Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 238 (2009) 281–291
www.elsevier.com/locate/entcs

The Second Rewrite Engines Competition

Francisco Durán,[a] Manuel Roldán,[a] Emilie Balland,[f]
Mark van den Brand,[b] Steven Eker,[c] Karl Trygve Kalleberg,[d]
Lennart C. L. Kats,[e] Pierre-Etienne Moreau,[f]
Ruslan Schevchenko,[g] and Eelco Visser[e]

[a] *Dpt. de Lenguajes y Ciencias de la Computación*
Universidad de Málaga, Málaga, Spain

[b] *Department of Mathematics and Computer Science*
Eindhoven University of Technology, Eindhoven, The Netherlands

[c] *Computer Science Laboratory*
SRI International, Menlo Park, CA, USA

[d] *University of Bergen, Bergen, Norway*

[e] *Department of Software Technology*
Delft University of Technology, The Netherlands

[f] *Centre de recherche INRIA Nancy - Grand Est*
Villers-lès-Nancy Cedex, France

[g] *Institute of Software Systems*
National Academy of Sciences of Ukraine, Kiev, Ukraine

Abstract

The Second Rewrite Engines Competition (REC) was celebrated as part of the 7th Workshop on Rewriting Logic and its Applications (WRLA 2008). In this edition of the competition participated five systems, namely ASF+SDF, Maude, Stratego/XT, TermWare, and Tom. We explain here how the competition was organized and conducted, and present its main results and conclusions.

Keywords: Rewriting Systems, Rewriting Competition, ASF+SDF, Maude, Stratego/XT, TermWare, Tom

1 Introduction

Term rewriting is a powerful programming paradigm, with applications in many different areas, including functional programming, computer algebra, symbolic computation, theorem proving, etc. It has been shown to be specially useful in the definition of programming languages, giving semantics to them, and in the generation, analysis, and transformation of programs.

These application areas are quite different, and many different researchers have been working on them following different approaches, and making different uses

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2009.05.025

of term rewriting engines. Thus, we may find different formalisms, languages, and engines giving support and implementing term rewriting. The First Rewrite Engines Competition [5] was organized with the belief that both rewrite engines users and developers would benefit of such a competition. Users could make a better selection of which engine to use for each particular application, and developers would know how their tool compare to others, would find new challenges, and would gain a better understanding of their 'competitors'. Thus, the whole community would benefit from it.

The main goal of the First Rewrite Engines Competition was to explore the viability of such competition, and to evaluate the interest from the community in it. They started rolling the ball, and given the interest shown we decided to go on. In the second edition we have gathered more systems and a bigger set of problems. We will present in this paper its main results, some conclusions and future challenges.

The first competition focused on efficiency, specifically speed, memory management and built-ins use. There were only two participants, ASF+SDF, represented by Mark van den Brand, and Maude, represented by Steven Eker. For the competition, a number of test examples were compiled, all of them using features supported by both systems. The examples were written in a mathematical and intuitive notation, and then translated, by hand, by an independent researcher (the developers helped and revised the specifications to make sure that the best code was written for each of the experiments).

The Second Rewrite Engines Competition was celebrated in 2008, right before the 7th International Workshop on Rewriting Logic and its Applications (WRLA 2008), where its results were presented. The competition was organized by Francisco Durán, who invited many of the existing rewriting engines developers to participate. Five of them accepted the invitation: ASF+SDF [8,7], represented by Mark van den Brand; Maude [3,4], represented by Francisco Durán and Steven Eker; Stratego/XT [9,2], represented by Lennart Kats, Karl Trygve Kalleberg and Eelco Visser; TermWare [6], represented by Ruslan Shevchenko; and Tom [1], represented by Pierre-Etienne Moreau and Emilie Balland.

2 On the organization

Once we had settled on the selection of engines, we opened a discussion on the way the competition should take place. As in the first edition, the participating systems were hard to compare. We had compilers and interpreters, we had general purpose engines and others more specific, ... ASF+SDF is very good at parsing; ASF+SDF, Stratego, and Tom are very good at program transformation; Maude is good at rewriting modulo and has a powerful formal environment; Tom is an extension of Java, and TermWare is a rule processing engine intended for embedding into Java applications; etc.

In this discussion, some interesting ideas were proposed. We nailed down some of them, others will have to wait for future editions. With such a wide range of systems it did not make sense to go for the intersection of the capabilities of the

systems. Instead, we decided to go for a wider range of problems, and split these in several categories. Each of the systems would participate in those categories it supported.

We selected a bunch of tiny and small problems, organized in four categories (unconditional rewriting, conditional rewriting, rewriting modulo, and context-sensitive rewriting/rewriting with local strategies). The various categories include many classical rewriting problems for their class of rewriting systems, e.g.: the *trs* category includes computation of fibonacci and factorial numbers; the *crts* category includes Towers of Hanoi and merge/quick/bubblesort; the *modulo* category includes specifications of a 3-value logic and a permutations calculation; and the *cs* category includes the Sieve of Eratosthenes. Most of these tests were used in the first competition.

One of our goals was to minimize the effort required in participating in the competition. We were all busy people, and we knew that asking for a big effort was a bad idea. And looking for a graduate student to do the coding, as in the first competition, would not work either. Writing all the tests in five different systems was too much to ask. Everything would be much simpler if we had a common language supported by all the systems. But this was not the case.

One suggested approach was to use non-trivial examples. Small programs or even complex problems, like a small theorem prover, an exploration of a search space, a transformation of XML (or a tree), etc. It was not clear though whether this would evaluate the systems or the application developers. Time was also a major concern.

Given the capabilities of some of the engines involved in the definition of languages and transformations, we came up with the idea to define a very simple rewriting language, which we called REC, in which all the problems were written. Then, as an additional, mid-size problem in the competition, we proposed writing programs transforming the problems in this REC syntax to the syntax of the corresponding tools. Once we have this program running, handling the other small programs should be simple. For those systems in which this was not easily doable, we still have the possibility of writing scripts or programs in other languages to get the codes to execute. Of course, we always have the option of doing it by hand. In fact, we wanted to have the option of providing alternative versions of the specifications for those cases in which an optimized version was possible. The REC language and the problems proposed are relatively simple, and do not assume any built-in or other features that could improve the specifications, like fancy syntactic facilities, memoization, default rules for handling untreated cases, etc. An optimized version of the problems, using any advanced feature provided by a system, could additionally be provided.

Thus, the competition was set up as follows:

- The problem specifications and the tests to be evaluated are specified using the REC language.
- Each specification is in a separate file, with the tests to be run on it after it.

- For each problem, we expect two resulting programs, one generated automatically and one optimized by hand.
- There should be some way of running all the tests and obtaining all the results in a file. Each of the systems would provide instructions on how to run them.

As one can see, these rules are quite vague, and there is a lot of processing and result gathering work to be done. Automation of these task is something to improve in future editions of the competition, but given the differences between the systems and the restrictions in some of them, we decided to do it like this.

The five systems were installed in a 1GHz/1GB linux machine. All the tests were executed and the results were collected. See Section 4 for details on the results. The installation of the systems was done by M. Roldán, who also ran most of the tests. P.-E. Moreau provided an inestimable help in a key moment.

3 The REC language

A very simple rewriting language, REC, was defined as a common language in which to write the problems, and tests on them. Figure 1 includes a BNF description of the syntax of the language. It is many-sorted, does not have any built-ins, uses prefix syntax, does not support overloading, allows conditional rules, and includes syntax for `assoc`, `comm`, `id`, and `strat` attributes *a la* OBJ.

Figure 2 shows a specification of the bubble sort algorithm in the REC syntax together with three rewriting commands on it. Notice that the list of the second command has been abbreviated for space reasons.

Each of the participants was asked to build a program transforming the problems in this REC syntax to the language of their corresponding tools. However, not all of them were able to make it. Only the Maude, Stratego/XT and Tom representatives provided the translators for their systems. The lack of time was with no doubt responsible for not having translators for the others. We are sure that it can be done, and that with some more time it would have been. Next time perhaps.

We thought that by comparing the translators provided, we could draw conclusions about their complexity, development time, and efficiency doing the transformations. However, the approaches followed in Maude, Stratego/XT and Tom to implement the REC translators were very different. While in Maude a programming environment was built, able to read REC programs and commands and give outputs, Stratego/XT and Tom representatives built programs that transformed the original programs and commands, and were later loaded and executed. As such, we were not able to draw such conclusions from the translators themselves: the translators were implemented by experts in each of the systems, in separated locations, and without a clear previous criterion. However, we must say that the facilities provided by the three systems, for this kind of applications, is quite good, and that the development times were small. Regarding parsing, we must say that ASF+SDF and Stratego/XT are very good at defining syntax of languages, and Stratego/XT did a very good job in the competition. Tom and Maude also, although they presented a few limitations at the lexical level:

```
⟨spec⟩          ::= REC-SPEC ⟨id⟩
                      [ SORTS ⟨idlist⟩ ]
                      [ VARS ⟨vardecllist⟩ ]
                      [ OPS ⟨opdecllist⟩ ]
                      [ RULES ⟨rulelist⟩ ]
                    END-SPEC
⟨idlist⟩        ::= ⟨id⟩ ⟨idlist⟩ | ϵ
⟨vardecllist⟩   ::= ⟨idlist⟩ : ⟨id⟩ ⟨vardecllist⟩ | ϵ
⟨opdecllist⟩    ::= ⟨opdecl⟩ ⟨opdecllist⟩ | ϵ
⟨opdecl⟩        ::= op ⟨id⟩ : ⟨idlist⟩ -> ⟨id⟩
                    | op ⟨id⟩ : ⟨idlist⟩ -> ⟨id⟩  ⟨opattrlist⟩
⟨opattrlist⟩    ::= ⟨opattr⟩ ⟨opattrlist⟩ | ϵ
⟨opattr⟩        ::= assoc | comm | id( ⟨term⟩ ) | strat( ⟨intlist⟩ )
⟨rulelist⟩      ::= ⟨rule⟩ ⟨ruleslist⟩ | ϵ
⟨rule⟩          ::= ⟨term⟩ -> ⟨term⟩
                    | ⟨term⟩ -> ⟨term⟩ if ⟨condlist⟩
⟨condlist⟩      ::= ⟨cond⟩ | ⟨cond⟩ , ⟨condlist⟩
⟨cond⟩          ::= ⟨term⟩ -><- ⟨term⟩                    % ==
                    | ⟨term⟩ ->/<- ⟨term⟩                  % =/=
⟨term⟩          ::= ⟨id⟩ | ⟨id⟩ ( ) | ⟨id⟩ ( ⟨termlist⟩ )
⟨termlist⟩      ::= ⟨term⟩ | ⟨term⟩ , ⟨termlist⟩
⟨intlist⟩       ::= ⟨int⟩ ⟨intlist⟩ | ϵ

⟨command⟩       ::= get normal form for: ⟨term⟩
                    | check the confluence of: ⟨term⟩ -><- ⟨term⟩
```

⟨*id*⟩ are non-empty sequences of any characters except ' ', '(', ')', '{', '}', '"' and ',' ; and excluding ':', '->', '-><-', '->/<-', 'if', and keywords REC-SPEC, SORTS, VARS, OPS, RULES, and END-SPEC.
⟨*int*⟩ are non-empty sequences of digits.
Comments are given using '%'. All the text written in the a line after a '%' is discarded.

Fig. 1. The REC language.

- Tom cannot handle symbols like &, +, ...
- For Maude, REC modules and commands must be enclosed in parentheses, and comments must be given as in Maude.

4 The results from the competition

We present a selection of the results in this paper, and refer to the web site of the Second Rewrite Engines Competition, at http://maude.lcc.uma.es/REC, for further details. All the files and results of the competition are available in this web site, where one can find a table that includes, for each of the problems, the specification and the tests run on it in REC syntax, and the corresponding problems in the syntax of each the participant systems, both in the automatically generated and manually generated/optimized versions, together with the times consumed in their computation and the solutions given.

Tables 1, 2, 3, and 4 show the times (in milliseconds) consumed by the five systems in two of the four categories, namely unconditional term rewrite systems (TRS) and conditional TRS (CTRS). Since Maude was the only system in the competition supporting rewriting modulo axioms and local strategies, the results for these categories are not included here. They are available at the competition's web site.

```
REC-SPEC BubbleSort
SORTS
  Bool Nat NatList
OPS
  true : -> Bool
  false : -> Bool
  0 : -> Nat                                          % zero
  s : Nat -> Nat                                      % succesor
  lt : Nat Nat -> Bool                                % less than
  nil : -> NatList
  cons : Nat NatList -> NatList
  plus : Nat Nat -> Nat
  times : Nat Nat -> Nat
  rev : Nat -> NatList
  fact : Nat -> Nat
VARS
  N M : Nat
  L : NatList
RULES
  lt(0, s(N)) -> true
  lt(s(N), 0) -> false
  lt(N, N) -> false
  lt(s(N), s(M)) -> lt(N, M)
  plus(0, N) -> N
  plus(s(N), M) -> s(plus(N, M))
  times(0, N) -> 0
  times(s(N), M) -> plus(M, times(N, M))
  fact(0) -> s(0)
  fact(s(N)) -> times(s(N), fact(N))
  rev(s(N)) -> cons(s(N), rev(N))
  rev(0) -> cons(0, nil)
  cons(N, cons(M, L)) -> cons(M, cons(N, L))
    if lt(M, N) -><- true
END-SPEC

get normal form for:
  rev(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))))))))))

get normal form for:
  cons(s(0), cons(s(s(0)), cons(s(s(s(0))), ... a huge list ...)))

get normal form for:
  rev(fact(s(s(s(s(0))))))
```

Fig. 2. REC specification of the bubblesort algorithm and some rewriting commands on it.

Tables 1 and 3 show the results for the REC problems automatically transformed by each of the tools. As said above, ASF+SDF and TermWare did not provide translators, and therefore they do not appear in these tables. Tables 2 and 4 show the results for the manually generated versions. ASF+SDF did not provide solutions for all the problems; the missing specifications are missionaries, quicksort, and mergesort, for which a '—' is given. These specifications where provided after the competition, but due to technical problems they could not be included in this edition.

Although the Stratego/XT representatives did not optimize their specifications, the results obtained for the automatically generated ones are included in Tables 2 and 4; one may expect that these numbers could be improved in some cases.

In those cases in which the computation terminated abruptly or did not terminate in a reasonable amount of time (15 minutes) the place in the table shows a '—'. In particular, none of the systems were able to complete the second factorial test in the amounted time.

One should take the specificity of the five systems into account when evaluating the results in these tables. Between the competing systems we can find compilers and interpreters, traditional rewriting systems and Java-embedded rewriting sys-

		Maude	Stratego/XT	Tom
ASF+SDF benchmark	test 1	7	0	177
	test 2	5402	2450	3253
	test 3	7	0	231
	test 4	7109	3330	3400
	test 5	11	651320	—
	test 6	17609	—	—
factorial	test 1	1	—	—
	test 2	—	—	—
fibonacci	test 1	1	0	17
garbage collection	test 1	0	—	19
	test 2	0	0	0
reverse	test 1	1	770	27

Table 1
Times (in milliseconds) for the unconditional term rewriting systems, automatically generated from the REC specifications.

tems, general-purpose systems and specific systems, etc. It is commonly accepted that the performance difference between a compiler and an interpreter may be in an order of magnitude, but it is not easy to measure the other circumstances. Moreover, although we have tried to consider problems not handled by all the systems, so that some of the capabilities not in the intersection could be shown, much more needs to be done. We have shown some of the capabilities for defining programming languages, and for transforming their programs. We cannot give any conclusion on this other than what has already been said in Section 3.

In [5], some remarks were given by S. Eker and M. van den Brand explaining the results of Maude and ASF+SDF in the first engines competition. Taking into consideration the increased scope of the current edition of the competition, and the heterogeneity of the systems involved, we cannot offer an in-depth analysis of all the results. Instead, we limit ourselves to highlight some of the them.

The ASF+SDF benchmark (Tables 1 and 2) is a benchmark proposed in [7] to study resource usage for brute-force rewriting (no built-ins, no strategies). We can see how ASF+SDF outperforms all the other systems, and that the compilers of Stratego/XT and Tom also run much faster that the interpreters.

Notice however that the ASF+SDF benchmark was designed specifically for the ASF+SDF system, what is a bit unfair for the other systems. Stratego, for example, focuses on user-defined strategies to control rewrite rules. In the programs generated for the benchmarks, a fixed evaluation strategy is used instead, which is

		ASF+SDF	Maude	Stratego/XT	TermWare	Tom
ASF+SDF benchmark	test 1	40	7	0	102	108
	test 2	430	5402	2450	86712	3236
	test 3	0	7	0	127	17
	test 4	420	7109	3330	178269	3471
	test 5	30	1	651320	154	16
	test 6	490	17609	—	150614	3795
factorial	test 1	0	0	—	—	—
	test 2	—	0	—	—	—
fibonacci	test 1	—	0	0	4	17
garbage collection	test 1	—	0	—	2	19
	test 2	—	0	0	0	0
reverse	test 1	0	0	770	1	18

Table 2
Times (in milliseconds) for the unconditional term rewriting systems, manually generated or optimized from the REC specifications (no optimization was provided by Stratego/XT, the values from Table 1 are included here to simplify their comparison).

rare in regular Stratego programs, and not something it is optimized for. This may explain the results of tests 4 and up in the ASF+SDF benchmark. This may also be the case for the sorting algorithms of Table 3. For Stratego, this is a somewhat unnatural way of expressing these problems. A hand-coded solution would likely be very dissimilar to the presented problem specification, making it hard to compare it with solutions of other systems in a meaningful way. This is the reason why they chose to focus on a purely translator-based approach.

Although the Tom system fails at the automatically generated version of the ASF+SDF benchmark for tests 5 and 6 (see Table 1), the optimizations introduced allows it to handle them (see Table 2), and quite efficiently, we must say.

Since we do not have ASF+SDF versions of the mergesort and quicksort problems, and the Stratego/XT developers did not optimize their codes, no conclusion can be given from the results for these problems, but notice how the optimizations introduced in the Maude code allows it to outperform the other systems. This was not the case when considering the automatically generated ones, were the Tom compiler is much faster. The Stratego/XT system seems to have some problems handling these tests.

As to Maude's performance, there were no big surprises. In an interpreter, unused features cannot be optimized away an so trade offs have to be made, whether to optimize the interpretation code for the most common case or whether to optimize for a particular feature. In the case of the Maude interpreter, unconditional

		Maude	Stratego/XT	Tom
bubblesort	test 1	4	0	35
	test 2	471	140	218
	test 3	319	140	184
fibfree	test 1	28	20	70
hanoi	test 1	1	—	44
mergesort	test 1	1	0	58
	test 2	50	—	26
	test 3	23599	—	614
missionaries	test 1	40	20	102
oddeven	test 1	192	0	18
	test 2	7	0	235
	test 3	—	0	1
quicksort	test 1	2	—	30
	test 2	113	—	64
	test 3	495579	—	11397

Table 3
Times (in milliseconds) for the conditional term rewriting systems, automatically generated from the REC specifications.

rewriting, particularly modulo, is highly optimized, at the expense of having a lot of state that is distributed over multiple internal data structures. In order to evaluate a condition this state must be saved before the condition is evaluated and restored afterwards, which makes conditional rewriting very expensive. This can be seen in the quicksort example, where a naive translation makes use of conditional equations whereas a translation by a human expert uses the builtin `if_then_else_fi` operator, which is very cheap.

5 Conclusions

As in the First Rewrite Engines Competition, we believe that both rewrite engines users and developers have benefited from this second edition of the competition. Although in this second edition we took a great step forward, gathering more systems, proposing the REC language and providing translators for some of the systems, there is still a lot to be done. In any case, our main goals were satisfied: we got to know each of the systems better, some of the strengths and weaknesses of the engines were shown, and we got more motivation to go on working on our

		ASF+SDF	Maude	Stratego/XT	TermWare	Tom
bubblesort	test 1	0	4	0	417	35
	test 2	0	471	140	67	218
	test 3	60	319	140	—	184
fibfree	test 1	—	28	20	464	65
hanoi	test 1	0	1	—	8	45
mergesort	test 1	—	0	0	62	101
	test 2	—	0	—	1228	30
	test 3	—	11	—	—	600
missionaries	test 1	—	7	20	358	102
oddeven	test 1	0	0	0	2	21
	test 2	0	0	0	0	235
	test 3	0	0	0	0	1
quicksort	test 1	—	0	—	5	30
	test 2	—	6	—	22	64
	test 3	—	859	—	2782	11397

Table 4
Times (in milliseconds) for the conditional term rewriting systems, manually generated or optimized from the REC specifications (no optimization was provided by Stratego/XT, the values from Table 3 are included here to simplify their comparison).

respective systems. In addition to these results, the five systems were presented in a special session of WRLA'08, what also helped to give some additional visibility to the systems in the competition, and to the competition itself.

As a conclusion, we can say that we have very fast rewrite engines out there. They are very specific, but very good at what they were conceived for.

Compilers are more efficient in the problems they support. It would be good to see how they compare doing AC matching, or how new techniques are developed so that systems like Maude can go compiled.

In future editions of the competitions, we would like to get a better idea of:

- memory usage, other than some of the engines crashed for some of the problems;
- development times, for the REC translators, for example; how to measure this?
- correctness, notice that we were assuming that the outputs given by the engines was correct; checking the results given should not be complex,
- big problems, what about scalability?

And one wish for the competition: More automatization is required! For entering

the programs, without requiring later optimization, time capture, results table generation, etc. During the course of the weeks before the competition was presented, we came up with extensions to the REC language to automatically validate tests, and worked on automated building and testing. This process could be improved in the future.

Acknowledgement

We would like to thank Grigore Roşu, as organizer of WRLA 2008 and the First Rewrite Engines Competition, for his help and support. His experience and comments were very useful. And, of course, we have to thank all the people who has participated in the development of all the rewrite engines in the competition.

References

[1] Balland, E., P. Brauner, R. Kopetz, P.-E. Moreau and A. Reilles, *Tom: Piggybacking rewriting on java*, in: F. Baader, editor, *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, Lecture Notes in Computer Science **4533** (2007), pp. 36–47.

[2] Bravenboer, M., K. T. Kalleberg, R. Vermaas and E. Visser, *Stratego/xt 0.17. a language and toolset for program transformation*, Science of Computer Programming **72** (2008), pp. 52–70.

[3] Clavel, M., F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and J. Quesada, *Maude: Specification and programming in rewriting logic*, Theoretical Computer Science **285** (2002), pp. 187–243.

[4] Clavel, M., F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and C. Talcott, "All About Maude - A High-Performance Logical Framework," Lecture Notes in Computer Science **4350**, Springer, 2007.

[5] Denker, G., C. Talcott, G. Rosu, M. van den Brand, S. Eker and T. F. Şerbănuţă, *Rewriting logic systems*, ENTCS **176** (2007), pp. 233–247.

[6] Shevchenko, R. and A. Doroshenko, *A rewriting framework for rule-based programming dynamic applications*, Fundamenta Informaticae **72** (2006), pp. 95–108.

[7] van den Brand, M. G. J., J. Heering, P. Klint and P. A. Olivier, *Compiling language definitions: the ASF+SDF compiler*, ACM Transactions on Programming Languages and Systems **24** (2002), pp. 334–368.

[8] van den Brand, M. G. J., A. van Deursen, J. Heering, H. Jong, M. Jonge, T. Kuipers, P. Klint, L. Moonen, P. A. Olivier, J. Scheerder, J. Vinju, E. Visser and J. Visser, *The ASF+SDF Meta-Environment: a Component-Based Language Development Environment*, in: R. Wilhelm, editor, *CC'01*, LNCS **2027** (2001), pp. 365–370.

[9] Visser, E., *Stratego: A language for program transformation based on rewriting strategies. System description of Stratego 0.5*, in: A. Middeldorp, editor, *Rewriting Techniques and Applications (RTA'01)*, Lecture Notes in Computer Science **2051** (2001), pp. 357–361.