Twenty years later*

Jean-Pierre Jouannaud**

LIX, École Polytechnique
91400 Palaiseau, France
Email: jouannaud@lix.polytechnique.fr
http://www.lix.polytechnique.fr/Labo/jouannaud

Abstract. The first RTA conference took place in Dijon, in 1985. This year, 2005, it takes place in Nara. Nara and Dijon share a glorious past but can be considered as being "Sleeping Beauties", after the title of a book by the Nobel price novelist yasunari Kawabata.
Is RTA sleeping on its glorious past ? Back in the late 80s, many of us feared that this would soon be the case, that research in rewrite systems was deepening the gap with everyday's computer science practice, and that we should develop rewrite-based powerful provers that would make a difference with the state of art and help address real applications such as software verification.
More than ten years later this has not really happened in the way we thought it would. What has happened is that many research areas, such as programming languages, constraint solving, first-order provers, proof assistants, security theory, and verification have all been fertilized by ideas coming from term rewriting. In return, our field has been renewed by new problems and techniques coming from outside our small community.
I am convinced that this will continue, and that new subject areas will join the journey. There are at least two reasons. To quote a celebrated sentence that I have read in many papers: *Equations are ubiquitous in computer science*. This is the first reason : we all like to use equations for modeling problems. The second is that we have developed extremely powerful, sophisticated tools to reason with equations. Many computer scientists do not know these tools. It is our responsibility to preach for their use by showing all we can do with them.

1 Introduction

My goal is to illustrate several aspects of the contributions of rewriting theory to problems originating from programming and theorem proving, two closely related fields that benefit from a term rewriting perspective. I will concentrate on ordered paramodulation, a very old problem which is still progressing, rule-based programming, tree automata, and proof assistants. I will not refrain quoting my self.

---

* *Twenty years later* is the title of a novel by the french writer Alexandre Dumas.

** Project LogiCal, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

2 Basic ordered resolution and paramodulation

Ordered completion is a generalization of ground completion, because it yields a convergent rewrite system from an arbitrary set of equations, therefore providing with a uniform technique to reason about the word problem in an arbitrary equational theory, which is based on the existence of normal forms. The main difference with the case of ground completion is that the rewrite system may be infinite, and therefore no decidability result can be obtained in general. This is indeed the main strength of this method, that decidability is not required.

Ordered completion was first successfully addressed by Peterson [50], who addressed the case of a finite rewrite ordering, before the general case was solved by Hsiang and Rusinowitch who developed *transfinite semantic trees* for that purpose [31]. The use of transfinite semantic trees was made necessary by the possible use of transfinite orderings in ordered completion.

A more general, earlier framework was elaborated by Robinson and Wos [54], combining resolution and paramodulation: they were the first to propose the replacement of the axioms for equality by specific inference rules in order to reduce the search space. The idea of restricting the set of inferences further by systematically using normal forms generated by a rewrite ordering as it is the case with ordered completion is due to Lankford [42]. Lankford, however, did not have the tools to solve the problem in its full generality. This was done much later in a series of papers pioneered by Bachmair and Ganzinger, who came up with a novel, model theoretic method based on the idea of forcing [3, 2, 4, 5]. It is interesting to notice that Goubault succeeded recently to improve over Bachmair and Ganzinger by using finite semantic trees: the very simple but beautiful idea, which Goubault himself ascribes to Rusinowitch, consists in applying a compactness argument before to construct the finite semantic tree of a given unsatisfiable set of clauses [28].

The last 15 years saw another achievement, with constraints taking over unification in deduction calculi [40]. This is an important phenomenon: after logic programming, constraints are making their way everywhere, in term rewriting theory where the difficult problem of local confluence of order-sorted rewrite rules was reduced to satisfiability of membership constraints [20], in automated deduction seen as a generalization of logic programming, but also in model checking where it allows to go smoothly from finite to some decidable infinite systems [25], or in

functional programming where it yields a more elegant and powerful tool to express type inference algorithms [1]. In particular, constraints have been used very successfully to block inferences that were made inside substitutions inherited from previous inferences [48, 6]. This restriction of deduction calculi is dubbed *basic* after Hullot's pionneering work on *basic narrowing* [35].

Examples of use of ordered completion to *modularity* problems include : modular unification algorithms [13], modular confluence properties [36], and the study of CCC, a calculus of constructions embedding the congruence closure algorithm into the conversion rule [11]. Examples of use of constrained deduction to *decidability* results include: decidability of set constraints [7] and decidability of standard theories [47].

3 Rule-based programming

This topic is probably best examplified with MAUDE, a language developed by José Meseguer and his collaborators at SRI first, and now at the state university of Illinois at Urbana-Champaign [45]. Related efforts were conducted in parallel in France and in Japan, by Claude Kirchner and his group, who developed the language ELAN [41], and by Kokichi Futatsugi and his collaborators, who developed the language CAFE [46]. All three languages owe their origin to the OBJ-family of languages, a project started in California by Joseph Goguen in the early 80s, following his earlier work on Clear [16].

The main novelty of MAUDE was to consider that both functional and concurrent programming could be addressed uniformly by rewriting, depending whether confluence was satisfied or not. Elan goes even further by internalizing rewriting in the so-called ρ-calculus via a specific binding construct generalizing the λ-calculus [8], while Cafe insists on the use of co-algebrsa [23]. The use of rewriting as a functional model was of course well accepted [22], while the use of rewriting for non-functional programming had been advocated before for particular applications, especially unification, and more generally constraint solving [37].

There are even more familiar programming languages that use rule based constructs: this is the case of the Ocaml family of languages, where the case construct bases selection on pattern matching.

Another language based on rewriting is Isabelle, implementing Nipkow's higher-order rewrite systems [44]. Isabelle is targeting applications in which programs operate on data structures with binders, like program

transformations. What makes this work apparently different from a language like MAUDE is that it uses higher-order pattern matching instead of plain matching. But this singularity is not really relevant: MAUDE uses pattern matching modulo associativity and commutativity, and a close look to Nipkow's higher-order rewrite systems shows that the problems are exactly those of rewriting modulo [39].

4 Tree automata

At the first RTA, there was not a single paper using tree automata. There were of course papers in formal language theory using word automata. But no tree automaton. However, there were many informal talks about an almost published paper by David Plaisted, who solved the problem of inductive reducibility [51] [1].

It is easy to see that tree automata are equivalent to OBJ's order-sorted signatures, and they were actually introduced in the 60s in a related context, see also [14]. But they had been almost completely forgotten. In some sense, RTA'85 was their second birth. They were later used in many different context, with the strong push coming from the rewriting community: set constraints [26], higher-order matching [33], strong sequentiality [19], Presbuger arithmetic [12], AC-inductive-reducibility [43], inductive theorem proving [15] and more. The theory of tree-automata and its many applications is studied in depth in [32].

5 Proof assistants

Many will agree with me when saying that Isabelle, Coq and PVS are three among the most important proof assistants. Isabelle is based on Nipkow's higher-order rewriting [44]. PVS is based on Shostak's decision procedure for a combination of convex theories, whose ideas are clearly based on rewriting [52, 53]. Originally based on the Calculus of Constructions [21], then on the Calculus of Inductive Constructions [49], Coq is now rapidly moving towards a heavy use of rewriting, for defining inductive types on the one hand [10], and for specifying the conversion

[1] Actually, the proof was wrong. I had found a counterexample to a *simple lemma* stated without proof, and the whole proof could not be repaired. When Emmanuel Kounalis and myself explained the problem to David Plaisted, he succeeded to found a new, completely different proof at the blackboard in ten minutes. This was really impressing: he understood our counterexample much better than ourselves. This new proof contained a complex argument that was much later understood as a pumping lemma on tree automata with equality tests [17].

rule on the other hand [9]. None of these proof-assistants was available in 1985. At that time, most people in our community believed in the future of first-order provers, rather than higher-order ones. The situation is now reversed: many believe in the superiority of higher-order languages for modeling purposes : first-order provers are often seen as supplementing tactics for higher-order provers. And first-order decidability results are accordingly seen as a particular way to automate the higher-order prover in these cases. An even stronger argument is the existence of the Curry-Howard isomorphism which allows to see intuitionistic logic as a kind of abstract machine for implementing formal proofs. On the other hand, first-order provers have been successful for solving very particular problems such as crypto-attacks, for which a blind search appears adequate [24].

My own perspective on this question is that the coming years will see a new generation of proof assistants, in which (higher-order) rewriting superseeds the lambda calculus. Isabelle is the first prover of this kind, but has lost many of the important features of Curry-Howard based calculi. I anticipate both approaches to merge in the coming years, and some work has been done already [9, 11]. An other merge is coming as well: dependent types are making their way in programming languages [55], while modules and functors have been successfully added to the calculus of inductive constructions [18] as well as a compiler for reductions [29, 30]. This move towards harmony will make its way through in the coming years. I do not see a good reason in the present dichotomy between programming languages and proof assistants.

6 Conclusion

I tried to sketch what important unexpected developments based on rewriting took place in the past. I will try now to give my idea about the future.

First, I think that we need to continue investigating the fundamental properties of term rewriting formats: type preservation, termination and Church-Rosser properties are equally important. Any progress there means a progress with the applications. Another important fundamental question is the relationship between term rewriting and tree automata. We need to investigate these questions in various contexts, first-order, higher-order, and modulo. And we need to continue our work on abstract rewriting, in the light of Huet's work for the first-order case [34], and

what was later done for the modulo [38] and higher-order cases [27]. The higher-order case, especially, needs more work, since the only abstract property investigated there was the finite development theorem.

Second I think we need to continue investigating the efficient implementation techniques of rewriting systems. Much has been done for the first-order case with Maude and Elan, and with the work of Ganzinger's group and of Nieuwenhuis's group with the SATURATE and SPASS systems, but this is not the end of the road. Since type-checking in proof-assistants like Coq relies on rewriting techniques, compilation techniques must be throroughly studied which combine first- and higher-order pattern matching.

Third, I think that we need to understand which other areas of computer science may benefit from our work. A recent interesting example was provided by security protocols: since rewrite rules can be seen as a specification language, security protocols can be modeled by rules. Using this approach, Rusinowitch showed that finding an attack to a cryptographic protocol could be achieved by using narrowing. Comon and others also showed that rewriting was a good tool for modeling security protocols since it allowed to smoothly integrate properties of the cryptographic primitives which were naturally expressed as equations.

Last, but not least, I think that we need to integrate the different existing kinds of rewriting, plain rewriting based on plain pattern matching, rewriting modulo based on plain pattern matching, rewriting modulo based on pattern matching modulo, normalized rewriting, normal rewriting, higher-order rewriting based on plain pattern matching, higher-order rewriting based on higher-order pattern matching, higher-order rewriting based on higher-order pattern matching modulo, into a single coherent framework in order to better understand how to design an abstract machine to implement them all, and make them available to users. This question is of course directly related to my view on the future of proof assistants that I sketched in the previous section. It is also related to the need of an abstract investigation of the fundamental properties of term rewriting formats

Acknowledgments: to all those I met during these 20 years and helped me understand that their problems were more important than mine.

References

1. A. Aiken and E. Wimmers. Type inclusion constraints and type inference. In *Proc. 7th ACM Conference on Functional programming and Computer Archtecture*, pages 31–41, Copenhaguen, Denmark, 1993.
2. Leo Bachmair and Harald Ganzinger. Completion of first-order clause with equality by strict superposition. In *Proc. 2nd Int. Workshop on Conditional and Typed Rewriting Systems, Montreal, LNCS 516*, 1990.
3. Leo Bachmair and Harald Ganzinger. On restrictions of ordered paramodulation with simplification. In *Proc. 10th Int. Conf. on Automated Deduction, Kaiserslautern, LNCS 449*, 1990.
4. Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. Technical Report MPI-I-91-208, Max-Planck-Institut für Informatik, Saarbrücken, September 1991. to appear in Journal of Logic and Computation.
5. Leo Bachmair and Harald Ganzinger. Non-clausal resolution and superposition with selection and redundancy criteria. In *Proc. of the LPAR'92*, 1992. Lecture Notes in Computer Science.
6. Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder. Basic paramodulation and superposition. In Deepak Kapur, editor, *Proc. 11th Int. Conf. on Automated Deduction, Saratoga Springs, NY, LNAI 607*. Springer-Verlag, June 1992.
7. Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Set constraints are the monadic class. In *Proceedings of the Eigth Annual IEEE Symposium on Logic in Computer Science*. IEEE Comp. Soc. Press, 1993.
8. Gilles Barthe, Horatiu Cirstea, Claude Kirchner, and Luigi Liquori. Pure patern type systems. In *Conference Record of the 30th Symposium on Principles of Programming Languages*, New-Orleans, USA, January 2003. ACM.
9. Frédéric Blanqui. Definitions by rewriting in the Calculus of Constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.
10. Frédéric Blanqui. Inductive types in the Calculus of Algebraic Constructions. *Fundamenta Informaticae*, to appear.
11. Frédéric Blanqui, Jean-Pierre Jouannaud, and Pierre-Yves Strub. A calculus of congruent constructions. Technical report, École Polytechnique, 2005. submitted.
12. Alexandre Boudet and Hubert Comon. Diophantine equations, Presburger arithmetic and finite automata. In H. Kirchner, editor, *Proc. Coll. on Trees in Algebra and Programming (CAAP'96)*, Lecture Notes in Computer Science, pages 30–43, 1996.
13. Alexandre Boudet, Jean-Pierre Jouannaud, and Manfred Schmidt-Schauß. Unification in Boolean rings and Abelian groups. *Journal of Symbolic Computation*, 8:449–477, 1989.
14. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. In Michel Bidoit and Max Dauchet, editors, *Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 67–92, Lille, France, April 1997. Springer-Verlag.
15. Adel Bouhoula and Jean-Pierre Jouannaud. Automata-driven automated induction. In *Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 14–25, Warsaw,Poland, June 1997. IEEE Comp. Soc. Press.
16. R. M. Burstall and J. A. Goguen. The semantics of CLEAR, a specification language. In *Proc. Winter School on Abstract Software Specifications, Copenhagen, LNCS 86*, 1979.
17. Anne-Cécile Caron, Jean-Luc Coquidé, and Max Dauchet. Encompassment properties and automata with constraints. In Claude Kirchner, editor, *5th International Conference on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 328–342, Montreal, Canada, June 1993. Springer-Verlag.

18. Jacek Chrzaszcz. Modules in cow are and will be correct. In M. Coppo S. Berardi and F. Damiani, editors, *Proceedings TYPES'03*, volume 3085 of *Lecture Notes in Computer Science*, pages 135–150, Torino, Italy, 2003. Springer-Verlag.
19. Hubert Comon. Sequentiality, second-order monadic logic and tree automata. In Dexter Kozen, editor, *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 508–517, San Diego, CA, June 1995. IEEE Comp. Soc. Press.
20. Hubert Comon and Catherine Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, August 1994.
21. Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76:95–120, February 1988.
22. Nachum Dershowitz. Equations as programming language. In *Proceedings of the Fourth Jerusalem Conference on Information Technology*, pages 114–124, Jerusalem, Israel, May 1984. IEEE Computer Society.
23. Razvan Diaconescu and Kokichi Futatsugi. Cafeobj-report: The language, proof techniques and methodologies for object-oriented algebraic specification. In *AMAST series in Computing*, volume 6. World Scientific, 1998.
24. Michaël Rusinowitch et alii. The aviss security protocols analysis tool – system description. In *Proceedings of Computer-Aided Verification 02*, 2003.
25. Laurent Fribourg and Morcos Veloso Peixoto. Automates concurrents à contraintes. *Technique et Science Informatiques*, 13(6), 1994.
26. Rémy Gilleron, Sophie Tison, and Marc Tommasi. Solving systems of set constraints using tree automata. In *stacs93*, 1993.
27. G. Gonthier, J.-J. Lévy, and P.-A. Mellies. An abstract standardisation theorem. In *Proc. 7th IEEE Symp. on Logic in Computer Science*, Santa Cruz, CA, 1992.
28. Jean Goubault-Larrecq. Résolution ordonnée avec sélection et classes décidables de la logique du premier ordre, 2004. available from the web.
29. Benjamin Gregoire. *Compilation de termes de preuves. Un mariage entre Coq et OCaml*. PhD thesis, École Polytechnique, Palaiseau, France, 2003.
30. Olivier Hermant. A rewriting abstract machine for coq, 2004.
31. Jieh Hsiang and Michaël Rusinowitch. On word problems in equational theories. In Thomas Ottmann, editor, *14th International Colloquium on Automata, Languages and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 54–71, Karlsruhe, Germany, July 1987. Springer-Verlag.
32. Denis Lugiez Hubert Comon, Max Dauchet and Sophie Tison, editors. *Tree Automata techniques and Applications*. http://www.grappa.univ-lille3.fr/tata/, Lille, France, 2002.
33. Hubert Comon and Yann Jürsski. Higher-order matching and tree automata. In M. Nielsen and W. Thomas, editors, *Proc. 11th Computer Science Logic*, volume 1414 of *Lecture Notes in Computer Science*, Aarhus, Denmark, August 1997. Springer-Verlag.
34. Gérard Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, October 1980.
35. J.-M. Hullot. Canonical forms and unification. In W. Bibel and R. Kowalski, editors, *5th International Conference on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, Les Arcs, France, July 1980. Springer-Verlag.
36. Jean-Pierre Jouannaud. Modular associative commutative confluence. Technical report, École Polytechnique, 2005.
37. Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT-Press, 1991.
38. Jean-Pierre Jouannaud and Emmanuel Kounalis. Automatic proofs by induction in equational theories without constructors. In *Logic in Computer Science*, June 1986.

39. Jean-Pierre Jouannaud, Albert Rubio, and Femke Van Raamsdonk. Higher-order rewriting with types and arities. Technical report, École Polytechnique, 2005. submitted.
40. Claude Kirchner, Helene Kirchner, and Michaël Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on automatic deduction.
41. Claude Kirchner and Piere Moreau. Non deterministic computations in elan. In J.L. Fiadeiro, editor, *Proceedings 13th workshop on abstract data types*, volume 1589 of *Lecture Notes in Computer Science*, Lisbon, Portugal, October 1999. Springer-Verlag.
42. Dallas S. Lankford. Canonical inference. Memo ATP-32, University of Texas at Austin, March 1975.
43. D. Lugiez and J.-L. Moysset. Complement problems and tree automata in AC-like theories. In *Proc. Symp. on Theoretical Aspects of Computer Science*, Würzburg, 1993. also available as techincal report CRIN 92-R-175.
44. Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, February 1998.
45. José Meseguer. A logical theory of concurrent objects and its realization in the maude language. To appear in G. Agha, P.Wegner, and A.Yoneezawa (editors), Research Directions in Object-Based Concurrency, 1992.
46. A.T. Nakagawa and K. Futatsugi. An overview of cafe specification environment. In *Proc. of the 1st IEEE International Conference on Formal Engineering Methods*, pages 170–181. IEEE Computer Society Press, 1997.
47. Robert Nieuwenhuis. Basic paramodulation and decidable theories. In Amy Felty, editor, *Eleventh Annual IEEE Symposium on Logic in Computer Science*, New-Brunswick, CA, June 1996. IEEE Comp. Soc. Press.
48. Robert Nieuwenhuis and Albert Rubio. Completion of first-order clauses by basic superposition with ordering constraints. Tech. report, Dept. L.S.I., Univ. Polit. Catalunya, 1991. To appear in Proc. 11th Conf. on Automated Deduction, Saratoga Springs, 1992.
49. Christine Paulin-Mohring. Inductive definitions in the system COQ. In *Typed Lambda Calculi and Applications*, pages 328–345. Springer-Verlag, 1993. LNCS 664.
50. Gerald E. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM Journal on Computing*, 12(1):82–100, February 1983.
51. David A. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65(2-3):182–215, May/June 1985.
52. R. E. Shostak. An efficient decision procedure for arithmetic with function symbols. *J. of the Association for Computing Machinery*, 26(2):351–360, April 1979.
53. R.E. Shostak. Deciding combinations of theories. Technical Report CSL 132, SRI International, February 1982.
54. L. Wos, G. Robinson, D. Carson, and L. Shalla. The concept of demodulation in theorem proving. *Journal of the ACM*, 14:698–709, 1967.
55. Hongwei Xi and Franck Pfenning. Dependent types in practical programming. In *Conference Record of the 21st Symposium on Principles of Programming Languages*, San Antonio, Texas, 1998. ACM.