The Rewrite Engines Competitions: A RECtrospective

Francisco Durán[1] **and Hubert Garavel**[2]

[1] Universidad de Málaga, Málaga, Spain
[2] Univ. Grenoble Alpes, Inria, Cnrs, Grenoble Inp, Lig, F-38000 Grenoble, France
E-mail: `duran@lcc.uma.es`, `hubert.garavel@inria.fr`

Abstract

Term rewriting is a simple, yet expressive model of computation, which finds direct applications in specification and programming languages (many of which embody rewrite rules, pattern matching, and abstract data types), but also indirect applications, e.g., to express the semantics of data types or concurrent processes, to specify program transformations, to perform computer-aided verification, etc. The Rewrite Engines Competition (REC) was created under the aegis of the Workshop on Rewriting Logic and its Applications (WRLA) to serve three main goals: (i) being a forum in which tool developers and potential users of term rewrite engines can share experience; (ii) bringing together the various language features and implementation techniques used for term rewriting; and (iii) comparing the available term rewriting languages and tools in their common features. The present article provides a retrospective overview of the four editions of the Rewrite Engines Competition (2006, 2008, 2010, and 2018) and traces their evolution over time.

Keywords: Abstract data type, algebraic specification, compiler, functional programming, interpreter, object-oriented programming, programming language, specification language, term rewrite engine, term rewrite system

1 Introduction

When searching Google for "rewrite engine", most of the references are about Apache web servers and rewrite engines for URLs. Such engines perform *string rewriting*, which is a particular case of *term rewriting* [1, 3], a very general model of computation based on the repeated application of simplification rules. Despite its simplicity, term rewriting has shown itself a suitable paradigm for expressing fundamental concepts of logics, mathematics, and computer science (e.g., concurrency, communication, interaction, etc.).

Beyond such theoretical aspects, the ideas of term rewriting influenced the design of specification and programming languages, many of which incorporate algebraic terms and rewrite rules. Software implementations of term rewriting have been developed including, of course, rewrite engines, but also a large variety of tools for compiler construction, program transformation, and formal verification by theorem proving or model checking.

In order to evaluate and compare the various rewrite engines available, a software competition named REC (*Rewrite Engines Competition*) was created in 2006. Organized together with WRLA (*Workshop on Rewriting Logic and its Applications*), REC provides a forum for sharing experiences among tool developers and potential users. Four editions of this competition have taken place so far: REC1 (2006), REC2 (2008), REC3 (2010), and REC4 (2018).

The present article, which is part of the TOOLympics project to celebrate the 25th anniversary of the TACAS conference by gathering numerous software competitions at ETAPS, provides a retrospective overview of past editions of the REC competition. Sect. 2 summarizes the developments of the competition; Sect 3 lists all tools that have been assessed, and Sect. 4 presents the collection of benchmarks accumulated during the successive editions; finally, Sect. 5 draws perspectives for future editions of the REC competition.

2 Evolution of REC Competitions

In the mid-2000's, it became manifest that the term-rewriting community was lacking a comparative study of the different rewrite engines available. The following excerpt, quoted from [9], articulates the motivation for such a study:

> "The idea of organizing a rewrite competition arose from noticing various applications of rewriting in different areas and by different categories of researchers, many of them manifesting a genuine and explicit interest in term rewriting. We believe that many of us can benefit from such rewrite engine competitions, provided that they are fair and explicitly state what was tested in each case. For example, users of rewrite engine can more informatively select the right rewrite engine for their particular application. On the other hand, for rewrite engine developers, such events give them ideas on how to improve their tools and what to prioritize, as well as a clearer idea of how their engine compares to others."

It was not clear, however, how to conduct such a study. The abstract and general nature of term rewriting has given birth to a great diversity in software implementations. General-purpose rewrite engines differ in the various forms of rewriting they support (conditional, nondeterministic, context-sensitive, etc.). Many other rewrite engines are specialized for particular problems and embedded into programming languages, theorem provers, environments for compiler construction and program transformation, etc. (see Sect. 3 for examples).

- REC1 [9] faced such doubts about the right approach to follow and decided to focus on efficiency, measured in terms of CPU time and memory use. Only two tools participated in this first edition of the competition, organized together with WRLA 2006. A collection of benchmarks, namely term rewrite systems sorted in four categories (see Sect. 4), was produced. Each benchmark was translated by hand into the input language of each participating tool, and revised by tool developers to make sure this code was optimal for their tools.

- REC2 [13] expanded on the ideas of REC1, with a double goal: (i) broaden the comparison by assessing the efficiency of a larger number of rewrite engines — indeed, five tools participated in REC2; and (ii) being a showcase for the term-rewriting community, with a dedicated session at WRLA 2008, where all participating tools were presented by their developers, who exposed the features and strengths of each tool and discussed the outcomes of the competition. Tool developers actively participated in the whole process of REC2, not merely for adapting competition benchmarks to the tools, but also for exchanging views on how to organize the competition and present its results. As a result of fruitful discussions, several changes were implemented, such as the design of a common language for expressing the benchmarks (see Sect. 4 below).

- REC3 [12] followed the same approach as REC2, with a greater emphasis on automation and a larger set of term-rewriting benchmarks — including problems related to program transformation, a key application area of term rewriting. The developers of all the participating tools were involved in this competition, organized together with WRLA 2010. The reported results indicate the computation time spent by each tool on each benchmark.

- REC4 [15] was the result of a long-term effort undertaken in 2015 and presented at WRLA 2018. The competition's scope was broadened away from traditional rewrite engines to include functional and object-oriented languages. As a consequence, REC4 did not consider particular features implemented only in some tools, but focused instead on basic features common to all tools, namely term rewrite systems that are confluent and terminating, with free constructors and conditional rules. Tool execution and comparison of results was fully automated, making it unnecessary to include tool developers directly in the competition — although they were contacted by email, in case of problems, before the presentation of the results. A Top-5 podium was produced to indicate which tools can tackle the most problems within a given amount of time and memory.

3 Tools Assessed

So far, not fewer than 18 tools have been assessed during the REC competitions, as shown by Table 1. This table lists which tools participated in which editions of the competition. Not all tools have been assessed in all editions, as it happened, e.g., for prominent tools such as ELAN [4] and ASF+SDF [26], the development of which halted before or just after REC1.

language (tool)	web site	REC1	REC2	REC3	REC4
ASF+SDF [26]	`http://www.meta-environment.org`	×	×	×	
CafeOBJ [10]	`http://cafeobj.org`				×
Clean [24]	`http://clean.cs.ru.nl`				×
Haskell (GHC) [20]	`http://www.haskell.org`				×
LNT (CADP) [6, 14]	`http://cadp.inria.fr`				×
LOTOS (CADP) [17, 14]	`http://cadp.inria.fr`				×
Maude [7]	`http://maude.cs.illinois.edu`	×	×	×	×
mCRL2 [16]	`http://www.mcrl2.org`				×
OCaml [19]	`http://www.ocaml.org`				×
Opal (OCS) [23]	`http://github.com/TU-Berlin/opal`				×
Rascal [25]	`http://www.rascal-mpl.org`				×
Scala [22]	`http://www.scala-lang.org`				×
SML (MLton) [21]	`http://www.mlton.org`				×
SML (SML/NJ) [21]	`http://www.smlnj.org`				×
Stratego/XT [5]	`http://www.metaborg.org`		×	×	×
TermWare [11]	`http://gradsoft.ua/index_eng.html`		×		
Tom [2]	`http://tom.loria.fr`		×	×	×
TXL [8]	`http://txl.ca`			×	

Table 1: Languages and tools considered in the Rewrite Engines Competitions

It is worth pointing out the versatility of term rewriting and the diversity of its implementations. It is used in both specification and programming languages. These languages can be algebraic (e.g., CafeOBJ, LOTOS, Maude, mCRL2, Stratego/XT, etc.), functional (e.g., Clean, Haskell, LNT, OCaml, SML, etc.), or object-oriented (e.g., Rascal, Scala, Tom, etc.), and certain languages combine several of these traits, such as Opal, which is both algebraic and functional, or OCaml, which is both functional and object-oriented. Some languages also support higher-order programming (e.g., Haskell, OCaml), while others have built-in support for concurrency (e.g., LOTOS, LNT, Maude, mCRL2, etc.). Implementations encompass compilers and interpreters, certain languages (e.g., OCaml or Rascal) offering both, while other approaches (e.g., Tom) enable term rewrite systems to be embedded in general-purpose languages such as C or Java. Finally, some implementations (ASF/SDF, Stratego/XT, etc.) provide rich environments for language design, including support for lexical/syntactic analysis, construction and traversal of abstract syntax trees, as well as program transformations.

4 REC Benchmarks

As a byproduct of the efforts made in organizing the four REC competitions, a collection of benchmarks has been progressively accumulated[1]:

- REC1 [9] set up the foundations of this collection, by gathering 41 term rewrite systems, split into four distinct categories: *unconditional term rewrite systems* (in which no rewrite rule has Boolean premises), *conditional term rewrite systems* (in which some rewrite rules have Boolean premises), *rewriting modulo equations* (in which rewriting relies on certain axioms, such as commutativity and/or associativity), and *rewriting modulo strategies* (in which rewriting is context sensitive, guided by local strategies).

  Several REC1 benchmarks were derived from generic benchmarks parameterized by variables (e.g., the parameter of function computing the factorial of a natural number, the length of a list to be sorted, etc.) by giving particular values to these variables. Following the terminology used for the Model Checking Contest [18], we distinguish between *models*, which are generic benchmarks, and *instances*, which are benchmarks derived from generic benchmarks by giving actual values to parameters; the remaining benchmarks, which are not parameterized, are counted both as models and instances.

- REC2 [13] brought a significant evolution: in REC1, each benchmark was specified in the input language of each tool, which was only feasible as the number of tools was small. REC2 introduced, to express its benchmarks, a common language, which we name REC-2008 and which was inspired by the TPDB language used at that time by the Termination Competition (the Confluence Competition uses a similar language). Several tools were adapted to accept this new language REC-2008 as input; for the other tools, translation was done manually.

- REC3 [12] pursued in the same vein as REC2, while increasing the number of instances. REC3 also tried to expand the scope of the competition by introducing a separate collection of benchmarks meant for program transformation and expressed in an imperative language named TIL; however, this initiative was left with no follow-through.

- REC4 [15], in order to address a larger set of specification and programming languages, introduced a new language REC-2017 derived from REC-2008 with additional restrictions ensuring that benchmarks are deterministic (hence, confluent), terminating, and free from equations between constructors. Consequently, the 3rd and 4th categories (rewriting modulo equations and rewriting modulo strategies) were removed, and the 1st and 2nd categories (unconditional and conditional rewriting) were merged into a single

[1] These are available from `http://rec.gforge.inria.fr`.

one, as most languages do not make such a distinction. The remaining REC-2008 benchmarks were upgraded to the REC-2017 language, and many new, significantly complex benchmarks were added to the collection. To provide for an objective comparison, scripts were developed to translate REC-2017 specifications to the input languages of all tools under assessment.

Table 2 gives a quantitative overview of the evolution of the REC benchmark collection; each cells having the form "$(m)\ n$" denotes m models and n instances.

category	REC1	REC2	REC3	REC4
source language	tool-specific	REC-2008	REC-2008	REC-2017
unconditional term rewrite systems	(5) 7	(5) 12	(7) 26	(19) 43
conditional term rewrite systems	(9) 25	(8) 18	(6) 17	(24) 42
rewriting modulo equations	(4) 9	(4) 6	(4) 6	(0) 0
rewriting modulo strategies	(0) 0	(1) 1	(1) 3	(0) 0
TOTAL	(18) 41	(18) 37	(18) 52	(43) 85

Table 2: Benchmarks considered in the Rewrite Engines Competitions

5 Conclusion

Term rewriting is a fundamental topic with many applications, as illustrated by the multiplicity of term-rewriting implementations in compilers and interpreters.

The Rewrite Engines Competitions (REC), the evolutions of which have been reviewed in the present article, stimulate the research interest in this field. One main lesson to be retained from these competitions is that performance of term rewriting significantly differs across implementations: there is room for enhancements and, following the latest REC competition (2018), three developer teams already reported plans to improve their tools to take into account the REC results.

Future REC competitions should address at least two points: (i) more languages should be assessed, inviting recent tools in the competition and keeping in mind that some tools may disappear if they are no longer maintained; (ii) more benchmarks should be considered, which will require dedicated effort to develop new benchmarks, given the lack of large, computationally intensive term rewriting systems freely available on the Web, and the subtle semantic differences that exist between the various flavours of term rewrite systems.

References

[1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[2] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking Rewriting on Java. In Franz Baader, editor, *Proceedings of the 18th International Conference on Term Rewriting and Applications (RTA'07), Paris, France*, volume 4533 of *Lecture Notes in Computer Science*, pages 36–47. Springer, June 2007.

[3] Marc Bezem, Jan Willem Klop, Roel de Vrijer, and Terese (group), editors. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[4] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. *Electronic Notes in Theoretical Computer Science*, 15:55–70, 1998.

[5] Martin Bravenboer, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Visser. Stratego/XT 0.17 – A Language and Toolset for Program Transformation. *Science of Computer Programming*, 72(1–2):52–70, 2008.

[6] David Champelovier, Xavier Clerc, Hubert Garavel, Yves Guerte, Christine McKinty, Vincent Powazny, Frédéric Lang, Wendelin Serwe, and Gideon Smeding. Reference Manual of the LNT to LOTOS Translator (Version 6.6). INRIA, Grenoble, France, February 2017.

[7] Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. *Maude Manual (Version 2.7.1)*, July 2016.

[8] James R. Cordy. The TXL Source Transformation Language. *Science of Computer Programming*, 61(3):190–210, 2006.

[9] Grit Denker, Carolyn L. Talcott, Grigore Rosu, Mark van den Brand, Steven Eker, and Traian-Florin Serbanuta. Rewriting Logic Systems. *Electronic Notes in Theoretical Computer Science*, 176(4):233–247, 2007.

[10] Razvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report – The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.

[11] Anatoly E. Doroshenko and Ruslan Shevchenko. A Rewriting Framework for Rule-Based Programming Dynamic Applications. *Fundamenta Informaticae*, 72(1–3):95–108, 2006.

[12] Francisco Durán, Manuel Roldán, Jean-Christophe Bach, Emilie Balland, Mark van den Brand, James R. Cordy, Steven Eker, Luc Engelen, Maartje de Jonge, Karl Trygve Kalleberg, Lennart C. L. Kats, Pierre-Etienne Moreau, and Eelco Visser. The Third Rewrite Engines Competition. In Peter Csaba Ölveczky, editor, *Proceedings of the 8th International Workshop on Rewriting Logic and Its Applications (WRLA'10), Paphos, Cyprus*, volume 6381 of *Lecture Notes in Computer Science*, pages 243–261. Springer, 2010.

[13] Francisco Durán, Manuel Roldán, Emilie Balland, Mark van den Brand, Steven Eker, Karl Trygve Kalleberg, Lennart C. L. Kats, Pierre-Etienne Moreau, Ruslan Schevchenko, and Eelco Visser. The Second Rewrite Engines Competition. *Electronic Notes in Theoretical Computer Science*, 238(3):281–291, 2009.

[14] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 15(2):89–107, April 2013.

[15] Hubert Garavel, Mohammad-Ali Tabikh, and Imad-Seddik Arrada. Benchmarking Implementations of Term Rewriting and Pattern Matching in Algebraic, Functional, and Object-Oriented Languages – The 4th Rewrite Engines Competition. In Vlad Rusu, editor, *Proceedings of the 12th International Workshop on Rewriting Logic and its Applications (WRLA'18), Thessaloniki, Greece*, volume 11152 of *Lecture Notes in Computer Science*, pages 1–25. Springer, April 2018.

[16] Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. The MIT Press, 2014.

[17] ISO/IEC. LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Geneva, September 1989.

[18] Fabrice Kordon, Hubert Garavel, Lom Messan Hillah, Emmanuel Paviot-Adet, Loïg Jezequel, César Rodríguez, and Francis Hulin-Hubard. MCC'2015 – The Fifth Model Checking Contest. *Transactions on Petri Nets and Other Models of Concurrency*, XI:262–273, 2016.

[19] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. *The OCaml System Release 4.04 – Documentation and User's Manual*. INRIA, Paris, France, March 2016.

[20] Simon Marlow, editor. *Haskell 2010 Language Report*, April 2010.

[21] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *Definition of Standard ML (Revised)*. MIT Press, May 1997.

[22] Martin Odersky, Philippe Altherr, Vincent Cremet, Gilles Dubochet, Burak Emir, Philipp Haller, Stéphane Micheloud, Nikolay Mihaylov, Adriaan Moors, Lukas Rytz, Michel Schinz, Erik Stenman, and Matthias Zenger. *The Scala Language Specification – Version 2.11.* Programming Methods Laboratory, EPFL, Switzerland, March 2016.

[23] Peter Pepper and Florian Lorenzen, editors. *The Programming Language Opal – 6th Corrected Edition.* Department of Software Engineering and Theoretical Computer Science, Technische Universität Berlin, Germany, October 2012.

[24] Rinus Plasmeijer, Marko van Eekelen, and John van Groningen. *Clean Version 2.2 Language Report.* Department of Software Technology, University of Nijmegen, The Netherlands, December 2011.

[25] Jeroen van den Bos, Mark Hills, Paul Klint, Tijs van der Storm, and Jurgen J. Vinju. Rascal: From Algebraic Specification to Meta-Programming. In Francisco Durán and Vlad Rusu, editors, *Proceedings of the 2nd International Workshop on Algebraic Methods in Model-based Software Engineering (AMMSE'11), Zurich, Switzerland*, volume 56 of *Electronic Proceedings in Theoretical Computer Science*, pages 15–32, June 2011.

[26] Mark van den Brand, Jan Heering, Paul Klint, and Pieter A. Olivier. Compiling Language Definitions: The ASF+SDF Compiler. *ACM Transactions on Programming Languages and Systems*, 24(4):334–368, 2002.