SPECIAL SESIONS

6th Iberian Mathematical Meeting

S3
Theoretical Informatics / Computer Science

Thu 6th, 17:00 - 17:25, Aula 10 – J. Silva:
Algorithmic Debugging: A Road Map

Thu 6th, 18:00 - 18:25, Aula 10 – P. Lucio:
Software Development with Automatic Deductive Verifiers

Thu 6th, 19:00 - 19:25, Aula 10 – P. Real:
Toward a mathematical model for parallel topological computation within 3D digital image context

Fri 7th, 11:30 - 11:55, Aula 10 – G. Moreno:
Fuzzy Logic Programming and the FLOPER Environment

Fri 7th, 12:00 - 12:25, Aula 10 – C. Sánchez:
A Gentle Introduction to Linear Temporal Logic and How To Increase its Expressive Power

Fri 7th, 12:30 - 12:55, Aula 5 – X.A. Vila:
Analysis of heart rate variability with RHRV

Fri 7th, 16:30 - 16:55, Aula 10 – M. Gallardo:
Model Checking: A Formal Verification Technique with Industrial Applications

Fri 7th, 17:30 - 17:55, Aula 10 – E. Mayordomo:
Efficient Computation of Absolutely Normal Numbers

Fri 7th, 18:00 - 18:25, Aula 10 – C. Gómez:
On the NP-Hardness of Optimizing Binarizations of Context-Free Grammars

Fri 7th, 18:30 - 18:55, Aula 10 – M. A. Insua:
A refined algorithm for testing the Libniz n-algebra structure

Sat 8th, 9:30 - 9:55, Aula 10 – A. Pereira do Vale:
The Geometry of Musical Chords according to D. Tymoczko

Sat 8th, 10:00 - 10:25, Aula 10 – D. Losada:
Multi-Armed Bandits for Information Retrieval

Sat 8th, 10:30 - 10:55, Aula 10 – P. Páez:
Non-degeneracy conditions in automated proving and discovery

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

Algorithmic Debugging: A Road Map

Josep Silva[1]

Algorithmic debugging [1, 2, 3] is a semi-automatic technique to discover bugs in programs. It was originally defined in the logic programming paradigm, but it has been later adapted to all paradigms. The high level of abstraction of this technique allows for debugging programs without the need to see (or know about) the source code of the program being debugged. In this work I analyze the evolution of algorithmic debugging along its history, which last more than three decades. I present the milestones reached along that time, the problems found, the solutions proposed, and the main applications of algorithmic debugging. On the practical side, I analyze the main features that a modern algorithmic debugger must have, and I review the use in the industry and the academia of current algorithmic debuggers.

Keywords: Software Engineering, Debugging, Algorithmic Debugging

MSC 2010: 68N15, 68W40

References

[1] E. Y. SHAPIRO, *Algorithmic Program Debugging*. MIT Press, 1982.

[2] J. SILVA, A Survey on Algorithmic Debugging Strategies. *Advances in Engineering Software* **42**(11), 976–991 (2011).

[3] D. INSA, J. SILVA, A Generalized Model for Algorithmic Debugging. In *Logic-Based Program Synthesis and Transformation*, Moreno Falaschi (ed.), 261–276. LNCS, 2015.

[1]Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n - ZIP: 46022 - Valencia (Spain)
jsilva@dsic.upv.es

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

Software Development with Automatic Deductive Verifiers*

Paqui Lucio[1]

Formal verification is a technique that mathematically proves the absence of errors in computer programs. Deductive verification is based on the axiomatic semantics of programming languages that was introduced by C.A.R. Hoare in the late 60s: Hoare's formal system or Hoare's logic ([1]. Such formal system has been widely studied and applied in different areas of programming languages and software engineering, given rise to the software development method known as "design by contract", programming languages that incorporate automated assertions, and verifiers that check (or help the user to check) that these assertions are true. In particular, deductive verifiers are based on using an automated theorem prover to automatically show the properties in which the program correction (with respect to its specification) is based. In this talk we introduce the evolution from Hoare logic to today's automated deductive verification tools (see e.g. [2]).

Keywords: Formal verification, Assertion, Hoare's logic.

MSC 2010: 68N30, 68Q55

References

[1] HOARE, C. A. R., An Axiomatic Basis for Computer Programming. *Communications of the ACM*, **volumen**(12), 576–580 (1969).

[2] BERNHARD BECKERT AND REINER HÄHNLE, Reasoning and Verification: State of the Art and Current Trends. *Intelligent Systems, IEEE* **volumen**(29), 20–29 (2014).

[1]Department of Computer Languages and Systems
University of the Basque Country
Paseo Manuel de Lardizabal, 1, 2018-San Sebastián
paqui.lucio@ehu.eus

*Partially supported by the Spanish Project COMMAS (TIN2013-46181-C2-2-R), the Basque Project LoRea (GIU12/26) and grant BAILab (UFI11/45).

[1]Department of Mathematics
The University of Texas at Austin
2515 Speedway, Austin, TX 78712
asodre@math.utexas.edu

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

Toward a mathematical model for parallel topological computation within 3D digital image context

F. Diaz del Rio[1] **D. Onchis-Moaca**[2] **P. Real**[3]

This talk is concerned with the problem of developing a topologically-consistent framework for efficient parallel topological analysis and recognition in 3D digital imagery. A topological consistency proof of such systems is provided in most of the cases by means of a mathematical model of digital images and objects, under which all theoretical formulae related to topology are true and there is no room for paradoxes. The idea is to suitably generalize to 3D the promising parallel algorithmic results on combinatorial optimization over 2D digital images obtained in [1, 2, 3]. We propose a suitable generalization of the classical notion of Abstract Cell Complex, called primal-dual abstract cell complex (or pACC for short), as theoretical model of our framework and particular asymmetric pACCs, called Homological Spanning Forest (HSF, for short), as a peculiar system of "interaction dynamics", topologically describing digital objects with 6-adjacency.. We aim to achieve parallel architectures compatible with this framework and to reduce drastically the time complexity in topological computations within 3D digital imagery. The new notion of topological hole tree structure of a binary 3D digital image is defined and an algorithm for computing it starting from a HSF representation is given.

Keywords: digital image, topological representation, parallel algorithm

References

[1] F. DIAZ-DEL-RIO, P. REAL, D. ONCHIS, A parallel Homological Spanning Forest framework for 2D topological image analysis, *Accepted in Pattern Recognition Letters(2016)*

[2] F. DIAZ-DEL-RIO, P. REAL, D. ONCHIS, A Parallel Implementation for Computing the Region-Adjacency-Tree of a Segmentation of a 2D Digital Image, *Lecture Notes in Computer Science* **vol. 9555** (2016) 98–109.

[3] H. MOLINA-ABRIL, P. REAL, Homological spanning forest framework for 2d image analysis, *Annals of Mathematics and Artificial Intelligence* **64** (2012) 385–409.

[1]H.T.S. Informatics Engineering,
University of Seville, Seville, Spain
fdiaz@us.es, real@us.es

[2]Department of Mathematics
Faculty of Mathematics
University of Vienna, Austria
darian.onchis@univie.ac.at

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

Fuzzy Logic Programming and the FLOPER Environment*

Ginés Moreno[1]

The challenging research area of *Fuzzy Logic Programming*, which is devoted to introduce *fuzzy logic* concepts into *logic programming* in order to explicitly deal with vagueness in a natural way, has provided an extensive variety of Prolog dialects along the last three decades. FASILL (acronym of *"Fuzzy Aggregators and Similarity Into a Logic Language"*) is a fuzzy logic language with truth degree annotations, a great variety of connectives and unification by similarity. Here we describe its syntax, operational semantics (where the fuzzified resolution principle replaces syntactic unification by weak, similarity-based unification) and declarative semantics (based on a fuzzy variant of the classical notion of least Herbrand model coping now with truth degrees and similarity relations). We also give some implementation details on the FLOPER system developed in our research group, which has been used for coding real-world applications in emergent fields like cloud computing or the semantic web.

Keywords: Fuzzy Logic Programming, Fuzzy Logic, Logic Programming

MSC 2010: 03B52, 68N17, 68T35, 06B23

References

[1] J. M. Almendros-Jiménez, A. Luna, and G. Moreno. Fuzzy xpath through fuzzy logic programming. *New Generation Computing*, 33(2):173–209, 2015.

[2] P. Julián Iranzo, G. Moreno, J. Penabad, and C. Vázquez. A declarative semantics for a fuzzy logic language managing similarities and truth degrees. In *Proc of the 10th Int. Symposium RuleML 2016, LNCS 9718, Springer*, pages 68–82, 2016.

[3] P. Julián-Iranzo, G. Moreno, J. Penabad, and C. Vázquez. A fuzzy logic programming environment for managing similarity and truth degrees. In *Proc. of XIV Jornadas sobre Programación y Lenguajes, PROLE'14*, volume 173 of *EPTCS*, pages 71–86, 2015.

[4] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Similarity-based Unification: a multi-adjoint approach. *Fuzzy Sets and Systems*, 146:43–62, 2004.

*Work supported by the EU (FEDER), and the Spanish MINECO Ministry under grant TIN2013-45732-C4-2-P.

[5] G. Moreno and C. Vázquez. Fuzzy logic programming in action with FLOPER. *Journal of Software Engineering and Applications*, 7:237–298, 2014.

[6] M. I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1-2):389–426, 2002.

[1]Dept. Computing Systems, U. Castilla-La Mancha, 02071, Albacete, SPAIN
Gines.Moreno@uclm.es

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

A Gentle Introduction to Linear Temporal Logic and How To Increase its Expressive Power

César Sánchez[1]

Temporal logic was invented by the philosopher Arthur Prior and developed by Hans Kamp [1] in the 1950s and 1960s to model the reasoning and representation of time. Later, in 1977 Amir Pnueli [2] introduced temporal logics to computer science in 1977 to express behaviors of computational system.

In this talk we will gently introduce linear-temporal logic (LTL), where time is interpreted as a linear infinite sequence of instants. Many results since the 80s and 90s have established connections between temporal logic, automata theory and game theory. For example, it is well known that LTL is strictly less expressive than non-deterministic finite automata on infinite words (known as Büchi automata), which has slowed the adoption of LTL to practical applications like hardware verification. Several adaptations have been proposed to mitigate this lack of expressiveness, like using fix-point calculi or encoding automata in the specification language, but most of these efforts do not preserve the elegance of a simple logic with a finite number of modal operators. At the end of this talk I will present our proposal, regular linear temporal logic (RLTL) [4, 3], to mitigate this lack of expressiveness.

Keywords: Logic, Temporal Logic, Verification, Formal Methods, Reactive Systems, Computer Science

MSC 2010: 68Q60, 68N30, 03B44

References

[1] H. W. KAMP, *Tense logic and the theory of linear order*. Phd thesis, University of California, Los Angeles, 1968.

[2] AMIR PNUELI, *The temporal logic of programs*. Proc. of the 18th Annual Symposium on Foundations of Computer Science (FOCS), 1977, 46–57.

[3] CÉSAR SÁNCHEZ AND MARTIN LEUCKER, *Regular linear temporal logic with past*. Proc. of Verification, Model Checking, and Abstract Interpretation, 295-311, 2010.

[4] MARTIN LEUCKER AND CÉSAR SÁNCHEZ, *Regular linear temporal logic*. Proc. of Theoretical Aspects of Computing–ICTAC 2007, 291-305.

[1]IMDEA Software Institute
Campus de Montegancedo s/n
28223 Pozuelo de Alarcon, Madrid, Spain
cesar.sanchez@imdea.org

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

Analysis of heart rate variability with RHRV

Xosé A. Vila[1]

Our heart beats rhythmically pushing blood throughout our body. Cardiologists use the electrocardiogram (ECG) to check if the rhythm of the heartbeat and its morphology is normal. The heart is not beating like a metronome, in fact analyzing beat-to-beat distances provide relevant informaction for the diagnosis and monitoring of patients. Mathematics and technical advances have allowed in recent years to expand the range of possible uses of this information.

This paper will explain how the heart rate variability (HRV) is obtained, which techniques are used to extract relevant information from a clinical point of view, which problems arises and how mathematics help to solve them.

Currently there are not many software tools available to clinicians able to perform automatic HRV analysis. In our group we have developed a free package for R, which called RHRV, that permitits to obtain the heart rate from the ECG and to obtain information from them.

Keywords: ECG, HRV, digital signal processing

MSC 2010: 68N99, 62P10

References

[1] García, CA and Otero, A and Presedo, J and Vila, X and Félix, P, A software toolkit for nonlinear Heart Rate Variability analysis *Computing in Cardiology*, 393 – 396, IEEE, 2013.

[2] Rodríguez-Liñares, L and Méndez, AJ and Lado, MJ and Olivieri, DN and Vila, XA and Gómez-Conde, I, An open source tool for heart rate variability spectral analysis *Computer methods and programs in biomedicine* **103**(1), 39–50 (2011).

[1]Department of Computer Science
University of Vigo
Campus Universitario As Lagoas s/n. 32004 Ourense.
anton@uvigo.es

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

Model Checking: A Formal Verification Technique with Industrial Applications

María del Mar Gallardo[1]

Nowadays, an increasing number of devices and tools include so-called critical software, that is, software that carries out highly sensitive tasks whose failure is unacceptable for security reasons. Examples of such critical applications may be found in different domains such as automotive [1], health [2], railways [3] or avionics [4]. The analysis/verification of critical systems wrt the most essential properties involves having to deal with their inherent complexity deriving from different sources. For instance, critical code is usually composed of thousands of code lines which makes a non-automatized analysis impossible. In addition, its behaviour is generally non-deterministic as critical software is usually concurrent and/or because the interaction with the environment does not occur in an orderly fashion.

In this communication, we briefly describe the foundations and current applications of model checking for the verification of critical concurrent systems. Model checking [5, 6] is a well-established formal technique characterised by displaying a good balance between mathematical rigor, needed to guarantee software correctness, and the practical applicability which comes from its algorithmic nature. Since its very beginnings, the development of the technique has been carried out in parallel with the construction of tools, the so-called model checkers.

As first glance, the idea behind model checking is simple. Roughly, to analyse a system, we only have to build the whole reachability graph produced by the system, containing all possible system behaviours, and check whether all possible executions satisfy a set of desirable properties. It is clear that the drawback to this apparently brute force technique is the well-known state explosion problem that occurs when the system graph to be analysed is too big. However, despite this non-trivial problem, the advantage of the method with respect to the deductive approaches is that it is completely algorithmic. Observe that the word algorithm in this sentence refers not only to the task of constructing the system graph, but also to the way of proving properties on this graph. This is the strength of the model checking technique and the reason why E.M. Clarke and E.A. Emerson, and J. Sifakis received the ACM Turing Award in 2007 [7]. The two research groups independently found model checking algorithms to verify CTL (computational tree logic) properties on systems [8, 9]. The complexity of these algorithms is polynomial. Specifically, the model checking procedure in [8] ran in a time proportional to the square of the system size (the number of reachable states) and linear to the size of the property analysed. However, these results were clearly improved upon in later papers. These model checking

algorithms were based on the iterative fixpoint calculations of basic temporal modalities. An interesting result from M. Vardi and P. Wolper [10] was the observation that when desirable properties are described in LTL (Linear Temporal Logic), systems to be analysed and properties are formally equivalent under the common notion of automata. Thus, combining software systems and properties is reduced to constructing a product automata, which can be done automatically and efficiently. This new interpretation of model checking made it possible to construct very efficient model checkers such as SPIN [11] developed by G. J. Holzmann who received the ACM Software System Award in 2001.

Over the last 25 years, many model checking algorithms and associated techniques have appeared, most of them trying to palliate the state explosion problem (symbolic model checking, abstract model checking, symmetry reduction algorithms, partial order reduction algorithms and so on). Currently, we can say that although the technique has a physical limitation due to the state space problem, in practice, it can be successfully applied to real systems. For instance, the symbolic model checker NuMSVM has been able to analyse systems with 10^{120} reachable states [12].

Therefore, we can conclude that model checking is a 'push-button' verification technique. Thus, a model checker accepts the specification of a system and a property written in temporal logic and returns 'yes' when all the possible executions of the system satisfy the property, or 'no', when the tool finds a malfunctioning execution. In addition, in the latter case, the tool also returns the erroneous behaviour (the counterexample) which can be used to debug the system. The fact that the model checkers appear as software tools whose inner complexity is hidden from users and that can be used by non-expert programmers has led to the gradual integration of model checking, and formal methods, in general, in the industry. Currently, many software companies such as Lucent Technologies (currently, part of NOKIA), Intel, NASA, Microsoft and IBM have departments dedicated to software analysis using model checking.

Keywords: Critical Software, Verification, Model Checking

MSC 2010: 68-02

References

[1] E. Y. KANG; G. PERROUIN; P. Y. SCHOBBENS, Model-Based Verification of Energy-Aware Real-Time Automotive Systems. In *Proc. of 2013 18th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 135–144. IEEE Computer Society, Washington, DC, USA, 2013.

[2] L. A. TUAN; M. C. ZHENG; Q. T. THO, Modeling and Verification of Safety Critical Systems: A Case Study on Pacemaker. In *Proc. of 2010 Fourth Interna-*

tional Conference on Secure Software Integration and Reliability Improvement (SSIRI'10), 23–32. IEEE Computer Society, Washington, DC, USA, 2010.

[3] J. QIAN; J. LIU; X. CHEN; J. SUN, Modeling and Verification of Zone Controller: The SCADE Experience in China's Railway Systems. In *Proc. of 2015 IEEE/ACM 1st International Workshop on Complex Faults and Failures in Large Software Systems (COUFLESS)*, 48–54. IEEE Press, Piscataway, NJ, USA, 2015.

[4] P. CÁMARA; J. R. CASTRO; M. M. GALLARDO; P. MERINO, Verification support for ARINC-653-based avionics software. *Softw. Test., Verif. Reliab.*, **21**(4),267–298 (2011).

[5] E. M. CLARKE; O. GRUMBERG; D. PELED, *Model Checking*. MIT Press, Cambridge, USA, 1999.

[6] C. BAIER; J.-P. KATOEN, *Principles of Model Checking*. MIT Press, Cambridge, USA, 2008.

[7] E. M. CLARKE ; E. A EMERSON ; J. SIFAKIS, Model checking: Algorithmic verification and debugging. *Communications of the ACM* **52**(11), 74–84 (2009).

[8] E. M. CLARKE; E. A. EMERSON, Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs, Workshop*, 52–71. Springer-Verlag, London, UK, 1982.

[9] J. P. QUIELLE; J. SIFAKIS, Specifcation and verifcation of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on Programming*, 337–350. LNCS 137, Springer Berlin Heidelberg, 1982.

[10] M. Y. VARDI; P. WOLPER, An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Symposium on Logic in Computer Science*, 322–331. IEEE Computer Society, Cambridge MA, 1986.

[11] G. J. HOLZMANN, *The SPIN Model Checker*. Addison-Wesley Professional, 2003.

[12] S. P. MILLER; M. W. WHALEN; D. D. COFER, Software Model Checking Takes Off. *Communications of the ACM* **53**(2), 74–84 (2010).

[1]Dpto. de Lenguajes y Ciencias de la Computación
University of Málaga
Campus Teatinos s/n, 29171 Málaga
gallardo@lcc.uma.es

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

Efficient Computation of Absolutely Normal Numbers

Jack H. Lutz[1], **Elvira Mayordomo**[2]

A real number x is *absolutely normal* if, for every base $b \geq 2$, every two equally long strings of digits appear with equal asymptotic frequency in the base-b expansion of x. We discuss recent progress on efficient algorithms for computing real numbers that are absolutely normal.

Keywords: absolutely normal numbers, finite-state randomness, Lempel-Ziv algorithm, martingale diagonalization, nearly linear time

MSC 2010: 03D32, 68W01, 11-04

References

[1] V. BECHER, S. FIGUEIRA, AND R. PICCHI, Turing's unpublished algorithm for normal numbers.
Theoretical Computer Science, **377**, 126–138, (2007).

[2] E. BOREL, Sur les probabilités dénombrables et leurs applications arithmétiques.
Rendiconti del Circolo Matematico di Palermo, **27**(1), 247–271, (1909).

[1]Department of Computer Science
Iowa State University
Ames, IA 50011 USA
lutz@cs.iastate.edu

[2]Departamento de Informática e Ingeniería de Sistemas, Instituto de Investigación en Ingeniería de Aragón
Universidad de Zaragoza
50018 Zaragoza, SPAIN
elvira@unizar.es

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

On the NP-Hardness of Optimizing Binarizations of Context-Free Grammars

Carlos Gómez-Rodríguez[1]

Binarization, the process of transforming a grammar to an equivalent form where each rule has at most two symbols in its right-hand side, is a key task for a wide range of parsers using context-free grammar and other grammatical formalisms. As non-trivial grammars can be binarized in multiple ways, it is convenient for efficiency reasons to choose a binarization that is as small as possible. This optimization problem has been addressed with heuristics that yield relatively compact binarizations, but no efficient algorithm is known that guarantees to generate a grammar with minimum size. In this talk, I will show that the problem of finding a minimum binarization for a given context-free grammar is NP-hard, by reduction from vertex cover. This result has been published in [1]. The result also generalizes to other, more powerful grammar formalisms.

Keywords: Complexity, binarization, grammars, context-free grammar, parsing, natural language processing

MSC 2010: 68Q17, 68Q25, 68Q42, 68T50

References

[1] C. GÓMEZ-RODRÍGUEZ, Finding the smallest binarization of a CFG is NP-hard. *Journal of Computer and System Sciences* **80**(4), 796–805 (2016).

[1]Departamento de Computación
Universidade da Coruña
Campus de Elviña, s/n, 15071 A Coruña
carlos.gomez@udc.es

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

A refined algorithm for testing the Leibniz n-algebra structure

J. M. Casas[1], **M. A. Insua**[1] **M. Ladra**[2], **S. Ladra**[3]

We present a refinement of the algorithm given in [2] that checks if a multiplication table corresponds to a Leibniz n-algebra structure. This algorithm is based on the computation of a Gröbner basis of the ideal which is used in the construction of the universal enveloping algebra of a Leibniz algebra and it is implemented in a Mathematica notebook by means of the NCAlgebra package.

Essentially, the refinement consists of removing all the superfluous information in the generators of the ideal; this deletion allow us to decrease highly the computation time.

A comparative analysis between both implementations is provided.

Keywords: Leibniz n-algebras, Universal enveloping algebras, Groebner Bases

MSC 2010: 17A32, 68U99

Introduction

A Leibniz n-algebra [3] is a $\mathbb{K}$-vector space $\mathcal{L}$ endowed with an n-linear map $[-,\dots,-]\colon \mathcal{L}^{\otimes n} \to \mathcal{L}$ satisfying the Fundamental Identity

$$\big[[x_1,\dots,x_n],y_1,\dots,y_{n-1}\big] = \sum_{i=1}^{n} \big[x_1,\dots,x_{i-1},[x_i,y_1,\dots,y_{n-1}],x_{i+1},\dots,x_n\big] \tag{FI}$$

for all $x_1,\dots,x_n,y_1,\dots,y_{n-1} \in \mathcal{L}$.

When the n-bracket is skew-symmetric, the structure is named Lie n-algebra. Lie (respectively, Leibniz) 2-algebras are exactly Lie (respectively, Leibniz) algebras.

For finite-dimensional Leibniz n-algebras with basis $\{e_1,\dots,e_d\}$, the n-ary bracket is determined by structure constants $c^k_{i_1,i_2,\dots,i_n}$ such that $[e_{i_1},e_{i_2},\dots,e_{i_n}] = \sum_{k=1}^{d} c^k_{i_1,i_2,\dots,i_n} e_k$.

The problem of identify a Leibniz n-algebra structure in a given n-ary bracket is the subject of the paper [2], where a computer program in Mathematica that checks if a multiplication table satisfies (FI) is implemented. The algorithm is based on the computation of a Gröbner basis of the ideal which appears in the construction of the universal enveloping algebra of a Leibniz algebra [5], by means of the NCAlgebra

package [4] which enables the computation of Gröbner bases in non commutative associative algebras. This Gröbner basis provides a criterion in terms of existence of polynomials of degree 1 over convenient variables, which guarantees that the multiplication table corresponds to a Leibniz n-algebra or not. To decide whether a Leibniz n-algebra is a n-Lie algebra or not, it is necessary to check whether certain type of polynomials are equal to zero.

Although the implementation of this algorithm does not provide efficient results concerning times of computation in an Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz, 16 GB RAM, running Windows 7 (64 bits) with Mathematica®10, our goal in this talk is to present a refinement of the computer program in order to reduce times of computation and compare its efficiency with respect to the initial one given in [2].

Refinement of the algorithm

While we were developing the initial algorithm, it was very clear for us from the beginning [2], that the process could be quicker but the goal was not to get an efficient algorithm, at least at that point of the research. The main motivation was to proof that the Leibniz checking process can be done using Gröbner Basis. Doing things like this, we have enriched the problem and so it is possible to manage the situation from a different and useful point of view (Gröbner Basis Theory and Ideal Theory). Once the theoretical background is established, our interest changed from the existence to the efficiency of the algorithm.

The main idea, we followed to reduce computation time, was to avoid unnecessary computations and to remove all the superfluous information which is contained in the ideal.

The first criterion, we followed to avoid unnecessary computations, is the following: if we examine the proof of [2, Proposition 3.6], it is possible to check that the set of the expressions $[e_i, g_t(e_1, \ldots, e_d)]$ ($g_t \in \mathcal{D}_n(\mathcal{L})^{\mathrm{ann}}$) have an important role. If one of these brackets is not equal to zero, then the structure cannot be a Leibniz n-algebra (as if $\mathcal{L}$ is a Leibniz n-algebra, then $[-, (\mathcal{D}_n(\mathcal{L}))^{\mathrm{ann}}] = 0$).

The second criterion, we followed to remove all the superfluous information, is the following: if all the previous expressions are equal to zero, then it is easy to check, again following the proof of [2, Proposition 3.6], that it is possible to gather the information we need from a subideal of $\Phi(I)$, this subideal is $\langle \{x_i \cdot x_j - x_j \cdot x_i - \Phi(r_{[e_i,e_j]})\}_{i,j\in\{1,\ldots,m\},i<j} \bigcup \{y_i \cdot x_j - x_j \cdot y_i - \Phi(l_{[e_i,e_j]})\}_{i\in\{1,\ldots,d\},j\in\{1,\ldots,m\}} \rangle$, $m = \dim(\mathcal{D}_n(\mathcal{L})^{\mathrm{ann}})$. A natural question arises at this point, is the Gröbner Basis of this subideal finite?, the answer is affirmative because all the reductions drive us to 0 or a polynomials of degree 1.

Using these two criteria and other minor computational aspects, such that, stop the computation process as soon as we obtain the information we need, that is, no more computations will be done when the information is reached, helped us to construct a more efficient algorithm.

References

[1] J. M. CASAS; M. A. INSUA; M. LADRA, Poincaré-Birkhoff-Witt theorem for Leibniz n-algebras. *J. Symbolic Comput.* **volumen** (42 (11–12)), 1052–1065 (2007).

[2] J. M. CASAS; M. A. INSUA; M. LADRA; S. LADRA, Test for Leibniz n-algebra structure. *Linear Algebra Appl.* **volumen** (494), 138–155 (2016).

[3] J. M. CASAS, J.-L. LODAY, T. PIRASHVILI, Leibniz n-algebras, *Forum Math.* **volumen** (14 (2)), 189–207 (2002).

[4] J. W. HELTON, R. L. MILLER, M. STANKUS, NCAlgebra: A Mathematica package for doing noncommuting algebra, `http://math.ucsd.edu/~ncalg` (1996).

[5] M. A. INSUA, M. LADRA, Gröbner bases in universal enveloping algebras of Leibniz algebras, *J. Symbolic Comput.*, **volumen** (44 (5)), 517–526 (2009).

[1]Dpto. Matemática Aplicada I
Univ. de Vigo
36005 Pontevedra, Spain
jmcasas@uvigo.es

[2]Dpto. de Álxebra
Univ. de Santiago de Compostela
15782 Santiago, Spain
manuel.ladra@usc.es

[3]Dpto. de Computación
Univ. de A Coruña
15071 A Coruña, Spain
susana.ladra@udc.es

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

The Geometry of Musical Chords according to D. Tymoczko

Ana Pereira do Vale[1],

Theoretical models for composition and musical analysis have used in recent years, mathematically structured models that gain particular importance in computational issues related to music. These models have not been started recently. The first (tonnetz) two-dimensional latice of sounds known was presented in 1739 by Leonhard Euler ([1]) This latice established a link between the notes that formed musical intervals of major third and perfect fifth. In the mid-nineteenth century the "tonnetz" were rediscovered by some music theorists, namely Arthur von Oettingen (1836-1920) ([2]) and Hugo Riemann (1849-1919) ([3]). Hugo Riemann wrote several books on musical analysis and revolutionized the theory of musical analysis so deeply, that modern theories are called Neo-Riemannian.

In Neo-Riemannian theories there is a great effort to systematize, not only through latices ("tonnetz"), but also through the definition of specific functions examples of these can be found in the works of Allen Forte (1926-2014) ([4]) or David Lewin (1933-2003) ([5]).

One objective of these approaches is to establish a music structure that is global. That could be applied in the musical analysis of different types of music:tonal music, twelve-tone music, etc. Nowadays one uses different systems for each of them.

It should be noted that these models are not intended only for the analysis of musical compositions, they also serve as a system of rules for composition. Indeed the portuguese composer Paulo Bastos who, in his master's thesis examined the "Six piano pieces" Opus 19 by Schönberg using the theory of attractive notes of Edmond Costère (([6]), has recently composed several works using this structure as a support.

Some of these mathematical models developed are geometrical. It is the case of the Theory of Chord Dmitri Tymocko ([7]) and The Geometry of Rhythms ([8]).

It is known by musicians of the 12 notes model description distributing points in a circle or a dodecahedron. Notes are represented in equivalence classes. The octave is not important. All the notes that correspond to C (treble or bass) are represented by the same point and we obtain a representation of the 12 notes as a quotient set. Tymoczko describes a similar mode for musical chords, obtaining subsets of size 2, 3 and 4, as the chords consist of 2, 3 or 4 notes. Choosing an appropriate geometric representation of these spaces, we obtain a spatial organization of the chords we will provide important data for the theory of music. It will be explained in detail the model of two-note chords and explained how it will appear the model of three-note chords.

Keywords: Music Theory, Klein bottle, Voice leading, Tonnetz

MSC 2010: 00A65, 57M60

References

[1] LEONHARD EULER, *Tentamen novae theoriae musicae ex certissismis harmoniae principiis dilucide expositae*, 1739 Editorial, City, year of publication.

[2] ARTHUR VON OETINGEN, *Harmoniesystem in dualer Entwicklung*, Dorpat 1866

[3] HUGO RIEMANN, *Handbuch der Harmonielehre*, Leipzig, 1887

[4] ALLEN FORTE, *The Structure of Atonal Music,*, The Structure of Atonal Music,

[5] DAVID LEWIN, *Generalized Musical Intervals and Transformations*, Oxford University Press 2011

[6] EDMOND COSTÈRE, *Lois et Styles des Harmonies Musicales*, Presses Universitaires de France, 1954

[7] DMITRI TYMOCZKO, *A Geometry of Music*, Oxford University Press 2012

[8] GODFRIED T. TOUSSAINT, *The Geometry of Musical Rhythm*, CRC Press, 2013.

[1]Departamento de Matática
Universidade do Minho
Campus de Gualtar 4710-057 Braga Portugal
avale@math.uminho.pt

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

Multi-armed Bandits for Information Retrieval Evaluation

David E. Losada[1]

Evaluation is crucial to making progress in building better search engines. In the field of Information Retrieval, it is standard practice to build test collections and define evaluation measures to assess the effectiveness of search systems [1]. Each benchmark or test collection comprises a set of queries, a collection of documents and a set of relevance judgements. Relevance judgements are often done by humans and thus expensive to obtain. Consequently, relevance judgements are customarily incomplete. Only a subset of the collection, the pool, is judged for relevance. In popular evaluation campaigns, the pool is formed by the top retrieved documents supplied by systems participating in a certain evaluation task [2]. With multiple retrieval systems contributing to the pool, an exploration/exploitation trade-off arises naturally. Exploiting effective systems could find more relevant documents, but exploring weaker systems might also be valuable for the overall judgement process. In this talk, I will explain our research on Reinforcement Learning [3] for pooling-based evaluation of Information Retrieval systems. Our proposal is based on casting document judging as a multi-armed bandit problem [4]. This formal modelling leads to theoretically grounded adjudication strategies that improve over the state of the art. We show that simple instantiations of multi-armed bandit models are superior to all previous adjudication strategies.

Keywords: Multi-armed bandits, Information Retrieval, Evaluation, Reinforcement Learning

MSC 2010: 68P20, 68T05

References

[1] B. CROFT; D. METZLER; T. STROHMAN, *Search engines: Information Retrieval in Practice*. Pearson, 2010.

[2] M. SANDERSON, Test Collection based Evaluation of Information Retrieval Systems. *Foundations and Trends in Information Retrieval* **4**(4), 247–375 (2010).

[3] R. SUTTON; A. BARTO, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[4] H. ROBBINS, Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.* **58**(5), 527–535 (1952).

[1]Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)
Universidade de Santiago de Compostela
david.losada@usc.es

6th Iberian Mathematical Meeting
Santiago de Compostela, October 6th–8th, 2016

Non-degeneracy conditions in automated proving and discovery

Manuel Ladra[1], **Pilar Páez-Guillán**[1], **Tomás Recio**[2]

The automated theorem proving theory was initiated by Wu in [4], and its goal is to provide computer algebra algorithms in order to decide automatically if a given geometric statement is true. For this purpose, the method assigns coordinates to the points involved in the statement (some independent, say $x_1, \ldots, x_s$, and the others dependent on the previous ones, $x_{s+1}, \ldots, x_n$), and polynomial equations to the rest of the elements (lines, circles...) and conditions (parallelism, perpendicularity...), "translating" in this way the hypotheses (H) and the thesis (T). We symbolise the theorem as $H \Longrightarrow T$, and declare it true if $V(H) \subseteq V(T)$, roughly speaking.

We will be interested in the "non-degenerated" configurations of our theorems; i.e. those in which two different points are really different, a triangle does not collapse to a line, etc. The theorems which are true except on these exceptional cases are called *generally true*. There exist methods to determine successfully whether a theorem is generally true or not, as well as the extra polynomials needed to restrict the set of hypotheses H to the desired situation –they are called *non-degeneracy conditions*. Working over an algebraically closed field K, a theorem is generally true if and only if the ideal $(H + (T \cdot t - 1)) \cap K[x_1, \ldots, x_s]$ (where t is an auxiliary variable) is different from zero; similarly, we say that a theorem is *generally false* when $(H + T) \cap K[x_1, \ldots, x_s] \neq 0$, with the approach and terminology employed in [3]. The polynomials in the last ideal are necessary conditions for the thesis to be true: if we add them to H, and after that we obtain a generally true theorem, it is said that we have *discovered* it.

However, due to the high complexity of the algorithms to compute the previous ideals (triangularization, Gröbner basis, etc.), sometimes it is convenient to introduce manually some easy-to-guess non-degeneracy conditions, and then apply the method. At this point, a problem arises: our method is designed to deal with equations, and the non-degeneracy conditions are inequalities. The traditional way to transform a non-degeneracy condition into an equality is the so-called "Rabinowitsch trick": if we want to express $f \neq 0$, we can write $f \cdot t - 1 = 0$, and add this polynomial to H. Clearly, if $f(x_1, \ldots, x_n) = 0$, we have that $f(x_1, \ldots, x_n) \cdot t - 1 = -1$, and if $f(x_1, \ldots, x_n) \neq 0$, we take $t = 1/f(x_1, \ldots, x_n)$ and we are done. But in the context of algebraically closed fields, there is another option: the saturation $\texttt{Saturate}(H, f) = (H \colon f)^\infty = \{g \in K[x_1, \ldots, x_n] \colon g \cdot f^n \in H \text{ for some } n \in \mathbb{N}_{>0}\}$. This possibility is due to the following geometric interpretation: $V(\texttt{Saturate}(H, f)) = \overline{V(H) \setminus V(f)}$, where $\overline{U}$ denotes the Zariski closure

of the set $U \subseteq K[x_1, \ldots, x_n]$. As it is indicated in [1], both methods are related:

$$\texttt{Saturate}(H, f) = (H + (f \cdot t - 1)) \cap K[x_1, \ldots, x_n]. \tag{1}$$

It is our goal to show how the choice of the method employed for introducing the non-degeneracy conditions can affect our results.

Henceforth, we will write $H_1 \coloneqq H + (f \cdot t - 1)$, $H_2 \coloneqq \texttt{Saturate}(H, f)$, $\mathcal{H}_1 \coloneqq (H_1 + T) \cap K[x_1, \ldots, x_s]$ and $\mathcal{H}_2 \coloneqq (H_2 + T) \cap K[x_1, \ldots, x_s]$. By (1), we see that $H_2 \subseteq H_1$, and then $\mathcal{H}_2 \subseteq \mathcal{H}_1$: the Rabinowitsch trick provides more conditions for discovery than saturation. What is interesting is that $\mathcal{H}_1 = \texttt{Saturate}(\mathcal{H}_2, f)$ (analogously, $(H_1 + (T \cdot t' - 1)) \cap K[x_1, \ldots, x_s] = \texttt{Saturate}((H_2 + (T \cdot t' - 1)) \cap K[x_1, \ldots, x_s], f)$), as it is shown in [2]; so, the choice does not affect whether the theorem is generally true or false.

Anyway, we still have two different sets of additional hypotheses, $\mathcal{H}_1$ and $\mathcal{H}_2$, which we must add to the original set H to find out whether they lead to a new theorem or not. And, once again, we can introduce the non-degeneracy condition $f \neq 0$ by the traditional approach or by saturation. We differentiate four statements:

$$\begin{aligned}
St_1: &\quad H + \mathcal{H}_1 + (f \cdot t - 1) \Longrightarrow T,\\
St_2: &\quad H + \mathcal{H}_2 + (f \cdot t - 1) \Longrightarrow T,\\
St_3: &\quad \texttt{Saturate}(H + \mathcal{H}_1, f) \Longrightarrow T,\\
St_4: &\quad \texttt{Saturate}(H + \mathcal{H}_2, f) \Longrightarrow T,
\end{aligned}$$

being St_1 and St_4 more natural than St_2 and St_3.

Using that $\texttt{Saturate}(\mathcal{H}_1, f) = \texttt{Saturate}(\texttt{Saturate}(\mathcal{H}_2, f), f) = \texttt{Saturate}(\mathcal{H}_2, f)$, it is not difficult to prove that St_3 and St_4 are, in fact, the same statement. So, we infer from the previous results that if one of our statements is generally true, the others will be too. However, they are not all the same: using the Rabinowitsch trick, we end up with two new formally different theorems.

These results can be applied in the practical implementation of the method: if we only want to know the "class of truth" of the original theorem, or determine whether it leads to a discovery or not, it is more efficient the saturation than the Rabinowitsch trick, since the former only involves one saturation, and the latter, two. In fact, sometimes a usual computer cannot achieve this information using the Rabinowitsch trick, while no difficulties are found employing the saturation.

Keywords: Automated discovery, non-degeneracy conditions, Rabinowitsch trick, saturation

MSC 2010: 68T15, 68W30

References

[1] D. A. COX; J. LITTLE; D. O'SHEA, *Ideals, varieties, and algorithms. An introduction to computational algebraic geometry and commutative algebra*. Springer Cham, New York, 2015.

[2] G. DALZOTTO; T. RECIO, On protocols for the automated discovery of theorems in elementary geometry. *J. Automat. Reason* **43**(2), 203–236 (2009).

[3] T. RECIO; M. P. VÉLEZ, Automatic discovery of theorems in elementary geometry. *J. Automat. Reason* **23**(1), 63–82 (1999).

[4] W. T. WU, On the decision problem and the mechanization of theorem-proving in elementary geometry. *Sci. Sinica* **21**(2), 159–172 (1978).

[1]Depto. de Matemáticas
Universidad de Santiago de Compostela
Lope Gómez de Marzoa, s/n. 15782-Santiago de Compostela
manuel.ladra@usc.es, mariadelpilar.paez@rai.usc.es

[2]Depto. de Matemáticas, Estadística y Computación
Universidad de Cantabria
Avda. Los Castros, s/n. 39005-Santander
tomas.recio@unican.es