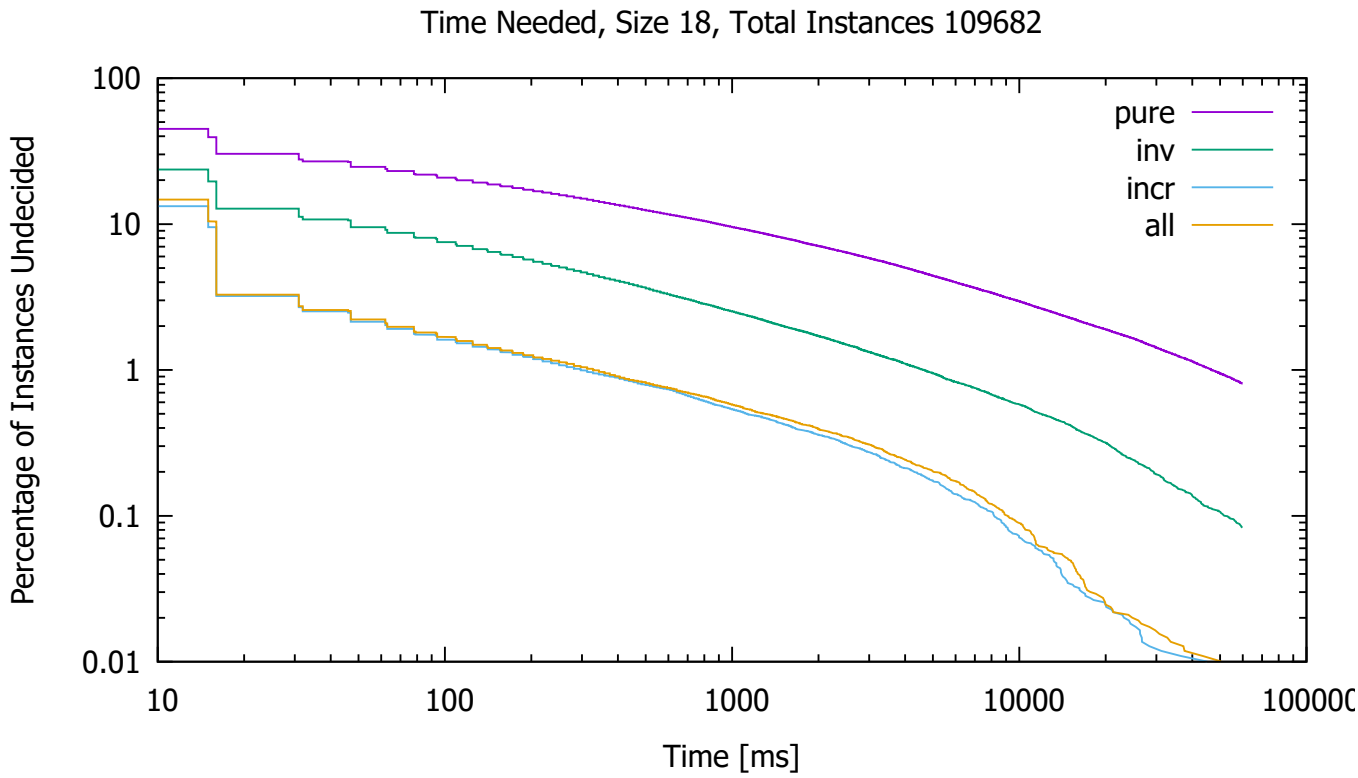
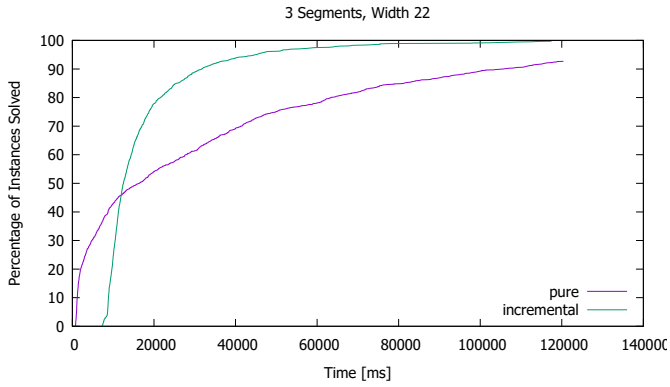


Figure 15.1 – Comparing constraint variants, undecided instances percentage for size 18 as a function of time, timeout is 60s.

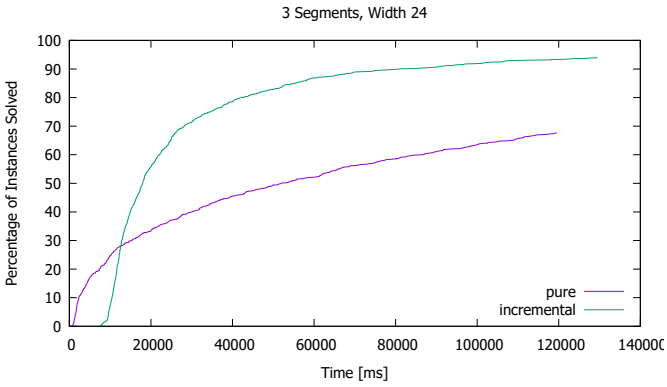
variant solves around 55% of the instances immediately, and leaves just under one percent unsolved within the timeout. The *invariants* version improves on this by pruning more infeasible problems immediately. On the other hand, stating the invariants on the full time series has no effect on feasible instances. When using the *incremental* version of the constraints, this has very little additional impact on infeasible problems, but improves the solution time for the feasible instances significantly. Adding (variant *all*) additional constraints further reduces the number of backtracks required, but these savings are largely balanced with the additional processing time, and therefore have no major impact on the overall results. After one second, around 9.5% of all instances are unsolved in the baseline, but only 0.5% in the *incremental* or *all* variant.

To test the method in a more realistic setting, we consider the conjunction of all 35 considered time-series constraints on electricity demand data provided by an industrial partner. The time series describes daily demand levels in half-hour intervals, giving 48 data points. 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 setup 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 shared signature and time-series 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 120s.

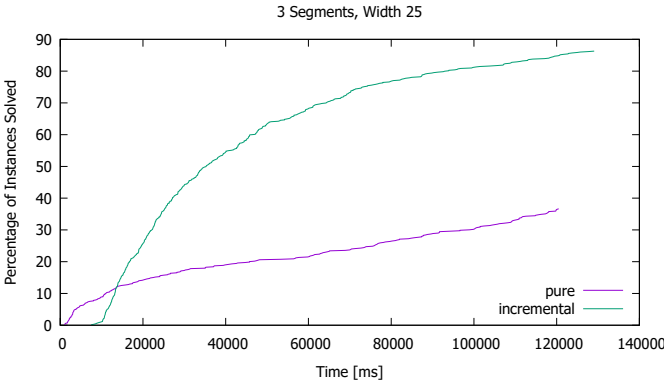
In order to understand the scaleability of the method, we also consider time series of 44 respectively 50 data points (three segments of length 22 and 25), extracted from the daily data stream covering a four year period (1448 samples). In Figure 15.2 we show the time and backtrack profiles for finding a first solution. 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



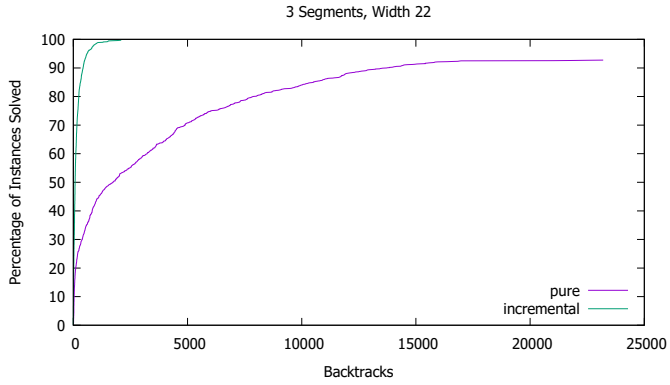
(A) Time, Size 22



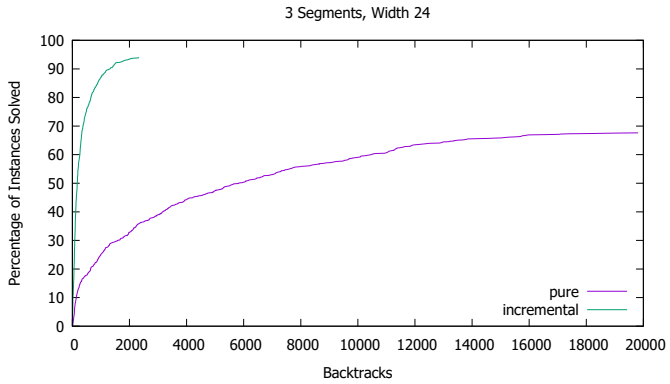
(B) Time, Size 24



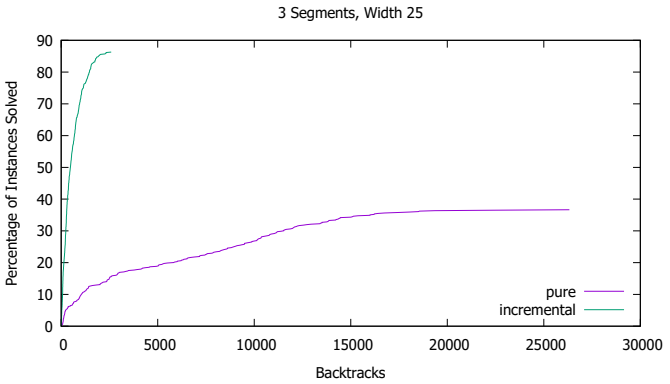
(C) Time, Size 25



(D) Backtracks, Size 22



(E) Backtracks, Size 24



(F) Backtracks, Size 25

Figure 15.2 – Percentage of problems solved for 3 overlapping segments of lengths 22, 24, and 25. Execution time in top row, backtracks required in bottom row.

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 time outs increases much more rapidly for the *pure* variant. Adding the invariants drastically reduces the search space in all cases, future work should consider if we can identify those invariants that actively contribute to the search by cutting off infeasible branches early on. Restricting the invariants to such an active subset should lead to a further improvement in execution time.

Chapter 16

Evaluation of the Impact of Non-Linear Invariants

In this chapter, we evaluate the impact of non-linear invariants, obtained by the method of Chapter 10.

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 $X = \langle X_1, X_2, \ldots, X_n \rangle$ with both γ_1 and γ_2 being in the union of the NB_σ and SUM_WIDTH_σ families of time-series constraints. After performing 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 length, it is unclear whether learning from small sequence length allows to also identify all infeasible combinations of R_1 and R_2 for larger sequence lengths. We investigate this question empirically by comparing the set of infeasible combinations of R_1 and R_2 learned by only using small sequences lengths (from 7 to 12) to the set of infeasible combinations of R_1 and R_2 generated by a systematic procedure for larger sequence lengths (from 13 to 24).
- 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 implementation of Chapter 15. 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 data base of invariants, i.e. to generate the hypotheses, to prove them, and to find the non-dominated set.

We use the generated invariants in our test data (lengths 13 to 24), by adding them to a baseline con-

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

Table 16.1 – Comparing the state-of-the-art baseline and the baseline with the generated invariants

sisting of the previous state-of-the-art implementation, i.e. Chapter 15, which uses the linear invariants of Chapter 9 and bounds of Chapter 7. Table 16.1 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.

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%.

Conclusion

17.1 Summary of this Thesis

Time-series constraints are constraints defined by means of functions in a compositional way. They provide a powerful modelling language, and have a number of potential real-life applications. The contributions of this thesis can be divided into two groups: 1) synthesising compositional combinatorial objects for time-series constraints and 2) extending transducer-based approach for representing constraints over integer sequences.

The purpose of the **synthesised combinatorial objects**, described in Chapters 7, 8, 9, 10, and 11 is to capture some aspect of a constraint or of a conjunction of constraints and to provide *functional scaleability* of the framework of time-series constraints. Namely our combinatorial objects allow us to reduce efforts required for adding a new constraint in the framework and for handling the combinatorial aspect of this new constraint. In addition, synthesised combinatorial objects can be used for different purposes including, but not limited to:

- as propagators in the context of CP;
- for obtaining a tight linear model in the context of MP;
- in the context of local search;
- in the context of data mining.

In this thesis we presented systematic methods for synthesising: 1) parameterised bounds on the result value of a time-series constraint, 2) parameterised AMONG implied constraints, 3) linear and 4) non-linear invariants linking the result values of several time-series constraints and parameterised by a function of the time-series length, and 5) conditional automata representing a condition on the result value of a time-series constraint. When synthesising combinatorial objects for a single time-series constraint, i.e. bounds, AMONG implied constraints, we used the declarative definition of time-series constraints, i.e. regular expression characteristics; and when synthesising objects for a conjunction of time-series constraints, i.e. linear and non-linear invariants, we used the operational view of time-series constraints, i.e. the seed transducers for each regular expression and register automata. Figure 17.1 summarises our contributions for synthesising compositional combinatorial objects for time-series constraints. In our benchmarks, we saw that the synthesised combinatorial objects have a significant impact on the propagation of time-series constraints.

We believe that ideas of our methods can be used not only for time-series constraints, but also for some other sequence constraints of the Global Constraint Catalogue [21].

The **extended transducer-based model**, introduced in Chapter 12, allows us to describe a number of existing sequence constraints using the same transducer-based model as for time-series constraints. In addition, the extended transducer-based model does not depend on the quantitative parameter b_σ, used for trimming the left extremity of occurrences of a regular expression, and can handle a larger class of regular expressions due to the new phase letter $\mathbf{maybe}_{\mathrm{r}}^{k}$ compared to the class of regular expressions of [68].

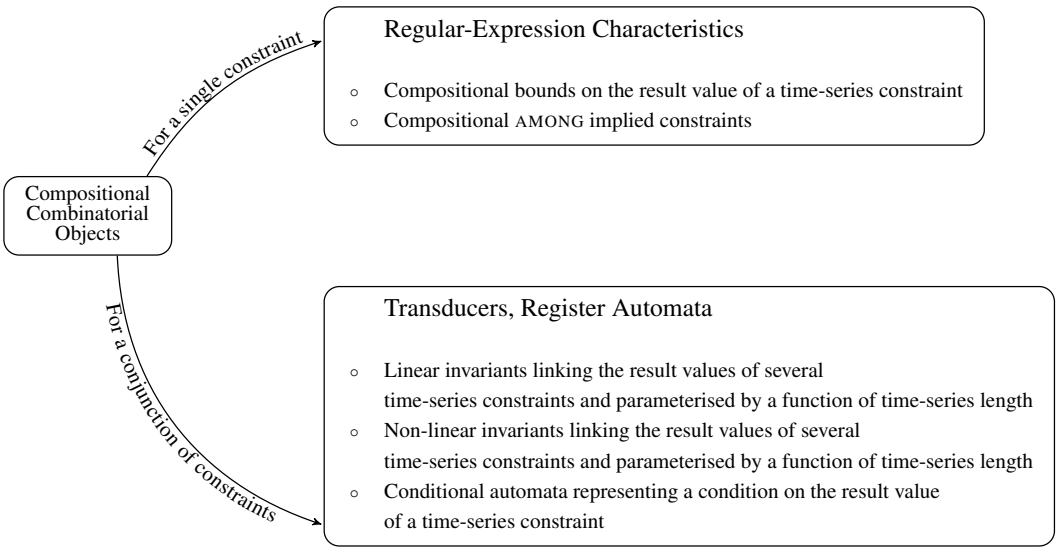


Figure 17.1 – Synthesised combinatorial objects, grouped by the case they are synthesised for, i.e. characterising a single constraint or a conjunction of constraints. The text on the top of each box provides to the key idea used for synthesising the corresponding combinatorial objects.

17.2 Future Work

Future work on time-series constraints can be done along several axes, presented in the following four sections. Each of the four sections gives directions for future work in one of the following four categories: improving the solving aspect, complexity analysis, formalisation and generalisation issues, and applications.

17.2.1 Improving the Solving Aspect

In this section, we discuss two directions for future work on improving the solving aspect of time-series constraints in both CP and MP contexts.

17.2.1.1 Non-Linear Guard Invariants for Time-Series Constraints

For some pairs of time-series constraints of the NB_$\sigma(X, R_1)$ and SUM_WIDTH_$\sigma(X, R_2)$ families, even after removing all infeasible combinations of R_1 and R_2, for some feasible combinations of R_1 and R_2, the solver still spends a lot of time searching for a feasible time series X. We have already done the work on synthesising linear guard invariants, but it is not enough and for some pairs of time-series constraints, there is a need for *non-linear guard invariants*, which could improve the propagation and fasten the search. Future work could look at *synthesising non-linear guard invariants* for time-series constraints where finding a feasible solution still takes a large amount of time.

17.2.1.2 Improving Linear Reformulation of Time-Series Constraints

Obtaining a tight linear representation of time-series constraints by making linear glue constraints is one of the axes of the future work. Linearising glue constraints [8] may improve solving for time-series constraints in the context of MP since we see that the synergy of bounds and glue constraints in the context of CP provides us with good improvement in propagation [8].

17.2.2 Complexity Analysis

In this section, we discuss two directions for future work related to the complexity analysis of time-series constraints and of computation of regular-expression characteristics.

17.2.2.1 Systematic Complexity Analysis of Time-Series Constraints

Although, we know that some time-series constraints of the SUM_SURF family are NP-complete, the complexity analysis was not done for other time-series constraints. Future work could look at a *systematic complexity analysis* of time-series constraints by finding out the reason for NP-completeness of some time-series constraints. Again the point would not be to analyse the complexity of each time-series constraint independently, but to come up with a compositional method parameterised by regular expressions, features, and aggregators to classify the complexity of the full set of a family of constraints.

17.2.2.2 Systematic Methods for Computing Characteristics of Regular Expressions

Regular-expression characteristics used in our bound formulae and AMONG implied constraints were computed manually. Most of the regular-expression characteristics minimise or maximise some quantity over the language of a regular expression, where we typically have to deal with an infinite set of words. One of the directions of future work could be developing systematic methods for computing regular-expression characteristics. Also, future work could analyse the complexity of the computation of regular-expression characteristics depending on the considered regular expression, and determine classes of regular expressions, for which characteristics can be computed in polynomial time, i.e. like graph classes for which computing some characteristics become polynomial [46].

17.2.3 Formalisation and Generalisation Issues

In this section, we discuss two directions of future work related to the formalisation of phase letters of the extended transducer-based model and to the generalisation of the reduced instruction set.

17.2.3.1 Formal-Logic Definition of Phase Letters in the Extended Transducer-Based Model

In the extended-transducer based model, described in Chapter 12, we only gave intuitions of phase letters. Future work could look at formal-logic definition of these letters. This would allow us to prove the well-formedness of seed transducers.

17.2.3.2 Generalising the Reduced Set of Instructions

In the extended-transducer based model, we presented a reduced instruction set, used for computing a function over integer sequences. However, this reduced instruction set does not suffice to compute the value of an arbitrary function over integer sequences. Future work could look at generalising the reduced instruction set so that it could handle a larger class of functions than now.

17.2.4 Applications

In this section, we discuss future work related to applications of time-series constraints.

17.2.4.1 Modelling and Solving an Industrial Problem with Time-Series Constraints

In [8], we have already modelled and solved a prototype of a staff scheduling at a call centre. Modelling and solving a real-life industrial problem using time-series constraints and synthesised combinatorial ob-

jects could 1) be the first industrial usage of time-series constraints and thus valorise them, and 2) highlight weak sides in the propagation of time-series constraints and thus inspire future work.

17.2.4.2 Feature Extraction and Time-Series Generation with Time-Series Constraints

Time series are common in many applications in different areas such as, for example, economics [101], astronomy [122], pattern recognition [74, 132], signal processing [112]. Time-series constraints could be used in these contexts for extracting symbolic features in time series, e.g. the number of peaks in a time series, and then generating time series with the same values of considered features, but optimising a certain quantity. Some work in this direction has been done in [94].

French Summary

Beaucoup de problèmes de la vie réelle où l'on doit planifier le personnel d'un centre d'appels ou planifier la production d'une centrale électrique peuvent être décrits et résolus comme des modèles mathématiques. Les deux composantes principales de tels modèles sont 1) des *variables* représentant les quantités que nous recherchons, par exemple la quantité d'électricité produite pour une centrale électrique donnée à un instant donné, pouvant prendre leurs valeurs dans des ensembles finis, et 2) des *contraintes*, imposant des relations entre ces variables et représentant des règles métiers, des restrictions techniques, etc. La *programmation mathématique* (PM) [126] et la *programmation par contraintes* (PPC) [118] sont deux approches complémentaires pour aborder de tels problèmes avec un certain nombre d'applications réussies dans les domaines de la planification, de l'emballage et du routage [134, 135, 48, 51, 110, 95].

La différence principale entre PPC et PM concerne le type de contraintes utilisées pour la modélisation. Dans le contexte de PM, les contraintes sont généralement linéaires ou convexes [16, 33, 115], alors que les modèles PPC utilisent souvent des *contraintes globales*. Le Global Constraint Catalogue [21] définit une contrainte globale comme une «condition expressive et concise impliquant un nombre non déterminé de variables». Par exemple, la contrainte ALLDIFFERENT$(\langle X_1, X_2, \ldots, X_n \rangle)$ [130] restreint une séquence de variables entières $\langle X_1, X_2, \ldots, X_n \rangle$ à prendre des valeurs distinctes. Par conséquent, la séquence $\langle 1, 8, 7, -1, 3 \rangle$ satisfait la contrainte ALLDIFFERENT, mais $\langle 1, 8, 1, -1, 3 \rangle$ ne la satisfait pas puisque X_1 a la même valeur que X_3. En PPC, une contrainte globale vient généralement avec un *propagateur*, c'est-à-dire un algorithme permettant de réduire les domaines des variables en supprimant les valeurs qui ne peuvent faire partie à aucune solution d'une contrainte.

Malgré différents types de contraintes, et donc différentes techniques de résolution, la PPC et la PM ont quelques inconvénients en commun motivant le travail de cette thèse :

- Aussi bien en PM qu'en PPC, la modélisation peut être difficile, à la fois d'un point de vue de la description du problème, et d'un point de vue d'inférence. En PM, cela est dû au fait que les contraintes doivent être linéaires ou convexes. En PPC, cela est dû au fait qu'une contrainte globale requise peut ne pas exister et doit donc être introduite. Ainsi, il y a un besoin commun de *définir les contraintes de manière compositionnelle*, ces contraintes pouvant ensuite être méthodiquement reformulées en programmes linéaires ou systématiquement encodées en termes de propagateurs.

- Lorsque les domaines des variables sont discrets, les modèles de PM et de PPC peuvent devenir difficiles à résoudre [106, 131]. Par conséquent, afin de résoudre un problème efficacement, on essaie de tirer parti de la structure du problème considéré. En PM, ceci est fait dans l'étape de prétraitement, où un solveur vérifie si un problème considéré a une structure bien connue, par exemple un problème du flot de coût minimum [63], puis soit applique une technique spécifique de prétraitement pour ce sous-problème ou soit génère des coupes. En PPC, ceci est fait en concevant des propagateurs spécialisés pour les contraintes globales du problème. D'où la nécessité de synthétiser des *objets combinatoires* capturant des facettes de la structure d'un problème considéré, par exemple, des bornes précises, des coupes linéaires ou des contraintes redondantes.

- La nécessité d'exploiter la structure du problème conduit à un grand nombre de méthodes dites *ad hoc*, par exemple des bornes, des algorithmes, des décompositions, des propagateurs, et des heuristiques toutes spécifiques. Ce sont des méthodes efficaces pour la résolution du problème pour lequel

elles ont été conçues, mais ne peuvent pas du tout être réutilisées pour tout autre problème, ou bien exigent un effort important d'adaptation. D'où la nécessité de développer des *méthodes systématiques* afin de synthétiser des objets combinatoires pour les contraintes d'un problème considéré.

Cette thèse étudie une famille de contraintes, nommées *contraintes sur les séries temporelles*. Ces contraintes sont définies d'une façon compositionnelle [22, 10]. Une contrainte sur les séries temporelles $\gamma(X, R)$ restreint la variable R, dite *valeur de résultat de* γ, à être le résultat des calculs faits à partir de la séquence des variables entières $X = \langle X_1, X_2, \ldots, X_n \rangle$, dite *série temporelle*, qui représente des mesures prises au fil du temps. Par exemple, R pourrait être le nombre de paires de variables consécutives $\langle X_i, X_{i+1} \rangle$ de X tel que $X_i < X_{i+1}$ avec i dans $[1, n-1]$. Les trois ingrédients principaux décrivant une contrainte sur les séries temporelles sont un *motif*, une *caractéristique*, et un *agrégateur*. Un motif est une forme régulière de sous-séquences, qui, est d'un point de vue formel, est caractérisée par une expression régulière sur l'alphabet de trois lettres $\{\text{`<'}, \text{`='}, \text{`>'}\}$. Par exemple, le motif DECREASING_SEQUENCE, qui correspond à toute sous-séquence monotone maximale décroissante $\langle X_i, X_{i+1}, \ldots, X_j \rangle$ d'une séquence des entiers $\langle X_1, X_2, \ldots, X_n \rangle$ est caractérisé par l'expression régulière '$(> (> \mid =)^*)^* >$', ce qui signifie que $X_i > X_{i+1} \ldots X_{j-1} > X_j$, et quelque soit k dans $[i+1, j-2]$, $X_k \geq X_{k+1}$. Une caractéristique et un agrégateur sont des fonctions sur des séquences entières, par exemple le maximum d'une séquence d'entiers, ou la somme des éléments d'une séquence des entiers.

Les séries temporelles sont très répandues en pratique. Nous donnons quelques exemples d'utilisations possibles des contraintes sur les séries temporelles :

- L'analyse de la production de centrales électriques sur plusieurs jours dans le contexte de la résolution du «Unit Commitment Problem»[28]. À partir des courbes de production connues des centrales électriques, on peut extraire un modèle en utilisant les contraintes sur les séries temporelles, puis générer une ou plusieurs courbes de production similaires satisfaisantes des restrictions supplémentaires pour la centrale considérée.
- Ordonnancement du personnel dans un centre d'appel [11]. Le problème consiste à couvrir la demande de la main-d'oeuvre donnée variant au fil du temps, tout en minimisant le coût global des ressources, et en satisfaisant les contraintes sur les séries temporelles données qui correspondent à des processus d'affaires, des règles d'emploi et des contrats syndicaux.
- La fouille de données dans le contexte de la gestion de l'alimentation pour systèmes distribués à grande échelle [26].
- L'analyse de trace pour le fournisseur d'Internet afin de tester la bande passante de la connexion de l'utilisateur [66].
- La prise de décision en temps réel, par exemple lorsqu'il faut analyser des flux de données afin d'ajuster certains paramètres, par exemple le taux de péage en fonction du trafic [5].

Pour de telles contraintes définies de manière compositionnelle, nous nous concentrons d'abord sur la construction de méthodes systématiques pour *synthétiser des objets combinatoires compositionnels* tels que des bornes précises, des invariants linéaires, des automates, etc., en exploitant leur nature compositionnelle au niveau combinatoire. Le mot «compositionnels»signifie ici que l'on peut non seulement combiner de tels objets pendant la phase de résolution, mais aussi les utiliser dans le cadre de techniques différentes, par exemple la PPC, la PM, ou la fouille de données.

Une formule capture certaines relations combinatoires entre des quantités différentes. L'idée mise en avance dans cette thèse est basée sur le pari qu'un ensemble de formules a potentiellement plus d'impact qu'un ensemble d'algorithmes sous réserve qu'il soit possible de les synthétiser. En effet, d'un point de vue compositionnel, les formules peuvent être utilisées conjointement et avec plusieurs techniques de résolution telles que la PPC ou la PM, ce qui s'avère beaucoup plus difficile dans le contexte des algorithmes. Un autre avantage des objets combinatoires est la *synergie* entre eux, c'est-à-dire que nous pouvons les composer. Des objets combinatoires différents combinés ensemble ont une meilleure performance que lorsqu'ils sont utilisés séparément. Un bon exemple d'une telle synergie est l'interaction entre des bornes précises sur la valeur de résultat d'une contrainte sur les séries temporelles γ et des *contraintes de colle* [8, 23]. Pour une séquence de variables $X = \langle X_1, X_2, \ldots, X_n \rangle$, un préfixe $P = \langle X_1, X_2, \ldots, X_i \rangle$ et un suffixe inversé

$S = \langle X_n, X_{n-1}, \ldots, X_i \rangle$ de X, une contrainte de colle relie les valeurs de résultat des trois contraintes sur les séries temporelles γ imposées sur X, sur P, et sur S.

Les objets combinatoires synthétisés peuvent être utilisés à différentes fins, y compris, mais pas limité à:

- Lors de la résolution d'un problème dans le contexte de la PPC, l'objectif est généralement d'élaguer le plus de valeurs irréalisables pour les variables, étant donné que plus petits sont les domaines, plus il est en principe facile de trouver une solution. Des objets combinatoires synthétisés peuvent être utilisés pour rendre le filtrage de contraintes sur les séries temporelles plus fort.
- Bien que les contraintes sur les séries temporelles puissent être reformulées en modèles linéaires [11] et intégrées dans des modèles linéaires existants, la reformulation linéaire obtenue n'est pas *précise*, c'est-à-dire qu'un solveur de programmation linéaire tel que CPLEX ou Gurobi passe généralement beaucoup de temps pour trouver une solution. Nos objets combinatoires peuvent être utilisés pour améliorer l'aspect résolution dans le contexte de la programmation linéaire.
- Les contraintes sur les séries temporelles peuvent être utilisées dans le contexte de l'*extraction de données*. Par exemple, des bornes précises sur la valeur de résultat d'une contrainte sur les séries temporelles sont utilisées pour regrouper des séries temporelles représentant la charge de travail d'un centre de données [94]; des bornes précises permettent de comparer les plages maximales de variation des valeur résultat de plusieurs contraintes sur les séries temporelles.

D'un point de vue opérationnel, toute contrainte sur les séries temporelles γ a une représentation en termes *d'automate à registres* synthétisé à partir du *transducteur* correspondant à l'expression régulière associée à γ [22]. Il a été montré dans [68] comment automatiquement générer un tel transducteur à partir d'une expression régulière. Tous les objets combinatoires que nous obtenons dans cette thèse seront soit synthétisés à partir de la vue déclarative des contraintes sur les séries temporelles, c'est-à-dire en utilisant des expressions régulières, soit à partir de leur représentation opérationnelle, c'est-à-dire en utilisant des automates à registres et des transducteurs. La Figure 17.2 donne la classification des objets combinatoires vus dans cette thèse en fonction de la représentation des contraintes sur les séries temporelles, à partir de laquelle ils ont été synthétisés, c'est-à-dire déclarative ou opérationnelle. Les objets combinatoires présentés dans la Figure 17.2 seront détaillés à la fin de ce résumé.

Bien que l'utilisation des transducteurs et des automates ait une longue tradition dans le contexte de la synthèse de composants logiciels fiables [133, 128], ils n'ont presque jamais été utilisés pour synthétiser

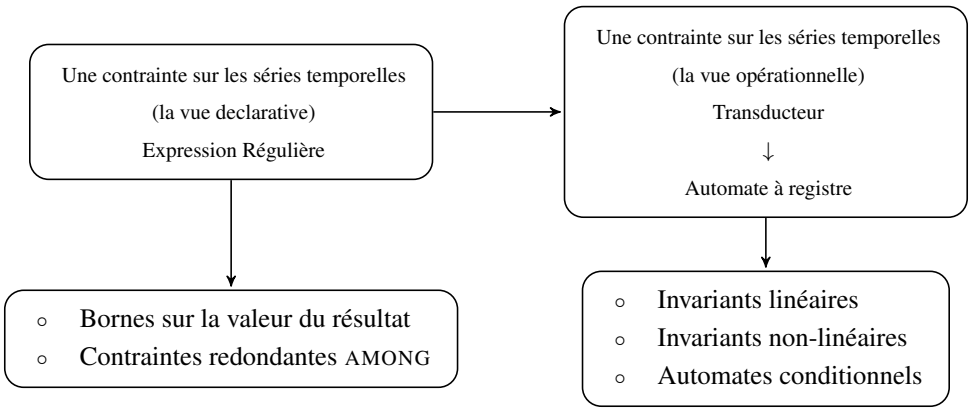


Figure 17.2 – Les objets combinatoires synthétisés et les facettes à partir desquelles ils étaient synthétisés, c'est-à-dire déclarative avec des expressions régulières ou opérationnelle avec des transducteurs et/ou des automates à registres. Une flèche de la source à la destination indique que la destination peut être synthétisée à partir de la source.

des objets combinatoires tels que des bornes précises, des coupes linéaires ou des contraintes de colle. Remarquons la correspondance suivante entre vérification assistée par ordinateur [55] et programmation par contraintes : premièrement, les deux utilisent parfois des spécifications déclaratives de haut niveau à partir desquelles des transducteurs et des automates sont synthétisés ; deuxièmement, il y a une correspondance entre les invariants qui sont généralement extraits de ces transducteurs et automates pour prouver une propriété d'un programme ou d'un système, et les conditions nécessaires que l'on souhaiterait synthétiser dans le contexte de la PPC ou la PM pour obtenir des inférences plus fortes: les deux sont des formules qui doivent toujours être vraies.

Le deuxième objectif de cette thèse est l'extension de *l'approche utilisée pour décrire les contraintes sur les séries temporelles* afin de capturer un grand nombre de contraintes sur les séquences telles que [25, 105, 108]. Le travail initial [22] utilise des transducteurs finis pour synthétiser des propagateurs pour les contraintes sur les séries temporelles. Cependant, le même modèle à base de transducteur peut être étendu pour synthétiser des propagateurs pour d'autres contraintes globales telles que AMONG [25], SIMILARITY [105], et STRETCH [108].

Avant de donner un aperçu de nos contributions, nous indiquons quatre raisons distinguant notre travail :

1. Premièrement, dans la littérature, il existe des approches se concentrant sur l'aspect combinatoire de contraintes spécifiques [116, 19, 39, 41] ou proposant des méthodes génériques pour décrire des contraintes [129] et synthétiser des propagateurs [100]. Les approches existantes ne gèrent pas l'aspect combinatoire d'une contrainte: ils reposent sur l'utilisateur pour décrire un propagateur par un ensemble de formules. Dans notre travail, nous allons un peu plus loin et explorons le sujet de *la synthèse automatique* de propagateurs sous la forme d'objets combinatoires pour une grande classe de contraintes sur les temporelles contraintes [22] contenant plus de 200 contraintes.
2. Deuxièmement, les objets combinatoires obtenus peuvent être utilisés, non seulement comme des propagateurs dans le contexte de la programmation par contraintes, mais aussi dans les contextes de la programmation linéaire, de l'exploration de données, ou de la recherche locale. Cela implique que ces objets représentent des informations essentielles sur l'aspect combinatoire d'une contrainte sur les séries temporelles, et sont donc indépendants du contexte dans lequel des contraintes sur les séries temporelles sont utilisées.
3. Troisièmement, les objets obtenus sont *paramétrés* par la description d'une contrainte sur les séries temporelles considérées, la longueur d'une série temporelle et les domaines des variables de la série temporelle, et sont synthétisés *une bonne fois pour toutes*. Cela nous permet de créer une *base de données* d'objets combinatoires pour les contraintes sur les séries temporelles [10] consultable dans des contextes complètement différents. Il n'est pas nécessaire de relancer nos méthodes de synthèse de ces objets combinatoires pour chaque instance de problème. Notons que, pour obtenir de tels objets combinatoires, nous devons *prouver automatiquement* qu'ils sont valables pour toute longueur de séquence.
4. Quatrièmement, rechercher des moyens uniformes pour représenter des familles de contraintes globales et gérer leur aspect combinatoire n'est pas habituel au sein de la communauté PPC, mais c'est malgré tout important, car nous finirions sinon avec un ensemble de contraintes dédiées à chaque problème ne communiquant pas entre elles.

Une visite guidée à travers les contributions principales de cette thèse.
Les contributions principales présentées dans cette thèse sont les suivantes :

- [**Les bornes supérieures et inférieures compositionnelles** sur la valeur du résultat de chaque contrainte sur les séries temporelles]
  Une formule de borne pour une contrainte sur les séries temporelles est paramétrée par la longueur n de la série temporelle et par les domaines des variables de la série temporelle. Chaque formule

de borne est obtenue à partir d'une formule générique, qui est paramétrée par une contrainte sur les séries temporelles considérées. Par conséquent, nous avons seulement besoin de prouver quelques formules génériques plutôt qu'une formule pour chaque contrainte sur les séries temporelles. Bien que la borne soit toujours valide, sa finesse n'est garantie que lorsque les domaines de toutes les variables de la série temporelle correspondent au même intervalle entier. Pour presque toutes les contraintes sur les séries temporelles les bornes supérieures et inférieures sont évaluées en temps constant, à l'exception de 12 contraintes pour lesquelles l'évaluation se fait en $O(n)$.
Ce travail a été publié dans le journal *Constraints* [14] et dans les actes de la conférence *CP'16* [8] ; les bornes pour toutes les contraintes sur les séries temporelles ont été intégrées dans le Volume II du Global Constraint Catalogue [10].

- [**Contraintes implicites AMONG** pour trois familles des contraintes sur les séries temporelles]
  Une contrainte AMONG [25] limite le nombre de variables d'une séquence de variables, pouvant prendre leurs valeurs dans un ensemble fini particulier de valeurs entières. Ici, le mot *implicite* signifie que ces contraintes sont redondantes, c'est-à-dire qu'elles ne changent pas l'ensemble des solutions du problème. Leur but est d'enlever des valeurs irréalisables des domaines des variables. Comme pour les bornes, il y a une seule contrainte AMONG générique par famille qui est paramétrée par le motif d'une contrainte sur les séries temporelles considérée. Par conséquent, nous avons seulement besoin de prouver trois contraintes AMONG implicites pour les utiliser avec 66 contraintes sur les séries temporelles.
  Ce travail a été publié dans les actes de la conférence *CP'17* [12], et les contraintes implicites AMONG pour 66 contraintes sur les séries temporelles ont été intégrées dans le Volume II du Global Constraint Catalogue [10].

- [**Inégalités implicites linéaires** reliant les valeurs de résultat d'une conjonction des contraintes sur les séries temporelles imposées sur la même séquence de longueur n, et paramétrés par n]
  Nous explorons les relations entre les valeurs du résultat de *plusieurs* contraintes sur les séries temporelles imposées sur une même séquence. On nomme ces inégalités *invariants linéaires*.
  Ce travail a été publié dans les actes de la conférence *CP'17* [13], et les invariants linéaires obtenus ont été intégrés dans le Volume II du Global Constraint Catalogue [10].

- [**Invariants non linéaires** reliant les valeurs de résultat d'une conjonction des contraintes sur les séries temporelles imposées sur une même séquence de longueur n, et paramétrés par une fonction de n]
  De tels invariants caractérisent des ensembles de combinaisons irréalisables des valeurs du résultat des contraintes sur les séries temporelles dans une conjonction telles que l'on ne peut pas les exprimer comme une combinaison linéaire de R_1, R_2 et n. En d'autres termes, ce sont des ensembles de combinaisons irréalisables qui sont situés dans l'enveloppe convexe de combinaisons réalisables. Ces invariants non linéaires ont été intégrés dans le Volume II du Global Constraint Catalogue [10].

- [**Automates de taille constante** représentant l'ensemble de toutes les séquences entières satisfaisantes une condition, par exemple toutes les séquences entières avec le nombre maximum de séquences décroissantes pour une longueur de séquence donnée]
  D'une part, les automates finis sont utilisés depuis le début de l'informatique pour modéliser de nombreux aspects du calcul [81]. D'autre part, les bornes sont omniprésentes dans un certain nombre de problèmes d'optimisation [88, 18] où elles permettent d'accélérer le processus de recherche. Alors que les bornes sont généralement exprimées par des formules paramétrées [30, 14], la question d'une représentation compacte et explicite de l'ensemble de *toutes les solutions* atteignant une borne particulière est passée inaperçue. Ces automates sont une partie cruciale de notre méthode pour synthétiser et prouver des contraintes implicites non linéaires, mentionnées dans le point précédent. Les automates obtenus ont été intégrés dans le Volume II du Global Constraint Catalogue [10].

Appendices

Appendix A

An Entry of the Global Constraint Catalogue

A.1 Metadata

The following synthesised code corresponds to the metadata of the Global Constraint Catalogue for the NB_PEAK time-series constraint. The bounds for time-series constraints, presented in Chapter 7, were integrated into the `ctr_restrictions` predicate, and the conditional automata, presented in Chapter 11, were integrated into the `ctr_specialisation` predicates.

```
:- multifile
    ctr_predefined/1,
    ctr_date/2,
    ctr_persons/2,
    ctr_origin/3,
    ctr_usual_name/2,
    ctr_synonyms/2,
    ctr_types/2,
    ctr_arguments/2,
    ctr_exchangeable/2,
    ctr_restrictions/2,
    ctr_typical/2,
    ctr_typical_model/2,
    ctr_pure_functional_dependency/2,
    ctr_functional_dependency/3,
    ctr_contractible/4,
    ctr_extensible/4,
    ctr_aggregate/3,
    ctr_example/2,
    ctr_draw_example/9,
    ctr_cond_imply/5,
    ctr_see_also/2,
    ctr_key_words/2,
    ctr_derived_collections/2,
    ctr_graph/7,
    ctr_graph/9,
    ctr_eval/2,
    ctr_automaton_signature/3,
    ctr_glue_matrix/2,
    ctr_specialisation/3,
    ctr_sol/6,
```

```
    ctr_logic/3,
    ctr_application/2.

ctr_date(nb_peak,['20141203']).

ctr_origin(nb_peak,
           'Based on the \\hyperlink{Ppeak}{\\pattern{peak}} pattern.',
           []).

ctr_arguments(nb_peak,
              ['VALUE'-dvar,
               'VARIABLES'-collection(var-dvar)]).

ctr_exchangeable(nb_peak,
                 [items('VARIABLES',reverse),
                  translate(['VARIABLES'^var])]).

ctr_restrictions(nb_peak, let([sv=size('VARIABLES'),
                               rv=range('VARIABLES'^var)],
                              [sv=<2#\/rv=<1#=>'VALUE'=0,
                               'VALUE'>=0,
                               'VALUE'=<markup(max(0,(sv-1)/2),1),
                               required('VARIABLES',var)])).

ctr_pure_functional_dependency(nb_peak, []).

ctr_functional_dependency(nb_peak, 1, [2]).

ctr_typical(nb_peak,[size('VARIABLES')>2,range('VARIABLES'^var)>1]).


ctr_example(nb_peak,
            nb_peak(3,
                    [[var-7],
                     [var-5],
                     [var-5],
                     [var-1],
                     [var-4],
                     [var-5],
                     [var-2],
                     [var-2],
                     [var-3],
                     [var-5],
                     [var-6],
                     [var-2],
                     [var-3],
                     [var-3],
                     [var-3],
                     [var-1]])).


ctr_key_words(nb_peak,[]).
```

```
ctr_eval(nb_peak, [checker(nb_peak_c1),
    checker(nb_peak_c),
    automaton(nb_peak_a1),
    automaton(nb_peak_a),
    automaton_with_signature(nb_peak_a1_s),
    automaton_with_signature(nb_peak_a_s)]).

% these are currently not used in time-series
ctr_cond_imply(-,-,-,-,-):-fail.
ctr_contractible(-,-,-,-):-fail.
ctr_extensible(-,-,-,-):-fail.
ctr_aggregate(-,-,-):-fail.
ctr_sol(-,-,-,-,-,-):-fail.

nb_peak_c(Value, VARIABLES) :-
        collection(VARIABLES, [int]),
        get_attr1(VARIABLES, VARS),
        VARS=[First|_],
        length(VARS, N),
        N1 is N+1,
        Default=0,
        nb_peak_c(VARS, Default, s, Default, 0, Default, Value).

nb_peak_c([_], Default, _, CLast, DLast, RLast, Value) :- !,
        Value is RLast+CLast.
nb_peak_c([Xi,Xj|Xs], Default, s, C, D, R, Result) :-
        Xi>=Xj, !,
        nb_peak_c([Xj|Xs], Default, s, C, D, R, Result).
nb_peak_c([Xi,Xj|Xs], Default, s, C, D, R, Result) :-
        Xi<Xj, !,
        nb_peak_c([Xj|Xs], Default, r, C, D, R, Result).
nb_peak_c([Xi,Xj|Xs], Default, r, C, D, R, Result) :-
        Xi>Xj, !,
        A1000 is max(D,1),
        nb_peak_c([Xj|Xs], Default, t, A1000, 0, R, Result).
nb_peak_c([Xi,Xj|Xs], Default, r, C, D, R, Result) :-
        Xi=<Xj, !,
        A1000 is max(D,1),
        nb_peak_c([Xj|Xs], Default, r, C, A1000, R, Result).
nb_peak_c([Xi,Xj|Xs], Default, t, C, D, R, Result) :-
        Xi>Xj, !,
        A1000 is max(C,max(D,1)),
        nb_peak_c([Xj|Xs], Default, t, A1000, 0, R, Result).
nb_peak_c([Xi,Xj|Xs], Default, t, C, D, R, Result) :-
        Xi=:=Xj, !,
        A1000 is max(D,1),
        nb_peak_c([Xj|Xs], Default, t, C, A1000, R, Result).
nb_peak_c([Xi,Xj|Xs], Default, t, C, D, R, Result) :-
        Xi<Xj, !,
        A1000 is R+C,
        nb_peak_c([Xj|Xs], Default, r, Default, 0, A1000, Result).

nb_peak_c1(Value, VARIABLES) :-
```

```
        collection(VARIABLES, [int]),
        get_attr1(VARIABLES, VARS),
        VARS=[First|_],
        length(VARS, N),
        N1 is N+1,
        Default=0,
        nb_peak_c1(VARS, Default, s, Default, Value).

nb_peak_c1([_], Default, _, RLast, Value) :- !,
        Value is RLast.
nb_peak_c1([Xi,Xj|Xs], Default, s, R, Result) :-
        Xi>=Xj, !,
        nb_peak_c1([Xj|Xs], Default, s, R, Result).
nb_peak_c1([Xi,Xj|Xs], Default, s, R, Result) :-
        Xi<Xj, !,
        nb_peak_c1([Xj|Xs], Default, r, R, Result).
nb_peak_c1([Xi,Xj|Xs], Default, r, R, Result) :-
        Xi>Xj, !,
        A1000 is R+1,
        nb_peak_c1([Xj|Xs], Default, t, A1000, Result).
nb_peak_c1([Xi,Xj|Xs], Default, r, R, Result) :-
        Xi=<Xj, !,
        nb_peak_c1([Xj|Xs], Default, r, R, Result).
nb_peak_c1([Xi,Xj|Xs], Default, t, R, Result) :-
        Xi>Xj, !,
        nb_peak_c1([Xj|Xs], Default, t, R, Result).
nb_peak_c1([Xi,Xj|Xs], Default, t, R, Result) :-
        Xi=:=Xj, !,
        nb_peak_c1([Xj|Xs], Default, t, R, Result).
nb_peak_c1([Xi,Xj|Xs], Default, t, R, Result) :-
        Xi<Xj, !,
        nb_peak_c1([Xj|Xs], Default, r, R, Result).

ctr_automaton_signature(nb_peak, nb_peak_a, pair_signature(2,signature)).

ctr_automaton_signature(nb_peak, nb_peak_a1, pair_signature(2,signature)).

nb_peak_a(Flag,Value,VARIABLES):-
        Default = 0,
        nb_peak_a(Flag,Value,VARIABLES,Default).

nb_peak_a(Flag,Value,VARIABLES,Default):-
        pair_signature(VARIABLES, Signature),
        nb_peak_a_s(Flag,Value,VARIABLES,Default,Signature).

nb_peak_a_s(Flag,Value,VARIABLES,Default,Signature):-
        collection(VARIABLES, [dvar]),
        get_attr1(VARIABLES, Xs),
        length(Xs, N),
        gen_pairs(Xs, XPairs),
        Xs = [First|_],
        LT = 0, EQ = 1, GT = 2,
        automaton(XPairs, Xi-Xj, Signature,
```

```
        [source(s),
        sink(s),
        sink(r),
        sink(t)],
        [arc(s,GT,s,([C, D, R])),
        arc(s,EQ,s,([C, D, R])),
        arc(s,LT,r,([C, D, R])),
        arc(r,GT,t,([max(D, 1), 0, R])),
        arc(r,LT,r,([C, max(D, 1), R])),
        arc(r,EQ,r,([C, max(D, 1), R])),
        arc(t,GT,t,([max(C, max(D, 1)), 0, R])),
        arc(t,EQ,t,([C, max(D, 1), R])),
        arc(t,LT,r,([Default, 0, R + C]))],
        [C,D,R], [Default,0,Default], [CLast,DLast,RLast]),
        Value #= RLast + CLast #<=> Flag.

nb_peak_a1(Flag,Value,VARIABLES):-
        Default = 0,
        nb_peak_a1(Flag,Value,VARIABLES,Default).

nb_peak_a1(Flag,Value,VARIABLES,Default):-
        pair_signature(VARIABLES, Signature),
        nb_peak_a1_s(Flag,Value,VARIABLES,Default,Signature).

nb_peak_a1_s(Flag,Value,VARIABLES,Default,Signature):-
        collection(VARIABLES, [dvar]),
        get_attr1(VARIABLES, Xs),
        length(Xs, N),
        gen_pairs(Xs, XPairs),
        Xs = [First|_],
        LT = 0, EQ = 1, GT = 2,
        automaton(XPairs, Xi-Xj, Signature,
        [source(s),
        sink(s),
        sink(r),
        sink(t)],
        [arc(s,GT,s,([R])),
        arc(s,EQ,s,([R])),
        arc(s,LT,r,([R])),
        arc(r,GT,t,([R + 1])),
        arc(r,LT,r,([R])),
        arc(r,EQ,r,([R])),
        arc(t,GT,t,([R])),
        arc(t,EQ,t,([R])),
        arc(t,LT,r,([R]))],
        [R], [Default], [RLast]),
        Value #= RLast #<=> Flag.

nb_peak_r(VALUE,Xs,SV):-
        nb_peak_r(VALUE,Xs,SV,SV).

nb_peak_r(VALUE,Xs,SV,ST):-
        range_int(Xs,RV),
```

```
        SV#=<2#=>VALUE#=0,
        RV#=<1#=>VALUE#=0,
        VALUE#>=0,
        VALUE#=<max(0,(SV-1)/2).


ctr_glue_matrix(nb_peak,
                [cell(s,s,cf+cb),
                 cell(s,r,cf+cb),
                 cell(s,t,cf+cb),
                 cell(r,s,cf+cb),
                 cell(r,r,1),
                 cell(r,t,1),
                 cell(t,s,cf+cb),
                 cell(t,r,1),
                 cell(t,t,cf+cb)]).

ctr_specialisation(nb_peak,
                   nb_peak_eq_0,
                   kernel([source(s),sink(s),sink(r)],
                          [arc(s,0,r,(true->[])),
                           arc(s,1,s,(true->[])),
                           arc(s,2,s,(true->[])),
                           arc(r,0,r,(true->[])),
                           arc(r,1,r,(true->[]))],
                          [],[],[])).

ctr_specialisation(nb_peak,
                   nb_peak_eq_1,
                   kernel([source(s(1)),sink(s(3)),sink(s(4))],
                          [arc(s(1),0,s(2),(true->[])),
                           arc(s(1),1,s(1),(true->[])),
                           arc(s(1),2,s(1),(true->[])),
                           arc(s(2),0,s(2),(true->[])),
                           arc(s(2),1,s(2),(true->[])),
                           arc(s(2),2,s(4),(true->[])),
                           arc(s(3),0,s(3),(true->[])),
                           arc(s(3),1,s(3),(true->[])),
                           arc(s(4),0,s(3),(true->[])),
                           arc(s(4),1,s(4),(true->[])),
                           arc(s(4),2,s(4),(true->[]))],
                          [],[],[])).

ctr_specialisation(nb_peak,
                   nb_peak_eq_2,
                   kernel([source(s(1)),sink(s(5)),sink(s(6))],
                          [arc(s(1),0,s(2),(true->[])),
                           arc(s(1),1,s(1),(true->[])),
                           arc(s(1),2,s(1),(true->[])),
                           arc(s(2),0,s(2),(true->[])),
                           arc(s(2),1,s(2),(true->[])),
                           arc(s(2),2,s(3),(true->[])),
                           arc(s(3),0,s(4),(true->[])),
```

```
                        arc(s(3),1,s(3),(true->[])),
                        arc(s(3),2,s(3),(true->[])),
                        arc(s(4),0,s(4),(true->[])),
                        arc(s(4),1,s(4),(true->[])),
                        arc(s(4),2,s(6),(true->[])),
                        arc(s(5),0,s(5),(true->[])),
                        arc(s(5),1,s(5),(true->[])),
                        arc(s(6),0,s(5),(true->[])),
                        arc(s(6),1,s(6),(true->[])),
                        arc(s(6),2,s(6),(true->[]))],
                       [],[],[])).

ctr_specialisation(nb_peak,
                   nb_peak_eq_3,
                   kernel([source(s(1)),sink(s(7)),sink(s(8))],
                          [arc(s(1),0,s(2),(true->[])),
                           arc(s(1),1,s(1),(true->[])),
                           arc(s(1),2,s(1),(true->[])),
                           arc(s(2),0,s(2),(true->[])),
                           arc(s(2),1,s(2),(true->[])),
                           arc(s(2),2,s(3),(true->[])),
                           arc(s(3),0,s(4),(true->[])),
                           arc(s(3),1,s(3),(true->[])),
                           arc(s(3),2,s(3),(true->[])),
                           arc(s(4),0,s(4),(true->[])),
                           arc(s(4),1,s(4),(true->[])),
                           arc(s(4),2,s(5),(true->[])),
                           arc(s(5),0,s(6),(true->[])),
                           arc(s(5),1,s(5),(true->[])),
                           arc(s(5),2,s(5),(true->[])),
                           arc(s(6),0,s(6),(true->[])),
                           arc(s(6),1,s(6),(true->[])),
                           arc(s(6),2,s(8),(true->[])),
                           arc(s(7),0,s(7),(true->[])),
                           arc(s(7),1,s(7),(true->[])),
                           arc(s(8),0,s(7),(true->[])),
                           arc(s(8),1,s(8),(true->[])),
                           arc(s(8),2,s(8),(true->[]))],
                          [],[],[])).


ctr_specialisation(nb_peak,
                   nb_peak_eq_4,
                   kernel([source(s(1)),sink(s(9)),sink(s(10))],
                          [arc(s(1),0,s(2),(true->[])),
                           arc(s(1),1,s(1),(true->[])),
                           arc(s(1),2,s(1),(true->[])),
                           arc(s(2),0,s(2),(true->[])),
                           arc(s(2),1,s(2),(true->[])),
                           arc(s(2),2,s(3),(true->[])),
                           arc(s(3),0,s(4),(true->[])),
                           arc(s(3),1,s(3),(true->[])),
                           arc(s(3),2,s(3),(true->[])),
```

```
                arc(s(4),0,s(4),(true->[])),
                arc(s(4),1,s(4),(true->[])),
                arc(s(4),2,s(5),(true->[])),
                arc(s(5),0,s(6),(true->[])),
                arc(s(5),1,s(5),(true->[])),
                arc(s(5),2,s(5),(true->[])),
                arc(s(6),0,s(6),(true->[])),
                arc(s(6),1,s(6),(true->[])),
                arc(s(6),2,s(7),(true->[])),
                arc(s(7),0,s(8),(true->[])),
                arc(s(7),1,s(7),(true->[])),
                arc(s(7),2,s(7),(true->[])),
                arc(s(8),0,s(8),(true->[])),
                arc(s(8),1,s(8),(true->[])),
                arc(s(8),2,s(10),(true->[])),
                arc(s(9),0,s(9),(true->[])),
                arc(s(9),1,s(9),(true->[])),
                arc(s(10),0,s(9),(true->[])),
                arc(s(10),1,s(10),(true->[])),
                arc(s(10),2,s(10),(true->[]))],
               [],[],[])).

ctr_specialisation(nb_peak, nb_peak_eq_5, kernel([source(s(1)),sink(s(11)),
    sink(s(12))],[arc(s(1),0,s(2),(true->[])),arc(s(1),1,s(1),(true->[])),arc(
    s(1),2,s(1),(true->[])),arc(s(2),0,s(2),(true->[])),arc(s(2),1,s(2),(true
    ->[])),arc(s(2),2,s(3),(true->[])),arc(s(3),0,s(4),(true->[])),arc(s(3),1,
    s(3),(true->[])),arc(s(3),2,s(3),(true->[])),arc(s(4),0,s(4),(true->[])),
    arc(s(4),1,s(4),(true->[])),arc(s(4),2,s(5),(true->[])),arc(s(5),0,s(6),(
    true->[])),arc(s(5),1,s(5),(true->[])),arc(s(5),2,s(5),(true->[])),arc(s
    (6),0,s(6),(true->[])),arc(s(6),1,s(6),(true->[])),arc(s(6),2,s(7),(true
    ->[])),arc(s(7),0,s(8),(true->[])),arc(s(7),1,s(7),(true->[])),arc(s(7),2,
    s(7),(true->[])),arc(s(8),0,s(8),(true->[])),arc(s(8),1,s(8),(true->[])),
    arc(s(8),2,s(9),(true->[])),arc(s(9),0,s(10),(true->[])),arc(s(9),1,s(9),(
    true->[])),arc(s(9),2,s(9),(true->[])),arc(s(10),0,s(10),(true->[])),arc(s
    (10),1,s(10),(true->[])),arc(s(10),2,s(12),(true->[])),arc(s(11),0,s(11),(
    true->[])),arc(s(11),1,s(11),(true->[])),arc(s(12),0,s(11),(true->[])),arc
    (s(12),1,s(12),(true->[])),arc(s(12),2,s(12),(true->[]))],[],[],[])).

ctr_specialisation(nb_peak, nb_peak_eq_up, kernel([source(t),sink(t),sink(r),
    sink(t1)],[arc(t,0,r,(true->[])),arc(t,1,t1,(true->[])),arc(t,2,t1,(true
    ->[])),arc(r,0,r1,(true->[])),arc(r,1,r1,(true->[])),arc(r,2,t,(true->[]))
    ,arc(t1,0,r1,(true->[])),arc(r1,2,t1,(true->[]))],[],[],[])).

ctr_specialisation(nb_peak, nb_peak_is_even, kernel([source(sE),sink(sE),sink
    (rE)],[arc(sE,0,rE,(true->[])),arc(sE,1,sE,(true->[])),arc(sE,2,sE,(true
    ->[])),arc(rE,0,rE,(true->[])),arc(rE,1,rE,(true->[])),arc(rE,2,sO,(true
    ->[])),arc(sO,0,rO,(true->[])),arc(sO,1,sO,(true->[])),arc(sO,2,sO,(true
    ->[])),arc(rO,0,rO,(true->[])),arc(rO,1,rO,(true->[])),arc(rO,2,sE,(true
    ->[]))],[],[],[])).

ctr_specialisation(nb_peak, nb_peak_is_odd, kernel([source(sE),sink(sO),sink(
    rO)],[arc(sE,0,rE,(true->[])),arc(sE,1,sE,(true->[])),arc(sE,2,sE,(true
    ->[])),arc(rE,0,rE,(true->[])),arc(rE,1,rE,(true->[])),arc(rE,2,sO,(true
```

```
->[])),arc(sO,0,rO,(true->[])),arc(sO,1,sO,(true->[])),arc(sO,2,sO,(true
->[])),arc(rO,0,rO,(true->[])),arc(rO,1,rO,(true->[])),arc(rO,2,sE,(true
->[]))],[],[],[])).
```

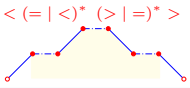
A.2 PDF Pages

The following synthesised PDF pages provides the corresponding synthesised LaTeX catalogue entry for the NB_PEAK time-series constraint. The bounds for time-series constraints, presented in Chapter 7, were integrated into the **Restrictions** slot, and the conditional automata, presented in Chapter 11, were integrated into the **Specialisation** slot.

PL

3.396 NB_PEAK

► ▷ ◁ ◄ DESCRIPTION AUTOMATON



Origin Based on the PEAK pattern.

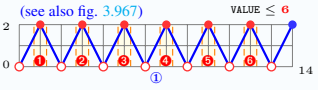
Constraint NB_PEAK(VALUE, VARIABLES)

Arguments

VALUE	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

$\texttt{sv} \leq 2 \vee \texttt{rv} \leq 1 \Rightarrow \texttt{VALUE} = 0$
$\texttt{VALUE} \geq 0$
$\texttt{VALUE} \leq \max(0, \lfloor(\texttt{sv} - 1)/2\rfloor)$①
required(VARIABLES, var)
where
$\texttt{sv} = |\texttt{VARIABLES}|$
rv =range(VARIABLES.var)



Purpose VALUE is the number of occurrences of the PEAK pattern in the time-series given by the VARIABLES collection. If the pattern does not occur, VALUE takes the default value 0. An occurrence of the pattern PEAK is the *maximal* subsequence which matches the regular expression '$< (= \mid <)^* (> \mid =)^* >$'.

Example $(3, \langle 7, 5, 5, 1, 4, 5, 2, 2, 3, 5, 6, 2, 3, 3, 3, 1\rangle)$

Figure 3.963 provides an example where the NB_PEAK $(3, [7, 5, 5, 1, 4, 5, 2, 2, 3, 5, 6, 2, 3, 3, 3, 1])$ constraint holds.

Typical
$|\texttt{VARIABLES}| > 2$
range(VARIABLES.var) > 1

Symmetries
- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties Functional dependency: VALUE determined by VARIABLES.

NB_PEAK 1789

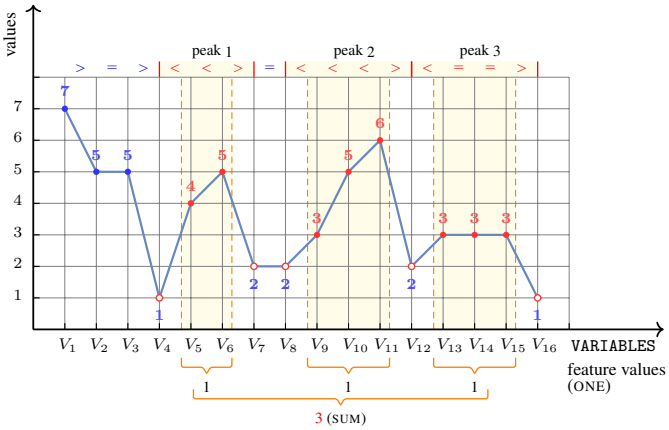


Figure 3.963: Illustrating the NB_PEAK constraint of the **Example** slot

1790 NB_PEAK

Automaton Figures 3.964 and 3.965 respectively depict the automaton associated with the constraint NB_PEAK and its simplified form.

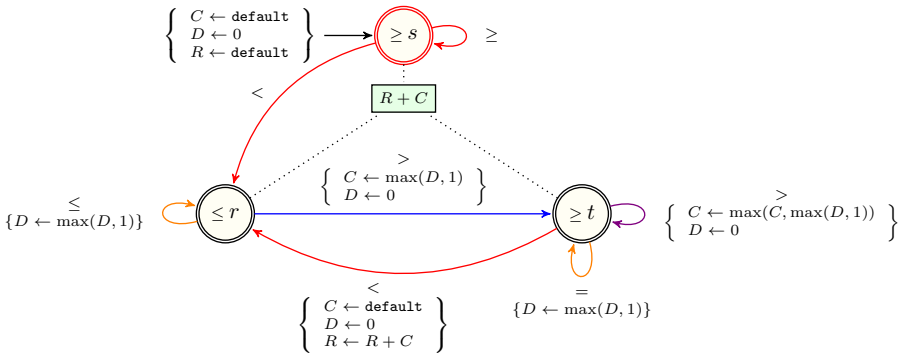


Figure 3.964: Automaton for the NB_PEAK constraint obtained by applying decoration Table 2.36 to the seed transducer of the PEAK pattern where `default` is 0

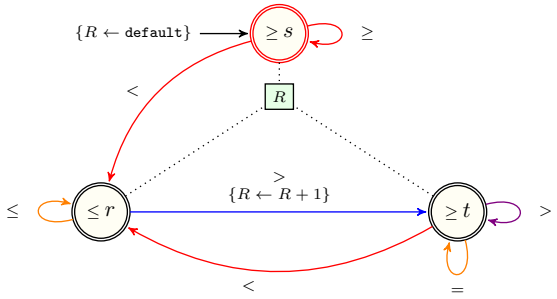


Figure 3.965: Simplified automaton for the NB_PEAK constraint obtained by applying decoration Table 2.38 to the seed transducer of the PEAK pattern where `default` is 0; $R_i - R_{i-1} \geq 0$ and $-R_i + R_{i-2} + 1 \geq 0$ are linear invariants.

NB_PEAK 1791

	s	r	t
s	$\overrightarrow{C} + \overleftarrow{C}$	$\overrightarrow{C} + \overleftarrow{C}$	$\overrightarrow{C} + \overleftarrow{C}$
r	$\overrightarrow{C} + \overleftarrow{C}$	1 C	1 R
t	$\overrightarrow{C} + \overleftarrow{C}$	1 L	$\overrightarrow{C} + \overleftarrow{C}$

Table 3.241: Concrete glue matrix, derived from the parametrised glue matrix 2.11, for the NB_PEAK constraint defined as the composition of the PEAK pattern, the feature ONE, and the aggregator sum; cells of the glue matrix are coloured with the colour of the constituent to which they are related.

	s	r	t
s	0	0	0
r	0	1 C	0 R
t	0	0 L	0

Table 3.242: Concrete glue matrix, derived from the parametrised glue matrix 2.11, for the simplified automaton of the NB_PEAK constraint defined as the composition of the PEAK pattern, the feature ONE, and the aggregator sum; cells of the glue matrix are coloured with the colour of the constituent to which they are related.

1792 NB_PEAK

Specialisation

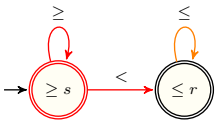


Figure 3.966: Automaton without registers for the NB_PEAK_EQ_0 constraint; it describes all sequences containing no occurrence of the PEAK pattern on a sequence of variables; it is derived from the automaton that counts the number of occurrences of the PEAK pattern by removing the register R, the **found** transition from state r to t that increments R, and the state t that becomes unreachable after removing transition $r \rightarrow t$.

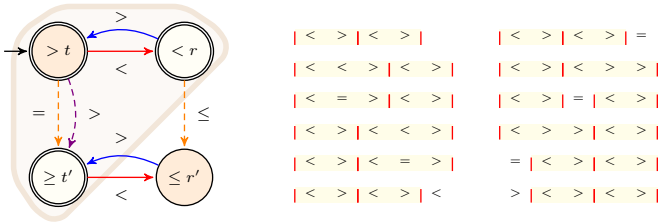


Figure 3.967: (**left**) Automaton without registers for the NB_PEAK_EQ_UP constraint; it describes all sequences containing the maximum number of occurrences of the PEAK pattern on a sequence of sv variables, i.e. $\max(0, \lfloor \frac{\mathtt{sv}-1}{2} \rfloor)$ of the **Restrictions** slot (see ①); transitions in blue correspond to a new occurrence of pattern, dashed transitions to slack, and accepting states have a light brown background; state t is accepting when $\mathtt{sv} \bmod 2 = 1$, while states r and t' are accepting when $\mathtt{sv} \bmod 2 = 0$. (**right**) All corresponding solutions for $\mathtt{sv} - 1 \in \{4, 5\}$.

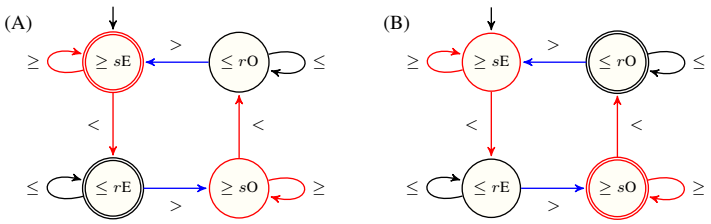


Figure 3.968: Automata without registers for the (A) NB_PEAK_IS_EVEN and the (B) NB_PEAK_IS_ODD constraints; they respectively achieve an even/odd number of occurrences of the PEAK pattern on a sequence of n variables; transitions in blue correspond to a new occurrence of pattern.

Appendix B

An Entry of the Database of Invariants of the Global Constraint Catalogue

B.1 Metadata

The following synthesised code illustrates the metadata of an entry of the database of invariants of the Global Constraint Catalogue. This entry contains linear and non-linear invariants obtained by the methods presented in Chapters 9 and 10, respectively.

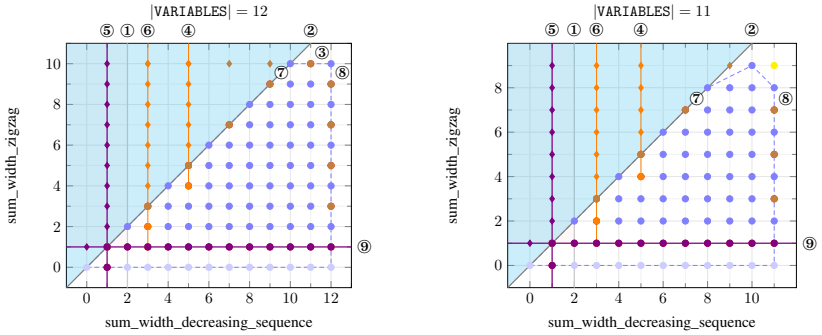
```
invariant_format([sum_width_decreasing_sequence,sum_width_zigzag], [x,y],
        [y>0#=>x>=2,
         y=<x,
         y=\=(sv-2)*min(1,max(0,sv-3))#\/x<3#\/x>sv*min(1,max(0,sv-1))-1#\/1=
             sv mod 2#\/0=x mod 2,
         x=\=5#\/y<4,
         x=\=1,
         x=\=3#\/y<1,
         y=\=x#\/0=x mod 2,
         x=\=sv*min(1,max(0,sv-1))#\/y<1#\/y>(sv-2)*min(1,max(0,sv-3))-1#\/0=
             y mod 2,
         y=\=1]).
```

B.2 PDF Pages

The following page provides the corresponding synthesised LaTeX entry of the database of invariants of the Global Constraint Catalogue.

SUM_WIDTH_DECREASING_SEQUENCE(x, VARIABLES) ∧
SUM_WIDTH_ZIGZAG(y, VARIABLES)

① $y > 0 \Rightarrow x \geq 2$
② $y \leq x$
③ $\bigvee \left(\begin{array}{l} y \neq (|\texttt{VARIABLES}| - 2) * \texttt{min}(1, \texttt{max}(0, |\texttt{VARIABLES}| - 3)), \\ x < 3, \\ x > |\texttt{VARIABLES}| * \texttt{min}(1, \texttt{max}(0, |\texttt{VARIABLES}| - 1)) - 1, \\ 1 = |\texttt{VARIABLES}| \bmod 2, \\ 0 = x \bmod 2 \end{array} \right)$
④ $x \neq 5 \vee y < 4$
⑤ $x \neq 1$
⑥ $x \neq 3 \vee y < 1$
⑦ $y \neq x \vee 0 = x \bmod 2$
⑧ $\bigvee \left(\begin{array}{l} x \neq |\texttt{VARIABLES}| * \texttt{min}(1, \texttt{max}(0, |\texttt{VARIABLES}| - 1)), \\ y < 1, \\ y > (|\texttt{VARIABLES}| - 2) * \texttt{min}(1, \texttt{max}(0, |\texttt{VARIABLES}| - 3)) - 1, \\ 0 = y \bmod 2 \end{array} \right)$
⑨ $y \neq 1$



Appendix C

Tables with Regular-Expression Characteristics

In this appendix, we give tables with the values of regular-expression characteristics presented in Chapters 7 and 8 for 22 regular expressions of [10].

name σ	regular expression	ω_σ
Bump	‘ >><>> ’	5
Dec	‘ > ’	1
DecSeq	‘(> (> | =)*)* > ’	1
DecTer	‘ >= =* > ’	3
Dip	‘ <<>>< ’	5
Gorge	‘(> | > (> | =)* >)(< | < (< | =)* <)’	2
Inc	‘ < ’	1
IncSeq	‘(< (< | =)*)* < ’	1
IncTer	‘ <= =* < ’	3
Inflexion	‘ < (< | =)* > | > (> | =)* < ’	2
Peak	‘ < (< | =)*(> | =)* > ’	2
Plain	‘ > =* < ’	2
Plateau	‘ < =* > ’	2
PropPlain	‘ >= =* < ’	3
PropPlateau	‘ <= =* > ’	3
Steady	‘ = ’	1
SteadySeq	‘ = =* ’	1
SDecSeq	‘ > >* ’	1
SIncSeq	‘ < <* ’	1
Summit	‘(< | < (< | =)* <)(> | > (> | =)* >)’	2
Valley	‘ > (> | =)*(< | =)* < ’	2
Zigzag	‘(<>)* <>< (> |ε) | (><)* ><> (< |ε)’	3

Table C.1 – Regular-expression short names σ (see Table 7.1) and corresponding *size* (see Definition 7.1.1); within each regular expression subparts corresponding to a smallest length word are highlighted in yellow.

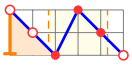
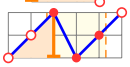
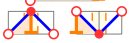
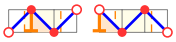
name σ	illustration	η_σ
`Bump`		2
`Dec`		1
`DecSeq`		1
`DecTer`		2
`Dip`		2
`Gorge`		1
`Inc`		1
`IncSeq`		1
`IncTer`		2
`Inflexion`		1
`Peak`		1
`Plain`		1
`Plateau`		1
`PropPlain`		1
`PropPlateau`		1
`Steady`		0
`SteadySeq`		0
`SDecSeq`		1
`SIncSeq`		1
`Summit`		1
`Valley`		1
`Zigzag`		1

Table C.2 – Regular-expression short names σ (see Table 7.1) and corresponding *height* shown as thick orange vertical line segments (see Definition 7.1.4)

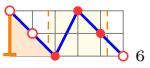
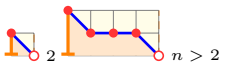
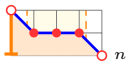
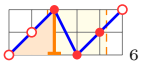
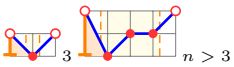
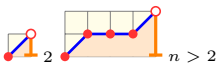
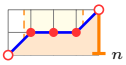
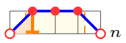
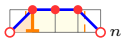
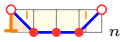
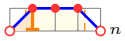
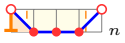
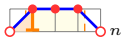
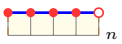
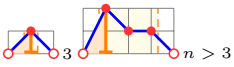
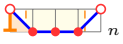
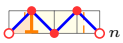
name σ	$\langle e_\sigma, c_\sigma \rangle$	illustration	$\phi_\sigma^{\langle n \rangle}$
`Bump`	undefined		$\begin{cases} 2 & \text{if } n = \omega_\sigma + 1 \\ \text{undefined} & \text{otherwise} \end{cases}$
`Dec`	undefined		$\begin{cases} 1 & \text{if } n = \omega_\sigma + 1 \\ \text{undefined} & \text{otherwise} \end{cases}$
`DecSeq`	$\langle 0, 1 \rangle$		$\begin{cases} 1 & \text{if } n = \omega_\sigma + 1 \\ 2 & \text{if } n > \omega_\sigma + 1 \end{cases}$
`DecTer`	$\langle 0, 0 \rangle$		2
`Dip`	undefined		$\begin{cases} 2 & \text{if } n = \omega_\sigma + 1 \\ \text{undefined} & \text{otherwise} \end{cases}$
`Gorge`	$\langle 0, 1 \rangle$		$\begin{cases} 1 & \text{if } n = \omega_\sigma + 1 \\ 2 & \text{if } n > \omega_\sigma + 1 \end{cases}$
`Inc`	undefined		$\begin{cases} 1 & \text{if } n = \omega_\sigma + 1 \\ \text{undefined} & \text{otherwise} \end{cases}$
`IncSeq`	$\langle 0, 1 \rangle$		$\begin{cases} 1 & \text{if } n = \omega_\sigma + 1 \\ 2 & \text{if } n > \omega_\sigma + 1 \end{cases}$
`IncTer`	$\langle 0, 0 \rangle$		2
`Inflexion`	$\langle 0, 0 \rangle$		1
`Peak`	$\langle 0, 0 \rangle$		1
`Plain`	$\langle 0, 0 \rangle$		1
`Plateau`	$\langle 0, 0 \rangle$		1
`PropPlain`	$\langle 0, 0 \rangle$		1
`PropPlateau`	$\langle 0, 0 \rangle$		1
`Steady`	undefined		$\begin{cases} 0 & \text{if } n = \omega_\sigma + 1 \\ \text{undefined} & \text{otherwise} \end{cases}$
`SteadySeq`	$\langle 0, 0 \rangle$		0
`SDecSeq`	$\langle 1, 0 \rangle$		$n - 1$
`SIncSeq`	$\langle 1, 0 \rangle$		$n - 1$
`Summit`	$\langle 0, 1 \rangle$		$\begin{cases} 1 & \text{if } n = \omega_\sigma + 1 \\ 2 & \text{if } n > \omega_\sigma + 1 \end{cases}$
`Valley`	$\langle 0, 0 \rangle$		1
`Zigzag`	$\langle 0, 0 \rangle$		1

Table C.3 – Regular-expression short names σ (see Table 7.1) and corresponding *range* shown as thick orange vertical line segments (see Definition 7.1.5); for a non-fixed-length regular expression σ and for any $n > \omega_\sigma + 1$, $\phi_\sigma^{\langle n \rangle} = e_\sigma \cdot (n - 1 - \eta_\sigma) + c_\sigma + \eta_\sigma$, where ω_σ and η_σ respectively correspond to the *size* (see Definition 7.1.1) and the *height* (see Definition 7.1.4) of the corresponding σ.

name σ	regular expression	Θ_σ
Bump	‘ >><>> ’	{‘ >><>> ’}
Dec	‘ > ’	{‘ > ’}
DecSeq	‘(> (> | =)*)* > ’	{‘ > ’}
DecTer	‘ >= =* > ’	{‘ >=> ’}
Dip	‘ <<><< ’	{‘ <<><< ’}
Gorge	‘(> (> | =)*)* >< ((< | =)* <)*’	{‘ >< ’}
Inc	‘ < ’	{‘ < ’}
IncSeq	‘(< (< | =)*)* < ’	{‘ < ’}
IncTer	‘ <= =* < ’	{‘ <=< ’}
Inflexion	‘ < (< | =)* > | > (> | =)* < ’	{‘ <> ’, ‘ >< ’}
Peak	‘ < (< | =)*(> | =)* > ’	{‘ <> ’}
Plain	‘ > =* < ’	{‘ >< ’}
Plateau	‘ < =* > ’	{‘ <> ’}
PropPlain	‘ >= =* < ’	{‘ >=< ’}
PropPlateau	‘ <= =* > ’	{‘ <=> ’}
Steady	‘ = ’	{‘ = ’}
SteadySeq	‘ = =* ’	{‘ = ’}
SDecSeq	‘ > >* ’	{‘ > ’}
SIncSeq	‘ < <* ’	{‘ < ’}
Summit	‘(< (< | =)*)* <> ((> | =)* >)*’	{‘ <> ’}
Valley	‘ > (> | =)*(< | =)* < ’	{‘ >< ’}
Zigzag	‘(<>)* <>< (> |ε) | (><)* ><> (< |ε)’	{‘ <>< ’, ‘ ><> ’}

Table C.4 – Regular-expression short names σ (see Table 7.1) and corresponding *inducing words* (see Definition 7.1.7)

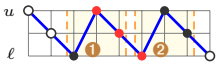
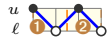
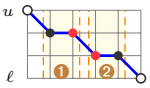
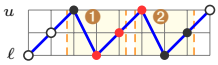
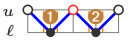
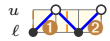
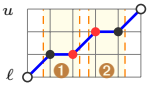
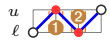
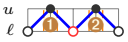
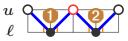
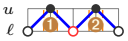
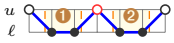
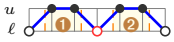
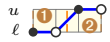
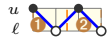
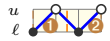
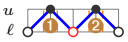
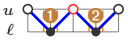
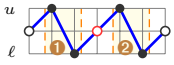
name σ	illustration	$o_\sigma^{\langle \ell, u \rangle}$
`Bump`		3
`Dec`		$\begin{cases} 0 & \text{if } u - \ell \leq \eta_\sigma \\ 1 & \text{otherwise} \end{cases}$
`DecSeq`		0
`DecTer`		$\begin{cases} 0 & \text{if } u - \ell \leq \eta_\sigma \\ 2 & \text{otherwise} \end{cases}$
`Dip`		3
`Gorge`		1
`Inc`		$\begin{cases} 0 & \text{if } u - \ell \leq \eta_\sigma \\ 1 & \text{otherwise} \end{cases}$
`IncSeq`		0
`IncTer`		$\begin{cases} 0 & \text{if } u - \ell \leq \eta_\sigma \\ 2 & \text{otherwise} \end{cases}$
`Inflexion`		2
`Peak`		1
`Plain`		1
`Plateau`		1
`PropPlain`		1
`PropPlateau`		1
`Steady`		1
`SteadySeq`		0
`SDecSeq`		0
`SIncSeq`		0
`Summit`		1
`Valley`		1
`Zigzag`		$\begin{cases} 0 & \text{if } u - \ell \leq \eta_\sigma \\ 1 & \text{otherwise} \end{cases}$

Table C.5 – Regular-expression short names σ (see Table 7.1) and corresponding *overlap* between two consecutive pattern occurrences ❶ and ❷ illustrated in red, i.e., • or ◦ (see Definition 7.1.10), where η_σ stands for the *height* characteristics of the corresponding σ (see Definition 7.1.4)

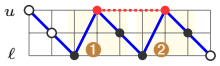
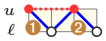
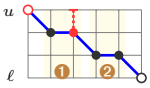
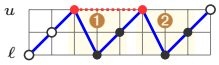
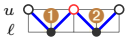
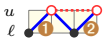
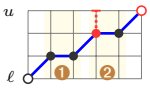
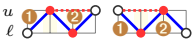
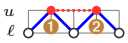
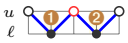
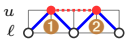
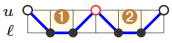
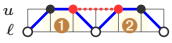
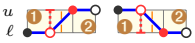
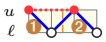
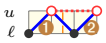
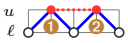
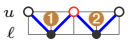
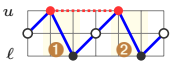
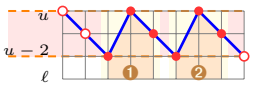
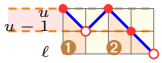
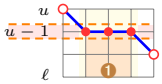
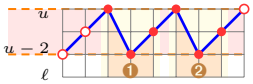
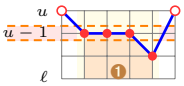
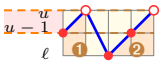
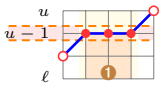
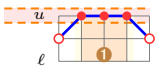
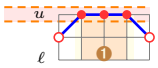
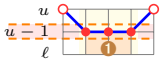
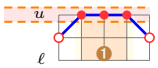
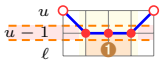
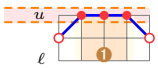
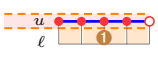
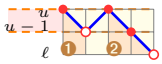
name σ	illustration	$\delta_\sigma^{\langle \ell,u \rangle}$
Bump		0
Dec		$\begin{cases} 0 & \text{if } u-\ell \le \eta_\sigma \\ -1 & \text{otherwise} \end{cases}$
DecSeq		0
DecTer		$\begin{cases} 0 & \text{if } u-\ell \le \eta_\sigma \\ -1 & \text{otherwise} \end{cases}$
Dip		0
Gorge		0
Inc		$\begin{cases} 0 & \text{if } u-\ell \le \eta_\sigma \\ 1 & \text{otherwise} \end{cases}$
IncSeq		0
IncTer		$\begin{cases} 0 & \text{if } u-\ell \le \eta_\sigma \\ 1 & \text{otherwise} \end{cases}$
Inflexion		0
Peak		0
Plain		0
Plateau		0
PropPlain		0
PropPlateau		0
Steady		0
SteadySeq		0
SDecSeq		0
SIncSeq		0
Summit		0
Valley		0
Zigzag		0

Table C.6 – Regular-expression short names σ (see Table 7.1) and corresponding *smallest variation of maxima* (see Definition 7.1.13), where η_σ stands for the *height* characteristics of the corresponding σ (see Definition 7.1.4); maxima of two consecutive pattern occurrences ❶ and ❷ are shown in red, i.e., ● or ○.

name σ	illustration	$\overline{\mathcal{I}}^{\langle \ell,u \rangle}_{\langle g,f,\sigma \rangle}$
`Bump`		$\begin{cases}[u-2,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Dec`		$\begin{cases}[u-1,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`DecSeq`		$\begin{cases}[u-1,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`DecTer`		$\begin{cases}[u-1,u-1], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Dip`		$\begin{cases}[u-2,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Gorge`		$\begin{cases}[u-1,u-1], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Inc`		$\begin{cases}[u-1,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`IncSeq`		$\begin{cases}[u-1,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`IncTer`		$\begin{cases}[u-1,u-1], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Inflexion`		$\begin{cases}[u,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Peak`		$\begin{cases}[u,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Plain`		$\begin{cases}[u-1,u-1], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Plateau`		$\begin{cases}[u,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`PropPlain`		$\begin{cases}[u-1,u-1], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`PropPlateau`		$\begin{cases}[u,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Steady`		$\begin{cases}[u,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`SteadySeq`		$\begin{cases}[u-1,u], \text{ if } u-\ell \geq \eta_\sigma \wedge g = \texttt{sum} \\ [u,u], \text{ if } u-\ell \geq \eta_\sigma \wedge (g = \texttt{max} \vee g = \texttt{min}) \\ \text{undefined, otherwise}\end{cases}$
`SDecSeq`		$\begin{cases}[u-1,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$

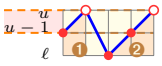
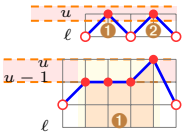
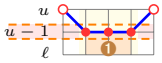
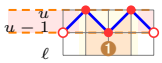
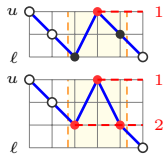
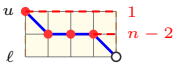
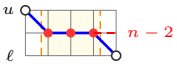
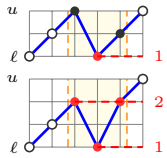
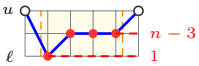
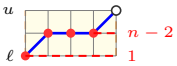
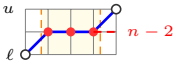
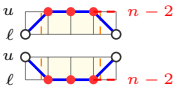
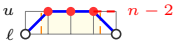
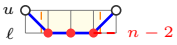
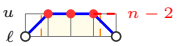
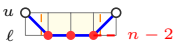
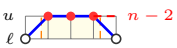
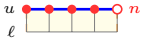
`SIncSeq`		$\begin{cases}[u-1,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Summit`		$\begin{cases}[u-1,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Valley`		$\begin{cases}[u-1,u-1], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$
`Zigzag`		$\begin{cases}[u-1,u], \text{ if } u-\ell \geq \eta_\sigma \\ \text{undefined, otherwise}\end{cases}$

Table C.7 – Regular-expression short names σ (see Table 7.1) and corresponding *interval of interest* (see Definition 8.2.1) of $\langle g, f, \sigma \rangle$ wrt $\langle \ell, u \rangle$, where η_σ stands for the *height* characteristics of the corresponding σ (see Definition 7.1.4); ℓ and u respectively stands for the minimum and maximum value of the variables of the time series.

name σ	illustration	$\mu_\sigma^{\langle \ell,u,n\rangle}(v)$
`Bump`		$\begin{cases} 1, \text{ if } u-\ell=\eta_\sigma \\ 1, \text{ if } u-\ell>\eta_\sigma \wedge v\in\{\ell,u-1,u\} \\ 2, \text{ if } u-\ell>\eta_\sigma \wedge v\in[\ell+1,u-2] \end{cases}$
`Dec`		$1, \forall v\in[\ell,u]$
`DecSeq`		$\begin{cases} 1, \text{ if } v\in\{\ell,u\} \\ n-2, \text{ if } v\in[\ell+1,u-1] \end{cases}$
`DecTer`		$\begin{cases} 0, \text{ if } v\in\{\ell,u\} \\ n-2, \text{ if } v\in[\ell+1,u-1] \end{cases}$
`Dip`		$\begin{cases} 1, \text{ if } u-\ell=\eta_\sigma \\ 1, \text{ if } u-\ell>\eta_\sigma \wedge v\in\{\ell,\ell+1,u\} \\ 2, \text{ if } u-\ell>\eta_\sigma \wedge v\in[\ell+2,u-1] \end{cases}$
`Gorge`		$\begin{cases} 0, \text{ if } v=u \\ n-3, \text{ if } v\in[\ell+1,u-1] \\ 1, \text{ if } v=\ell \end{cases}$
`Inc`		$1, \forall v\in[\ell,u]$
`IncSeq`		$\begin{cases} 1, \text{ if } v\in\{\ell,u\} \\ n-2, \text{ if } v\in[\ell+1,u-1] \end{cases}$
`IncTer`		$\begin{cases} 0, \text{ if } v\in\{\ell,u\} \\ n-2, \text{ if } v\in[\ell+1,u-1] \end{cases}$
`Inflexion`		$n-2, \ \forall v\in[\ell,u]$
`Peak`		$\begin{cases} 0, \text{ if } v=\ell \\ n-2, \text{ if } v\in[\ell+1,u] \end{cases}$
`Plain`		$\begin{cases} 0, \text{ if } v=u \\ n-2, \text{ if } v\in[\ell,u-1] \end{cases}$
`Plateau`		$\begin{cases} 0, \text{ if } v=\ell \\ n-2, \text{ if } v\in[\ell+1,u] \end{cases}$
`PropPlain`		$\begin{cases} 0, \text{ if } v=u \\ n-2, \text{ if } v\in[\ell,u-1] \end{cases}$
`PropPlateau`		$\begin{cases} 0, \text{ if } v=\ell \\ n-2, \forall v\in[\ell+1,u] \end{cases}$
`Steady`		$2, \ \forall v\in[\ell,u]$
`SteadySeq`		$n, \ \forall v\in[\ell,u]$
`SDecSeq`		$1, \ \forall v\in[\ell,u]$
`SIncSeq`		$1, \ \forall v\in[\ell,u]$

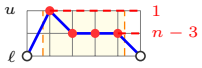
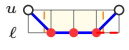
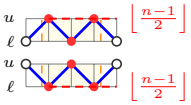
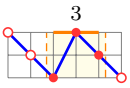
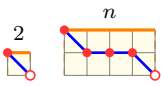
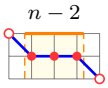
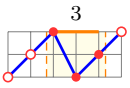
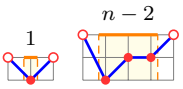
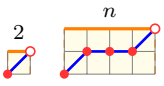
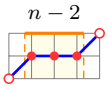
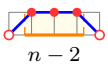
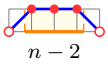
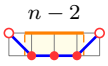
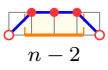
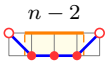
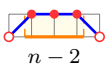
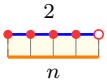
`Summit`		$\begin{cases} 1, \text{ if } v = u \\ n-3, \text{ if } v \in [\ell+1, u-1] \\ 0, \text{ if } v = \ell \end{cases}$
`Valley`		$\begin{cases} 0, \text{ if } v = u \\ n-2, \text{ if } v \in [\ell, u-1] \end{cases}$
`Zigzag`		$\left\lfloor \frac{n-1}{2} \right\rfloor, \forall v \in [\ell, u]$

Table C.8 – Regular-expression short names σ (see Table 7.1) and corresponding *maximum value occurrence number* of a value v (see Definition 8.2.2); for each pattern σ it assumes that the range of possible values is big enough to have at least one occurrence of pattern, i.e. $u - \ell \geq \eta_\sigma$ where u and ℓ are the largest and smallest value that can be used.

name σ	illustration	$\beta_\sigma^{\langle \ell, u \rangle}$
`Bump`		$\begin{cases} 3, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Dec`		$\begin{cases} 2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`DecSeq`		$\begin{cases} 2, \text{ if } u-\ell = \eta_\sigma \\ n, \text{ if } u-\ell > \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`DecTer`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Dip`		$\begin{cases} 3, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Gorge`		$\begin{cases} 1, \text{ if } u-\ell = \eta_\sigma \\ n-2, \text{ if } u-\ell > \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Inc`		$\begin{cases} 2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`IncSeq`		$\begin{cases} 2, \text{ if } u-\ell = \eta_\sigma \\ n, \text{ if } u-\ell > \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`IncTer`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Inflexion`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Peak`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Plain`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Plateau`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`PropPlain`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`PropPlateau`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Steady`		2
`SteadySeq`		n

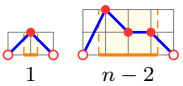
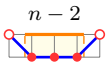
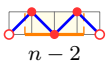
`SDecSeq`		$\begin{cases} u-\ell+1, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`SIncSeq`		$\begin{cases} u-\ell+1, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Summit`		$\begin{cases} 1, \text{ if } u-\ell = \eta_\sigma \\ n-2, \text{ if } u-\ell > \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Valley`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$
`Zigzag`		$\begin{cases} n-2, \text{ if } u-\ell \geq \eta_\sigma \\ 0, \text{ otherwise} \end{cases}$

Table C.9 – Regular-expression short names σ (see Table 7.1) and corresponding *big-width* (see Definition 8.2.3) shown as thick orange horizontal line segments, where η_σ stands for the *height* characteristics of the corresponding σ (see Definition 7.1.4)

Notation for Regular-Expression Characteristics

ω_σ the *size* of a regular expression σ (see Definition 7.1.1)

$\Omega_\sigma^{\langle \ell,u \rangle}(v)$ the *set of support time series* of two words v and w in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$ (see Definition 7.1.2)

$\eta_\sigma(v)$ the *height* of a word v in $\mathcal{L}_\sigma$ (see Definition 7.1.3)

η_σ the *height* of a regular expression σ (see Definition 7.1.4)

$\phi_\sigma^{\langle n \rangle}$ the *range* of a regular expression σ wrt $\langle n \rangle$ (see Definition 7.1.5)

Θ_σ the *set of inducing words* of a regular expression σ (see Definition 7.1.7, Table 5.2)

$\Gamma_\sigma^{\langle \ell,u \rangle}(v, w)$ the *set of superpositions* of two words v and w in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$ (see Definition 7.1.8)

$o_\sigma^{\langle \ell,u \rangle}(v, w)$ the *overlap* of two words v and w in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$ (see Definition 7.1.9)

$o_\sigma^{\langle \ell,u \rangle}$ the *overlap* of a regular expression σ wrt $\langle \ell, u \rangle$ (see Definition 7.1.10)

$\overline{\delta}_\sigma^{\langle \ell,u \rangle}(v, w, i)$ the *shift* of a subword w within a word v in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$ (see Definition 7.1.11)

$\delta_\sigma^{\langle \ell,u \rangle}(v, w)$ the *smallest variation of maxima* of two words w and v in $\mathcal{L}_\sigma$ wrt $\langle \ell, u \rangle$ (see Definition 7.1.12)

$\delta_\sigma^{\langle \ell,u \rangle}$ the *smallest variation of maxima* of a regular expression σ wrt $\langle \ell, u \rangle$ (see Definition 7.1.13)

$\mathcal{I}_{\langle g,f,\sigma \rangle}^{\langle \ell,u \rangle}$ the *interval of interest* of a constraint $\langle g, f, \sigma \rangle$ wrt $\langle \ell, u \rangle$ (see Definition 8.2.1)

$\mu_\sigma^{\langle \ell,u,n \rangle}(v)$ the *maximum value occurrence number* of an integer v wrt $\langle \ell, u, n \rangle$ (see Definition 8.2.2)

$\beta_\sigma^{\langle \ell,u,n \rangle}$ the *big width* of a regular expression σ wrt $\langle \ell, u, n \rangle$ (see Definition 8.2.3)

Index

Page numbers in bold face (as in **134**) point to a definition of a constraint name, or a concept. Page numbers in serif face (as in 134) indicate an occurrence of a constraint name or a concept.

0-gap automaton, 141
δ-gap automaton, 139, 140, **141**, 141–145, 153
`found`-transition, 149, 150
`maybe`$_{\texttt{b}}$-degree, **162**, 162
`maybe`$_{\texttt{b}}$-suffix, **162**
σ-pattern, **44**, 44, 47–50, 60, 61, 68–70, 77–79, 81–87, 89–92, 99–106, 141, 142, 144, 176, 177
ALLDIFFERENT, 109
AMONG, 17, 18, **37**, 37, 38, 40, 60, 61, 97, 98, 183
AMONG implied constraint, 18, 19, 22, 46, 59–61, 97–99, 101, 103, 104, 106, 107, 169, 175, 176
AUTOMATON, 40, **41**, 41, 109, 178
COST-REGULAR, **40**, 40, 41
MULTI-COST-REGULAR, 40, 41
PATTERN, 40
REGULAR, **40**, 40, 41
STRETCH, 17

abstract alphabet, 155, **156**, 156, 158–160, 162
abstract pattern, 155, **156**, 156, 159–162
acceptance function, **32**, 32, 49, 50, 118, 119, 135–137, 143–145, 150
accepting sequence, **33**, 33, 111–122
accepting state, **31**, 31–33, 111, 115, 118, 119, 136, 137, 139, 142, 144, 145
aggregator, 14, 15, 27, 43, **44**, 44, 45, 50, 53, 57, 59, 64, 66, 76, 86, 87, 92, 97, 100, 123, 158, 159, 164, 166, 176, 185
alphabet, 14, **29**, 29, 30, 33, 43, 44, 46, 47, 52, 109, 111, 139, 142, 157, 160–162
arity, **33**, 64, 112, 113, 156, 159
atomic relation, **129**, 130, 132, 135, 136, 138, 153, 154
automatically extracting and proving invariants, 126
automaton, 15, 16, 21, 22, 27, **31**, 31–35, 37, 39–41, 62, 65, 125, 130, 132, 135–139, 144, 145, 153, 154

backtrack, **39**, 39, 171, 172, 178
backtrack search, **39**
big width, 61, 99, **101**, 101
Boolean function, 125, 126, 129–132, 181
bound, 14–18, 21, 22, 59, 60, 65, 66, 169, 171–174, 177, 182–184
bound formula, 17, 185

checker, **39**, 40
combinatorial object, 14–17, 22, 23, 59, 169, 183
complement, 34, **35**, 35, 153, 154
concrete alphabet, **156**, 156, 158, 159
concrete pattern, 155, **156**, 156–162, 164
conditional automaton, 62, **130**, 135, 154, 183
conditional linear invariant, 121, 177
conjunction of global constraints, 109
conjunction of time-series constraints, 19, 22, 59, 61, 62, 125, 126, 128, 153, 175, 177, 178, 181, 183
constant atomic relation, **135**, 135–138, 153, 154
constant term, **110**, 115
constant-size automaton, 21, 62, 125, 128, 130, 134–136, 138, 139, 144, 153
constraint programming, 16, 17, 21, 22, 37, 39, 40
constraint satisfaction problem, 37, 38
convex hull, 19, 62, 120, 121, 126, 128, 129, 134
cut, 16

data mining, 15–17, 183
decomposition, 14, 155
decoration table, 49, 50, **51**, 52, 150
dependent atomic relation, 130
dependent Boolean function, **130**, 131, 132
disjunction of Boolean functions, 128, 129, 132
disjunction-capsuled regular expression, **30**, 30, 68, 69, 76

domain consistency, **38**, 38, 65, 109

e-occurrence, **44**, 44, 156–159, 162
extended σ-pattern, **44**, 44, 47, 60, 67, 69–74, 77, 78, 83, 84, 89, 90, 92
extended transducer-based model, 59, 63, 64, 155, 166, 183, 185

factor, **29**, 79–81, 91, 92
feature, 14, 15, 27, 43, **44**, 44, 45, 47, 49, 50, 53, 57, 59, 64, 66, 68, 76, 86, 97, 98, 155, 156, 164, 166, 185, 186
filtering, 39, 43
filtering algorithm, 38–40
footprint, 44
function over integer sequences, 15, 21, 155, **156**, 156–159, 185

gap, 139, 140, **141**, 141–148, 151, 152
gap atomic relation, **135**, 139, 153, 154
generalised arc consistency, **38**, 38
generic formula, 17, 18, 60
global constraint, 14, 17, 18, **37**, 37, 39–41, 64, 97, 166, 183
glue constraint, 15, 16, 43, 50, **51**, 52, 169, 171–174, 177, 184
glue expression, **51**, 52
glue matrix, **52**
guard, 121, 122
guard invariant, 121, 177, 184

height, 60, 66, 67, **68**, 68, 70, 71, 74, 75, 79–84, 86–92, 99, 102, 107, 223
heuristics, 14
hypothesis, 66, 81, 88, 92, 126, 128, 129, 181
hypothesis on the regular-expression characteristics, 76, 79, 84

i-occurrence, **44**, 44, 157, 158
implied constraint, 59, 99, 104, 105, 109
independent atomic relation, 130
independent Boolean function, **130**, 130
inducing word, 76
initial state, **31**, 31–33, 111, 115, 136, 137, 139, 144, 148, 162
input alphabet, **31**, 31–34, 40, 47, 51, 125, 160, 166
intersection, 31, **34**, 34, 35, 63, 109, 111–114, 116, 117, 117, 118, 120–122, 125, 130, 132, 153, 177
interval of interest, 61, 98, **99**, 99–102, 104, 106, 107
interval without restart, **77**, 77, 78, 82–85
invariant, 21, 61–63, 116, 117, 120, 121, 123, 177, 180–182
invariant digraph, **111**, 111–116, 119, 120, 122

language, 29, 30, 34, 35, 47, 48, 52, 60, 68, 71–73, 82, 83, 92, 100, 101, 109, 121, 132, 158, 183, 185
linear cut, 14, 123
linear implied inequality, 19
linear invariant, 15, 19, 34, 62, 109–111, 114–116, 117, 120–123, 126, 131, 169, 177, 182
linear model, 16, 183
linear programming, 16, 17
linear reformulation, 16
local search, 17, 183
loss, 140, **141**, 141–145, 147, 149, 150, 152
loss automaton, 140, 141, **142**, 142, 144–146, 148–151
loss interval, 142, 144, 148, 151–153
lower bound, 60, 61, 66–68, 86, 91, 92, 98, 101–104, 106, 107, 111–113, 145, 173

main register, **110**, 110, 113, 118, 119
maximal time series, **46**, 46, 76, 81, 83–85, 99, 100, 104–106
maximum value occurrence, 99
maximum value occurrence number, **100**, 100
methodology, 65, 125
minimal automaton, 136, 139, 141
mismatch overlap, 46, **47**, 47
modulo atomic relation, **135**, 138, 139, 154
modulo conditions, **138**, 138, 139

negative cycle, 111, 114, 115
non-linear implied constraint, 21
non-linear invariant, 19, 34, 62, 125, 126, 134, 135, 169, 181, 183
non-negativity conditions, 135, **136**, 136–138, 143–145, 151
not-greater atomic relation, **135**, 154
not-less atomic relation, **135**, 154
NP-complete, 38, 97
NP-completeness, 41, 185
NP-hard, 39, 109, 172
number of backtracks, 171, 175, 176, 178

output alphabet, **32**, 32, 47, 148, 160, 166
overlap, 60, 66, 69, **70**, 70, 71, 74–76, 79, 87, 88, 90, 99, 102, 107, 223

parameterised bound, 66
parameterised formula, 21, 60
pattern, 14, 18, 97, 155, 156, 159, 160, 162, 164
phase letter, **47**, 49, 148, 150, 160–162, 164–166, 183, 185
potential register, **110**, 111–113, 116–119
prefix, 15, **29**, 46, 47, 52, 150, 158, 161, 171, 174
prefix language, **46**
primary aggregator, **156**, 157
principal conditions, 140, **142**, 142–145, 150, 153
propagation, 38, 39, 50, 62, 65, 169, 183, 184, 186
propagator, 17, 63, 183
proper factor, **29**, 29, 71, 72, 83, 158

quantitative regular expression, 43, 52

range, 60, 66, **68**, 68, 74, 75, 86–89, 223
recognisable pattern, 46, **47**, 47, 136, 139, 146
reduced instruction set, 160, 162, 164, 185
reformulation, 16
register automaton, 16, 19, 22, 27, **31**, 31–35, 37, 39–41, 43, 46, 47, 49–52, 59, 61–63, 65, 109–116, 117, 117–123, 126, 132, 135–142, 145, 155, 160, 169, 171, 177, 183
regret, 140, **149**, 149, 150
regular expression, 14, 16, 22, 27, **29**, 29, 30, 34, 43–49, 51–53, 57, 59, 60, 62, 63, 65–69, 71, 73, 74, 76–93, 97, 98, 100–102, 104, 106, 107, 120, 121, 136, 139, 140, 142, 146, 148, 149, 151, 155, 156, 158, 160–162, 183, 185, 223
regular language, 27, 29, 30, 34, 65, 95
regular-expression characteristic, 59–62, 65–69, 71, 73, 74, 79, 85, 95, 97, 99–102, 107, 183, 185
regular-expression characteristics, 60
regular-expression overlap, **46**, 46
relative coefficient, **110**, 114, 115

s-occurrence, **44**, 44, 156–160, 162
secondary aggregator, **156**, 157, 159
seed transducer, 16, 46, **47**, 47–52, 62, 63, 136, 139, 140, 142, 145, 146, 148–150, 155, 160–162, 164, 183, 185
separated seed transducer, 148
sequence constraint, 16, 22, 59, 155, 183
set of inducing words, 60, 66, **68**, 75, 223
set of superpositions, **69**, 69, 71
set of supporting signatures, **130**, 130
set of supporting time series, **67**, 70, 82–84
sharp bound, 18, 59, 60, 68, 92, 107
sharp lower bound, 76, 77, 86, 91, 92
sharp upper bound, 18, 46, 67, 69–71, 76–79, 83–91
shortest path, 111, 115
signature, 21, **33**, 33–35, 41, 43–45, 47, 48, 50–52, 61, 62, 64–72, 76, 77, 79–87, 89, 90, 92, 100, 101, 110–114, 120–122, 130, 132, 135–145, 149, 150, 158, 159, 165, 178
signature function, 156
signature sequence, 159, 160, 162
size, 66, **67**, 67, 73–76, 223
smallest variation of maxima, 61, 66, 71, 72, **73**, 74, 75, 79, 82, 223
suffix, 15, **29**, 29, 47, 51, 52, 158, 161, 162, 171, 173–175, 177
superposition, **69**, 69, 71–73, 79–82
supporting signature, 137, 139
supporting time series, **67**, 71
sweep, 39
systematic method, 14, 22, 109, 120, 140, 146, 154, 183

t-occurrence, 161, 162
target hypothesis, 128, **129**, 129
time series, 14–17, 19, **43**, 43–45, 47–52, 60, 61, 65–74, 76–87, 89–92, 97–103, 105, 106, 125, 126, 128–132, 134, 135, 137, 140–147, 149–152, 171, 172, 175, 177, 178, 184, 186, 223
time-series constraint, 14–19, 21, 22, 27, 29, 30, 37, 40, 43, **44**, 44–47, 49–53, 57, 59–68, 76, 77, 79, 81, 83–92, 95, 97–104, 106, 107, 109, 112, 115, 125, 126, 128, 130, 131, 134, 135, 138–144, 146–153, 155, 169, 171, 172, 175, 177, 183–186
transducer, 16, 22, 27, 31, **32**, 32–34, 43, 47, 59, 65, 155, 160
transducer-based model, 17, 22, 183
transition, 21, **31**, 31–33, 35, 40, 41, 47–50, 63, 110–113, 116–122, 136, 137, 139, 140, 144, 145, 149, 150, 160, 161, 164
transition function, **31**, 32, 33, 137, 139, 144

union, 30, 31, 34, **35**, 35, 68, 99, 100, 153
unit commitment problem, 15
universally true Boolean function, 125, 129, 131, 132
upper bound, 60, 61, 66, 67, 76–79, 83, 84, 86–92, 139, 173

walk, **112**, 113

weighted digraph, 111, 115
well-formed function, **157**, 158, 161, 162, 164
well-formed seed transducer, **161**

Bibliography

[1] Sedigheh Abbasghorbani and Reza Tavoli. Survey on sequential pattern mining algorithms. In *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, pages 1153–1164, Nov 2015. 45

[2] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Math. Comput. Model.*, 17(7):57–73, April 1993. 39

[3] O. Zeynep Akşin, Mor Armony, and Vijay Mehrotra. The modern call center: A multi-disciplinary perspective on operations management research. *Production and Operations Management*, 16(6):665–688, 2007. 97

[4] Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 13–22. IEEE Computer Society, 2013. 65

[5] Rajeev Alur, Dana Fisman, and Mukund Raghothaman. Regular programming for quantitative properties of data streams. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 15–40. Springer, 2016. 15, 52, 65, 97, 166, 188

[6] Rajeev Alur, Konstantinos Mamouras, and Caleb Stanford. Automata-based stream processing. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 112:1–112:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. 52

[7] 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. 109

[8] 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, *CP 2016*, volume 9892 of *LNCS*, pages 13–29. Springer, 2016. 11, 15, 17, 18, 44, 50, 51, 52, 65, 66, 92, 93, 94, 97, 169, 171, 175, 177, 184, 185, 188, 191

[9] 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, 2016.

[10] Ekaterina Arafailova, Nicolas Beldiceanu, Rémi Douence, Mats Carlsson, Pierre Flener, María Andreína Francisco Rodríguez, Justin Pearson, and Helmut Simonis. Global Constraint Catalogue, Volume II, time-series constraints. *CoRR*, forthcoming, 2018. 9, 14, 17, 18, 19, 21, 30, 37, 48, 49, 50, 59, 65, 92, 121, 151, 155, 177, 188, 190, 191, 211

[11] 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, *CP-AI-OR 2016*, volume 9676 of *LNCS*, pages 18–34. Springer, 2016. 15, 16, 43, 49, 97, 136, 139, 171, 188, 189

[12] Ekaterina Arafailova, Nicolas Beldiceanu, and Helmut Simonis. among implied constraints for two families of time-series constraints. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, pages 38–54. Springer, Cham, 2017. 18, 97, 169, 175, 191

[13] 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, 23rd International Conference, CP 2017*, volume 10416 of *LNCS*, pages 21–37. Springer, Cham, 2017. 19, 35, 109, 131, 169, 177, 191

[14] 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. 11, 18, 21, 29, 44, 45, 65, 129, 141, 142, 169, 171, 191

[15] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, September 2001. 52

[16] J. Wesley Barnes and Robert M. Crisp Jr. Linear programming: A survey of general purpose algorithms. *A I I E Transactions*, 8(1):212–221, 1975. 14, 187

[17] Kaustubh Beedkar and Rainer Gemulla. Desq: Frequent sequence mining with subsequence constraints. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 793–798, Dec 2016. 166

[18] Luc de Raedt Behrouz Babaki, Tias Guns. Stochastic constraint programming with and-or branch-and-bound. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 539–545, 2017. 21, 191

[19] Nicolas Beldiceanu and Mats Carlsson. Sweep as a generic pruning technique applied to the non-overlapping rectangles constraints. In Toby Walsh, editor, *CP 2001*, volume 2239 of *LNCS*, pages 377–391. Springer, 2001. 17, 39, 190

[20] Nicolas Beldiceanu, Mats Carlsson, Romuald Debruyne, and Thierry Petit. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4):339–362, 2005. 27, 31, 57, 65, 155

[21] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present and future. *Constraints*, 12(1):21–62, Mar 2007. 14, 21, 37, 39, 155, 166, 183, 187

[22] 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. 13, 14, 16, 17, 27, 30, 40, 43, 44, 45, 46, 47, 48, 49, 51, 57, 59, 62, 63, 64, 65, 148, 151, 155, 160, 164, 166, 177, 188, 189, 190

[23] Nicolas Beldiceanu, Mats Carlsson, Pierre Flener, María Andreína Francisco Rodríguez, and Justin Pearson. Linking prefixes and suffixes for constraints encoded using automata with accumulators. In Barry O'Sullivan, editor, *CP 2014*, volume 8656 of *LNCS*, pages 142–157. Springer, 2014. 15, 50, 171, 188

[24] 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, 11th International Conference, CP 2005*, volume 3709 of *LNCS*, pages 92–106. Springer, 2005. 109

[25] Nicolas Beldiceanu and Evelyne Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97–123, 1994. 16, 17, 18, 37, 40, 64, 97, 98, 116, 155, 190, 191

[26] Nicolas Beldiceanu, Bárbara Dumas Feris, Philippe Gravey, Sabbir Hasan, Claude Jard, Thomas Ledoux, Yunbo Li, Didier Lime, Gilles Madi-Wamba, Jean-Marc Menaud, Pascal Morel, Michel Morvan, Marie-Laure Moulinard, Anne-Cécile Orgerie, Jean-Louis Pazat, Olivier Roux, and Ammar Sharaiha. Towards energy-proportional clouds partially powered by renewable energy. *Computing*, 99(1):3–22, Jan 2017. 15, 43, 188

[27] 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. 41, 109

[28] 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*, pages 733–748, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 15, 43, 97, 175, 177, 188

[29] 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, 10th International Conference, CP 2004*, volume 3258 of *LNCS*, pages 107–122. Springer, 2004. 27, 41, 109

[30] Nicolas Beldiceanu, Thierry Petit, and Guillaume Rochart. Bounds of graph characteristics. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, pages 742–746, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 21, 191

[31] Nicolas Beldiceanu, Thierry Petit, and Guillaume Rochart. Bounds of graph parameters for global constraints. *RAIRO - Operations Research*, 40(4):327–353, 2006. 60

[32] Nicolas Beldiceanu and Helmut Simonis. Modelseeker: Extracting global constraint models from positive examples. In *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, pages 77–95. 2016. 166

[33] Cédric Bentz, Denis Cornaz, and Bernard Ries. Packing and covering with linear programming: A survey. *European Journal of Operational Research*, 227(3):409 – 422, 2013. 14, 187

[34] Claude Berge. *Graphs and Hypergraphs*. Elsevier Science Ltd., Oxford, UK, UK, 1985. 39, 41

[35] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John Hooker. *Decision Diagrams for Optimization*. 01 2016. 17

[36] Christian Bessière, Remi Coletta, Emmanuel Hébrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Constraint Acquisition via Partial Queries. In *IJCAI 2013*, page 7, Beijing, China, June 2013. 97

[37] Christian Bessière, Remi Coletta, and Thierry Petit. Learning Implied Global Constraints. In *IJCAI 2007*, pages 50–55, Hyderabad, India, 2007. 97

[38] Christian Bessière, Abderrazak Daoudi, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Younes Mechqrane, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. New approaches to constraint acquisition. In *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, pages 51–76. 2016. 166

[39] Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, Claude-Guy Quimper, and Toby Walsh. Reformulating global constraints: The *slide* and *regular* constraints. In *SARA 2007*, volume 4612 of *LNAI*, pages 80–92. Springer, 2007. 17, 190

[40] Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Among, common and disjoint constraints. In *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2005*, pages 29–43, 2005. 98

[41] Christian Bessière, George Katsirelos, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Decomposition of the NValue constraint. In David Cohen, editor, *CP 2010*, volume 6308 of *LNCS*, pages 114–128. Springer, 2010. 17, 190

[42] John Adrian Bondy. *Graph Theory With Applications*. Elsevier Science Ltd., Oxford, UK, UK, 1976. 13

[43] Stéphane Bourdais, Philippe Galinier, and Gilles Pesant. Hibiscus: A constraint programming application to staff scheduling in health care. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, CP'03, pages 153–167, Berlin, Heidelberg, 2003. Springer-Verlag. 40

[44] 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*, pages 168–182, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. 140

[45] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 109

[46] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. 185

[47] David R. Brillinger. *Time Series: Data Analysis and Theory*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. 13

[48] Maurizio Bruglieri, Simona Mancini, Ferdinando Pezzella, and Ornella Pisacane. A new mathematical programming model for the green vehicle routing problem. *Electronic Notes in Discrete Mathematics*, 55:89 – 92, 2016. 14th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW16). 14, 187

[49] 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, 07–10 Jul 2017. PMLR. 128

[50] 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, February 1999. 128

[51] Marco Casazza and Alberto Ceselli. Mathematical programming algorithms for bin packing problems with item fragmentation. *Computers & Operations Research*, 46:1 – 11, 2014. 14, 187

[52] 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. 109, 125

[53] 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. 128

[54] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137 – 167, 1959. 40

[55] Edmund M. Clarke and Robert P. Kurshan. Computer-aided verification. *IEEE Spectr.*, 33(6):61–67, June 1996. 16, 190

[56] Thomas Colcombet and Laure Daviaud. Approximate comparison of functions computed by distance automata. *Theory Comput. Syst.*, 58(4):579–613, 2016. 65

[57] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007. 27, 29

[58] George B. Dantzig, Alex Orden, and Philip Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific J. Math.*, 5(2):183–195, 1955. 14

[59] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353 – 366, 1989. 39

[60] Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A `Cost-Regular` based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006. 40, 65

[61] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *ECAI*, pages 290–295, 1988. 109

[62] Lieven Eeckhout, Koen De Bosschere, and Henk Neefs. Performance analysis through synthetic trace generation. In *2000 ACM/IEEE Intl. Symp. Performance Analysis Syst. Software*, pages 1–6, 2000. 97

[63] Horst A. Eiselt and Carl-Louis Sandblom. *Applications of Network Flow Models*, pages 377–397. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. 14, 187

[64] Siemion Fajtlowicz. On Conjectures of Graffiti. *Annals of Discrete Mathematics*, 38:113–118, 1988. 125

[65] Filippo Focacci, Michela Milano, and Andrea Lodi. Solving tsp with time windows with constraints. In *Proceedings of the 1999 International Conference on Logic Programming*, pages 515–529, Cambridge, MA, USA, 1999. Massachusetts Institute of Technology. 14

[66] CeADAR Centre for Applied Data Analytics. `https://www.ceadar.ie`. 15, 188

[67] María Andreína Francisco Rodríguez, Pierre Flener, and Justin Pearson. Implied constraints for Automaton constraints. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *GCAI 2015*, volume 36 of *EasyChair Proceedings in Computing*, pages 113–126, 2015. 109

[68] María Andreína Francisco Rodríguez, Pierre Flener, and Justin Pearson. Automatic generation of descriptions of time-series constraints. In Alexander Brodsky, editor, *ICTAI 2017*. IEEE Computer Society, 2017. 16, 27, 46, 47, 62, 70, 136, 139, 146, 148, 183, 189

[69] Simon French, editor. *Decision Theory: An Introduction to the Mathematics of Rationality*. Halsted Press, New York, NY, USA, 1986. 140

[70] Eugene C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, January 1982. 39

[71] 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. 109

[72] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979. 41, 98

[73] Ian P. Gent, Chris Jefferson, Ian Miguel, and Peter Nightingale. Generating special-purpose stateless propagators for arbitrary constraints. In David Cohen, editor, *Principles and Practice of Constraint Programming – CP 2010*, pages 206–220, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 17

[74] Francisco Gómez-Vela, Francisco Martínez-Álvarez, Carlos D. Barranco, Norberto Díaz-Díaz, Domingo Savio Rodríguez-Baena, and Jesús S. Aguilar-Ruiz. Pattern recognition in biological time series. In Jose A. Lozano, José A. Gámez, and José A. Moreno, editors, *Advances in Artificial Intelligence*, pages 164–172, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 186

[75] Ralph Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958. 14, 123

[76] Tarik Hadzic, John N. Hooker, Barry O'Sullivan, and Peter Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In Peter J. Stuckey, editor, *Principles and Practice of Constraint Programming*, pages 448–462, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 17

[77] Pierre Hansen and Gilles Caporossi. Autographix: An automated system for finding conjectures in graph theory. *Electronic Notes in Discrete Mathematics*, 5:158–161, 2000. 109, 125

[78] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 607–613, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. 39

[79] John N. Hooker. *Integrated Methods for Optimization*. Springer Publishing Company, Incorporated, 2nd edition, 2011. 109

[80] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. 27, 31

[81] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 3rd edition, 2007. 21, 191

[82] Sophie Huczynska, Paul McKay, Ian Miguel, and Peter Nightingale. Modelling equidistant frequency permutation arrays: An application of constraints to mathematics. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, pages 50–64, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 13

[83] Yasunori Ishihara, Takuji Moroto, Shougo Shimizu, Kenji Hashimoto, and Toru Fujiwara. A tractable subclass of dtds for xpath satisfabilty with sibling axes. In Gardner Philippa and Geerts Floris, editors, *Database Programming Languages: 12th International Symposium, DBPL 2009s*, volume 5708 of *LNCS*, pages 68–83. Springer, 2009. 30

[84] Lars Kegel, Martin Hahmann, and Wolfgang Lehner. Template-based time series generation with loom. In *EDBT/ICDT Workshops 2016, Bordeaux, France*, 2016. 97

[85] Imre Lakatos. *Proofs and Refutations*. Cambridge University Press, 1976. 125

[86] Pat W. Langley, Herbert A. Simon, Gary L. Bradshaw, and Jan M. Zytkow. *Scientific Discovery – Computational Explorations of the Creative Process*. MIT Press, 1987. 125

[87] Craig Eric Larson and Nicolas Van Cleemput. Automated conjecturing I: Fajtlowicz's Dalmatian heuristic revisited. *Artif. Intell.*, 231:17–38, 2016. 125

[88] Eugene L. Lawler and David E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966. 21, 191

[89] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. Last conflict based reasoning. In *Proceedings of the 2006 Conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva Del Garda, Italy*, pages 133–137, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press. 39

[90] Jon Lee. All-different polytopes. *J. Comb. Optim.*, 6(3):335–352, 2002. 109

[91] Douglas B. Lenat. On automated scientific theory formation: a case study using the AM program. *Machine intelligence*, 9:251–286, 1979. 125

[92] Arnaud Letort, Mats Carlsson, and Nicolas Beldiceanu. Synchronized sweep algorithms for scalable scheduling constraints. *Constraints*, 20(2):183–234, Apr 2015. 39

[93] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 311–322, New York, NY, USA, 2005. ACM. 52

[94] Gilles Madi Wamba, Yunbo Li, Anne-Cécile Orgerie, Nicolas Beldiceanu, and Jean-Marc Menaud. Cloud workload prediction and generation models. In *SBAC-PAD: International Symposium on Computer Architecture and High Performance Computing*, Campinas, Brazil, October 2017. 16, 186, 189

[95] Arnaud Malapert, Christelle Guéret, and Louis-Martin Rousseau. A constraint programming approach for a batch processing problem with non-identical job sizes. *European Journal of Operational Research*, 221(3):533 – 545, 2012. 14, 187

[96] Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence A. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1-3):397–446, 2002. 14, 123

[97] Julien Menana. *Automata and Constraint Programming for Personnel Scheduling Problems*. Theses, Université de Nantes, October 2011. 109

[98] Julien Menana and Sophie Demassey. 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*, pages 178–192, 2009. 35, 109

[99] Julien Menana and Sophie Demassey. Sequencing and counting with the multicost-regular constraint. In Willem-Jan van Hoeve and John N. Hooker, editors, *CP-AI-OR 2009*, volume 5547 of *LNCS*, pages 178–192. Springer, 2009. 40, 41

[100] Jean-Noël Monette, Pierre Flener, and Justin Pearson. Towards solver-independent propagators. In Michela Milano, editor, *CP 2012*, volume 7514 of *LNCS*, pages 544–560. Springer, 2012. 17, 190

[101] Marc Nerlove, David M. Grether, and José L. Carvalho. *Analysis of economic time series*. Economic theory, econometrics, and mathematical economics. Academic Press, San Diego [u.a.], rev. ed edition, 1995. 186

[102] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP'07, pages 529–543, Berlin, Heidelberg, 2007. Springer-Verlag. 40

[103] Allen Newell and Herbert A. Simon. The logic theory machine – A complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79, 1956. 125

[104] Thiago F. Noronha, Andréa C. Santos, and Celso C. Ribeiro. Constraint programming for the diameter constrained minimum spanning tree problem. *Electronic Notes in Discrete Mathematics*, 30:93 – 98, 2008. The IV Latin-American Algorithms, Graphs, and Optimization Symposium. 13

[105] François Pachet and Pierre Roy. Automatic generation of music programs. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming – CP'99*, pages 331–345, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. 16, 17, 190

[106] Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, October 1981. 14, 187

[107] Guillaume Perez and Jean-Charles Régin. MDDs are efficient modeling tools: An application to some statistical constraints. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, pages 30–40, Cham, 2017. Springer International Publishing. 17

[108] Gilles Pesant. A filtering algorithm for the STRETCH constraint. In Toby Walsh, editor, *CP 2001*, volume 2239 of *LNCS*, pages 183–195. Springer, 2001. 16, 17, 64, 116, 155, 190

[109] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *CP 2004*, volume 3258 of *LNCS*, pages 482–495. Springer, 2004. 40, 65

[110] Gilles Pesant, Michel Gendreau, Jean-Yves Potvin, and Jean-Marc Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32:12–29, 1998. 14, 187

[111] Émilie Picard-Cantin, Mathieu Bouchard, Claude-Guy Quimper, and Jason Sweeney. Learning parameters for the sequence constraint from solutions. In Michel Rueher, editor, *CP 2016*, volume 9892 of *LNCS*, pages 405–420. Springer, 2016. 97

[112] D. Stephen G. Pollock. A handbook of time-series analysis, signal processing and dynamics. pages 1–733, 1999. 186

[113] CAMPUS 21 EU project. 15

[114] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017. 38

[115] Zhi quan Luo and Wei Yu. An introduction to convex optimization for communications and signal processing. *IEEE J. SEL. AREAS COMMUN*, pages 1426–1438, 2006. 14, 187

[116] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In Barbara Hayes-Roth and Richard E. Korf, editors, *AAAI 1994*, pages 362–367. AAAI Press, 1994. 17, 39, 190

[117] John Riordan. *An Introduction to Combinatorial Analysis*. Princeton University Press, 1978. 13

[118] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006. 13, 27, 37, 38, 187

[119] Jacques Sakarovitch. *Elements of Language Theory*. Cambridge University Press, 2009. 27, 32, 155

[120] Leonard Jimmie Savage. The theory of statistical decision. *Journal of the American Statistical Association*, 46(253):55–67, 1951. 140

[121] Andrea Sboner, Alessandro Romanel, Andrea Malossini, Federica Ciocchetta, Francesca Demichelis, Ivano Azzini, Enrico Blanzieri, and Rossana Dell'Anna. *Simple Methods for Peak and Valley Detection in Time Series Microarray Data*, pages 27–44. Springer US, Boston, MA, 2007. 45

[122] Jeffrey D. Scargle. Astronomical time series analysis. In Dan Maoz, Amiel Sternberg, and Elia M. Leibowitz, editors, *Astronomical Time Series*, pages 1–12, Dordrecht, 1997. Springer Netherlands. 186

[123] Felix Scholkmann, Jens Boss, and Martin Wolf. An efficient algorithm for automatic peak detection in noisy periodic and quasi-periodic signals. *Algorithms*, 5(4):588–603, 2012. 45

[124] Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961. 65

[125] Helmut Simonis, Paul Davern, Jacob Feldman, Deepak Mehta, Luis Quesada, and Mats Carlsson. A generic visualization platform for CP. In *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, pages 460–474, 2010. 172

[126] S.M. Sinha. *Mathematical Programming*. Elsevier Science, Burlington, 2006. 13, 187

[127] Ricardo Soto, Broderick Crawford, Eric Monfroy, Wenceslao Palma, and Fernando Paredes. Nurse and paramedic rostering with constraint programming: A case study. *Romanian Journal of Information Science and Technology*, 16:52–64, 01 2013. 40

[128] P. David Stotts and William Pugh. Parallel finite automata for modeling concurrent software systems. *Journal of Systems and Software*, 27(1):27 – 43, 1994. 16, 189

[129] Pascal Van Hentenryck, Vijay Saraswat, and Yves Deville. Design, implementation, and evaluation of the constraint language cc(FD). Technical Report 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. 17, 190

[130] Willem-Jan van Hoeve. The alldifferent Constraint: A Survey. *eprint arXiv:cs/0105015*, May 2001. 14, 187

[131] Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput*, 28:57–104, 1998. 14, 38, 187

[132] Eva Volna, Michal Janosek, and Martin Kotyrba. *Pattern recognition and classification in time series data*. Advances in Computational Intelligence and Robotics. Information Science Reference, 07 2016. 186

[133] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, and Peter Wolstenholme. *Modeling Software with Finite State Machines*. Auerbach Publications, Boston, MA, USA, 2006. 16, 189

[134] D. Michael Warner and Juan Prawda. A mathematical programming model for scheduling nursing personnel in a hospital. *Management Science*, 19(4):411–422, 1972. 14, 187

[135] Ebru Yilmaz. A mathematical programming model for scheduling of nurses' labor shifts. *Journal of Medical Systems*, 36(2):491–496, Apr 2012. 14, 187

[136] Chong Zhu, Xiangli Zhang, Jingguo Sun, and Bin Huang. Algorithm for mining sequential pattern in time series data. In *2009 WRI International Conference on Communications and Mobile Computing*, volume 3, pages 258–262, Jan 2009. 45

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Description fonctionnelle de contraintes sur des séquences et synthèse d'objets combinatoires

Mot clés : programmation par contraintes, automates, transducteurs, expressions régulières, séries temporelles, objets combinatoires paramétrés, invariants linéaires et non-linéaires

Resumé : à l'opposé de l'approche consistant à concevoir au cas par cas des contraintes et des algorithmes leur étant dédiés, l'objet de cette thèse concerne d'une part la description de familles de contraintes en termes de composition de fonctions, et d'autre part la synthèse d'objets combinatoires pour de telles contraintes. Les objets concernés sont des bornes précises, des coupes linéaires, des invariants non-linéaires et des automates finis ; leur but principal est de prendre en compte l'aspect combinatoire d'une seule contrainte ou d'une conjonction de contraintes. Ces objets sont obtenus d'une façon systématique et sont paramétrés par une ou plusieurs contraintes, par le nombre de variables dans une séquence, et par les domaines initiaux de ces variables. Cela nous permet d'obtenir des objets indépendants d'une instance considérée. Afin de synthétiser des objets combinatoires nous tirons partie de la vue declarative de telles contraintes, basée sur les expressions régulières, ansi que la vue opérationnelle, basée sur les automates à registres et les transducteurs finis. Il y a plusieurs avantages à synthétiser des objets combinatoires par rapport à la conception d'algorithmes dédiés : 1) on peut utiliser ces formules paramétrées dans plusieurs contextes, y compris la programmation par contraintes et la programmation linéaire, ce qui est beaucoup plus difficile avec des algorithmes ; 2) la synergie entre des objets combinatoires nous donne une meilleure performance en pratique ; 3) les quantités calculées par certaines des formules peuvent être utilisées non seulement dans le contexte de l'optimisation mais aussi pour la fouille de données.

Title : Functional Description of Sequence Constraints and Synthesis of Combinatorial Objects

Keywords : constraint programming, automata, transducers, regular expressions, time series, parameterised combinatorial objects, linear and non-linear invariants

Abstract: Contrary to the standard approach consisting in introducing ad hoc constraints and designing dedicated algorithms for handling their combinatorial aspect, this thesis takes another point of view. On the one hand, it focusses on describing a family of sequence constraints in a compositional way by multiple layers of functions. On the other hand, it addresses the combinatorial aspect of both a single constraint and a conjunction of such constraints by synthesising compositional combinatorial objects, namely bounds, linear inequalities, non-linear constraints and finite automata. These objects are obtained in a systematic way and are not instance-specific: they are parameterised by one or several constraints, by the number of variables in a considered sequence of variables, and by the initial domains of the variables. When synthesising such objects we draw full benefit both from the declarative view of such constraints, based on regular expressions, and from the operational view, based on finite transducers and register automata. There are many advantages of synthesising combinatorial objects rather than designing dedicated algorithms: 1) parameterised formulae can be applied in the context of several resolution techniques such as constraint programming or linear programming, whereas algorithms are typically tailored to a specific technique; 2) combinatorial objects can be combined together to provide better performance in practice; 3) finally, the quantities computed by some formulae cannot just be used in an optimisation setting, but also in the context of data mining.