Symbolic demonstrations in MuPAD-Combinat

HOUDA ABBAD[1]
ÉRIC LAUGEROTTE[2]

Evolutionary Engineering and Distributed Information Systems Laboratory,
Department of Computer Science, Djillali Liabes Univesity of Sidi Bel Abbes,
BP 89, 22000 Sidi Bel Abbes, Algeria[1]
Laboratoire d'Informatique de Traitement de l'Information et des Systèmes,
Department of Computer Science, University of Rouen,
Avenue de l'Université, BP 12, 76801 Saint Étienne du Rouvray Cedex, France[2]
[1]houda.abbad@lycos.com
[2]eric.laugerotte@univ-rouen.fr

Abstract. This paper reports on a platform integrated to the open source package MuPAD-Combinat dedicated to the computation with rational expressions. Its main feature is the capacity of dealing with expressions whose scalars may belong to various algebraic structures. The paper describes some features of the platform as well as the data structures used in the implementation, details some algorithms and presents several illustrations of symbolic computations.

Keywords: rational series, weighted automata, rational expressions, symbolic computations, MuPAD, MuPAD-Combinat.

(Received April 11, 2008 / Accepted October 29, 2008)

1 Introduction

Rational expressions are finite representations of a class of non-commutative formal series called rational [4, 15]. Initially created to describe formal languages, the rational expressions are naturally used in the interpretation of formal languages and compilers. Their applications take place in a lot of scientific areas as quantum computation, logic, bioinformatics, automatic search of text, natural language processing etc. Rational expressions have new application fields with the development of Internet, and the diffusion of hostile code and spam messages. The filters use expressions to detect the potentially harmful elements. In 1961, Schützenberger states the famous equivalence between the set of weighted automata and rational expressions [23]. Working with rational series via weighted automata is often more interesting. For this purpose, a rich toolbox dedicated to weighted automata in MuPAD-Combinat [1] has been developed. It is intended for use in research but also in teaching with pedagogical purposes. Many packages manipulating finite-state machines have been developed as Automate [6] or AMoRE [19], but for a given set of weights except FSM [21] and Vaucanson [9] platforms. Using a computer algebra system, our software offers more general computations. In fact, weights are taken from many semirings which are minimal algebraic structures for suitable computations. However, just few systems as our environment and Vaucanson are able to work with rational expressions. This library is included in the open-source algebraic combinatorics package MuPAD-Combinat [13] for the computer algebra system MuPAD [18] (2.0.0 and higher).

The purpose of this paper is not to be a user manual of the library and even not to list all its functionalities. We give here only a quick overview on what is to be found in the library. In the brief presentation that follows, we describe the design of a library devoted to manipulations of rational expressions whose scalars belong to any semiring. In the first part, we introduce the package MuPAD-Combinat and we justify the motivations of this choice. In the following section, some theoretical notions are reminded as semirings, automata and expressions. We illustrate these formal definitions with several examples created in MuPAD-Combinat. Next, we present a sharp tree structure for efficient handlings of rational expressions. Then, we detail just two algorithms used in symbolic computations. In section 6, we give an idea about the use of the software with a symbolic demonstration. Finally, we present a brief comparaison of the software performances with an analogous platform called Vaucanson. All tests have been run on a Centrino 1.6 GHz (1 GB of RAM) with MuPAD Pro 4.0.2, MuPAD-Combinat 1.3.2 and Vaucanson 0.7.

2 Inside MuPAD-Combinat

Our research works in combinatorics and computer science deal with formal series. Implementation of formal series in a computer algebra system allows to carry out many tests and experiences in order to visualize known results but also to validate conjectures. The research in algebraic combinatorics benefits from the computer exploration, which requires flexible and efficient symbolic tools. Such tools are valorized by the scientific community, both in research and teaching to illustrate current results and undertake new researches.

We want a flexible and extensible open-source toolbox being composed of standard combinatorial objects with a free license. Furthermore, we need an oriented-object language providing the inheritance between algebraic structures, the overloading mechanism and a rapid implementation. The open-source algebraic combinatorics package MuPAD-Combinat for the computer algebra system MuPAD [18] answers in part to our requirements. On the Florent Hivert and Nicolas Thiéry's initiative, the project started in spring 2001. The core of the package is integrated in the official library of MuPAD[1] since version 2.5.0. A detailed description of the project is introduced in [13]. MuPAD ensures us the stability of code, the technical support and the collaboration with its developers team. This collaboration includes code reviews, explanations on the internal functioning of MuPAD and discussions on common developments. MuPAD kernel is built on logical and sound bases allowing us easy tools development but also addition of new functionalities in the kernel. The last feature makes MuPAD reasonably open than other algebra systems. Moreover, MuPAD offers a categorial programming language with categories, domains and axioms which is suitable for symbolic handlings especially those of weighted automata or again rational expressions. The choice of the computer algebra system MuPAD has been done after a serious study of the existing softwares. Obviously, we would have preferred to use a computer algebra system that was already widely used like Maple [8] or Mathematica [24]. However, the programming language, the license conditions, and the long term maintenance of Maple are not suited to our needs. Technically, Mathematica looks better, but does not seem to fit our technical requirements concerning the object oriented features. For more technical details, the reader should refer to [13]. MuPAD-Combinat works with combinatorial objects such as words, trees, graphs, combinatorial algebras or weighted automata and recently rational expressions. It provides a number of domains or classes that are types and methods attached to these objects, in order to deal with them. During six years of existence, the project has been realized in 10 official versions among which 6 are stable. In addition, it has been cited in more than 25 research articles. Actually, more than thirty developers being interested in combinatorics and the computer exploration, participate to the project. A last release of the package can be loaded from the web page `http://mupad-combinat.sf.net.` In this web site, we find the reference manual, users and developers mailing lists, bug reports and how to contribute to the project.

[1]downloadable from the web site `http://www.mupad.com`.

3 Theoretical background

This section is devoted to theoretical definitions such as semirings, non-commutative formal series, weighted automata or again rational expressions. These prerequisites are important for the understanding of the paper.

3.1 Semirings

Semirings are convenient structures to compute with finite-state machines. A semiring $(R, \oplus, \otimes, 0, 1)$ is a set together with two laws and their neutrals. More precisely $(R, \oplus, 0)$ is a commutative monoid with 0 as neutral and $(R, \otimes, 1)$ is a monoid with 1 as neutral. The product is distributive with respect to the addition and zero is an annihilator. Besides the semiring of natural numbers, we can cite the boolean semiring, exotic semirings as the MinPlus semiring, numerical semirings (integers, rationals, reals or complex numbers), or again the family of all subsets of a nonempty set with union and intersection. A particular interest for our work is reserved for the star defined here in a formal way. Let $x \in R$, we call $\mathcal{L}(x)$ (resp. $\mathcal{R}(x)$) the set of solutions of the equation $y \otimes x \oplus 1 = y$ (resp. $x \otimes y \oplus 1 = y$). A star of x belongs to $\mathcal{S}(x) = \mathcal{L}(x) \cap \mathcal{R}(x)$ if it is not empty. Therefore, we write $x^{\circledast}$ the star of x if there exists. In this paper, we use semirings which have at most one star for each $x \in R$. In practice, the most used are these ones (as Kleene semirings or Conway semirings). In MuPAD, a semiring belongs to the category `Cat::SemiRing` that sets the main properties of semirings. It inherits `Cat::AbelianMonoid` (properties of abelian monoid for $\oplus$) and `Cat::Monoid` (properties of monoid for $\otimes$). According to the variety of semirings which can be used in the theory, we have developed in MuPAD-Combinat a domain constructor `Dom::SemiRing`. This domain creates a semiring where elements belong to a set or satisfy a property. The general call of this domain constructor is detailed below:

```
Dom::SemiRing(Coefficient=coefficient,
<, Prop=prop, Plus=plus, Zero=zero,
Negate=negate, Mult=mult, One=one,
Invert=invert, Star=star,
Categ=categ>)
```

The constructor of the domain `Dom::SemiRing` builds a semiring where elements belong to `coefficient` which can be a domain, a set , or an union of domains and sets written as a list. Note that the rest of parameters is optional for the constructor, since they may be inherited from `coefficient` (except `prop`, `star` and `categ`) which must be a domain in this case. If the parameter `prop` is specified, the elements must satisfy this property. The addition, zero and opposites can be given by `plus`, `zero` and `negate` respectively. The multiplication, one and inverses by `mult`, `one` and `invert` respectively, the star by `star`. When `categ` appears, it defines the category of the domain. For more complicated semirings, appropriate MuPAD domain constructors are advised. For more details, we refer the reader to [1]. For instance, we deal with the implementation of the well known boolean semiring that is composed of two elements 1 and 0 representing the boolean values TRUE and FALSE. The two boolean elements admit a unique star which is 1:

```
MuPAD Pro 4.0.2 --  The Open Computer
Algebra System
Copyright (c) 1997-2007 by SciFace
Software
All rights reserved.
Licensed to: MuPAD Combinat Developer

>> B := Dom::SemiRing(
>> Coefficient = {0,1},
>> Plus =
>> ((e,f) -> if bool(extop(e,1)=1)  or
>>             bool(extop(f,1)=1) then
>>             return(1)
>>             else
>>             return(0)
>>           end_if),
>> Zero = 0,
>> Mult =
>> ((e,f) -> if (extop(e,1)=1)  and
>>              (extop(f,1)=1) then
>>              return(1)
>>              else
>>              return(0)
>>           end_if),
>> One = 1,
>> Star = (e -> 1)):
```

The star of the arithmetical expression $1 \otimes 0 \oplus 1$ in the boolean semiring is then given by:

```
>> B::star(B(1)*B(0)+B(1))

                    1
```

Few semiring domains already exist in MuPAD, and they are predefined. We have usual numerical domains as `Dom::Integer` for integers, `Dom::Rational`, `Dom::Float` or `Dom::Complex` for rational, real or complex numbers. In addition, we have integrated less classical semirings into MuPAD- Combinat. Indeed, we have the possibility to work with the boolean semiring having the syntax `Dom::BooleanSemiRing` and exotic ones as MinPlus (`Dom::MinPlusSemiRing`), MaxPlus (`Dom::MaxPlusSemiRing`) and MinMax semiring `Dom::MinMaxSemiRing` or again MaxMin (`Dom::MaxMinSemiRing`). We point out that we can compute the star of a scalar belonging to any semiring when it exists via the method `star`. The following MuPAD code constructs the MinPlus semiring $(2\mathbb{N} \cup \infty, \min, +)$:

```
>> Z := Dom::Integer:
>> s := x -> 0:
>> t := Type::Union(stdlib::Infinity,
>>      Type::Predicate("2N",
>>      x -> bool(0 <= x) and
>>      testtype(x, Type::Even))):
>> N2 := Dom::SemiRing(Coefficient = Z,
>>      Prop = t, Plus =min,
>>      Zero = infinity,
>>      Mult = Z::_plus, One = 0,
>>      Star = s):
```

The scalar 5 can not belong to the semiring N2 since it is an odd number:

```
>> N2(5)

 Error: Wrong type of 1. argument (type \
 'Type::SemiRingElement' expected, got \
 argument '5'); during evaluation of 'r'
```

The evaluation of $6 \otimes 2$ gives the scalar 8 as the multiplication in the semiring N2 is the addition of integers:

```
>> N2(6)*N2(2)

                           8
```

3.2 Formal series and weighted automata

Let A be an alphabet. A non-commutative formal series is a map from the free monoid A^* to a semiring R which associates a weight $(S, w) \in R$ to the word $w \in A^*$. We denote by ε the empty word of the free monoid A^* and $|w|$ the length of a word $w \in A^*$. Non-commutative formal series make up a particular semiring in combinatorics denoted $R\langle\langle A\rangle\rangle$. Let $S \in R\langle\langle A\rangle\rangle$. It can be written $S = \sum_{w\in A^*}(S, w)w$. Addition and Cauchy product are symbolized by "$+$" and "$\cdot$" instead of "$\oplus$" and "$\otimes$", in order to distinguish them with the operations over the set of coefficients R. The star of a non-commutative formal series S exists if $(S, \varepsilon)^{\circledast}$ exists. In this case, we denote it S^*. A class of non-commutative formal series are the behaviours of weighted automata. A weighted automaton or a linear representation $\mathcal{A} = (\lambda, \mu, \gamma)$ with weights in a semiring R is given by a set of n states Q, the map $\mu : A \to R^{n\times n}$ associating a transition matrix to each letter, the initial vector $\lambda \in R^{1\times n}$ and the final vector $\gamma \in R^{n\times 1}$ (see Figure 1). If we point out the nature of weights, we just say R-automaton. Dimension of a weighted automaton $\mathcal{A}$ is its number of states. The map μ can be extended as morphism of monoids from A^* to $R^{n\times n}$. The weight of a word w for $\mathcal{A}$ is $\lambda\mu(w)\gamma$ and the behaviour of $\mathcal{A}$ is the formal non-commutative series defined by $(S_{\mathcal{A}}, w) = \lambda\mu(w)\gamma$. The set of behaviours of weighted automata is closed by addition, Cauchy product and star, but also shuffle, hadamard and infiltration products which are dual or extended rational operations.

In a previous work, we have designed a rich environment allowing general handlings related to weighted automata in MuPAD-Combinat. The environment provides the most usual algorithms on weighted automata as determinization and minimization which have a crucial importance for both time and space efficiency. Test of equivalence, automatic drawings of automata or again algebraic elimination of letters are also integrated. Users can work with these objects and compute weights of recognized words. Besides, all rational or dual operations and many other functions that change and simplify the geometry of weighted automata are available. Each of these algorithms has been written to support a large variety of automata. For a detailed description of the platform we refer the reader to [1].

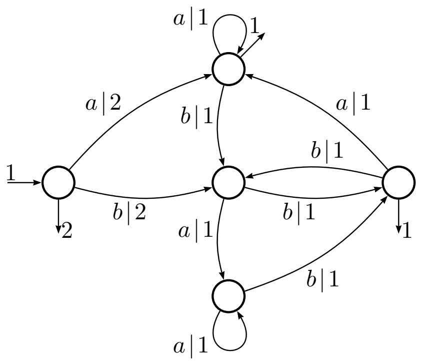

Figure 1: A $\mathbb{Z}$-automaton

Now let us create the $\mathbb{Z}$-automaton drawn in Figure 1. For this purpose, we use the domain constructor Dom::WeightedAutomaton, which is associated to the set of weighted automata in the package MuPAD-Combinat. We first construct the set of $\mathbb{Z}$-matrices and $\mathbb{Z}$-automata by:

```
>> Z := Dom::Integer:
>> ZM := Dom::Matrix(Z):
>> ZWA := Dom::WeightedAutomaton(Z):
```

We then define the alphabet, initial and final vectors and transition matrices of the automaton:

```
>> alphabet := [a, b]:
>> iv := ZM(1,5,table((1,1)=1)):
>> ma := ZM(5,5,table((1,2)=2,(2,2)=1,
>>          (5,2)=1,(3,4)=1,(4,4)=1)):
>> mb := ZM(5,5,table((1,3)=2,(2,3)=1,
>>          (4,5)=1,(5,3)=1,(3,5)=1)):
>> fv := ZM(5,1,table((1,1)=2,(2,1)=1,
>>          (5,1)=1)):
```

Line 2 means that only the first state is initial with the weight 1. In line 3, the syntax (1,2)=2 creates a transition from the state 1 to the state 2 with the label a and the weight 2. Lines 7 and 8 set that the states 1, 2 and 5 are final with the weights 2, 1 and 1 respectively. We finally build the $\mathbb{Z}$-automaton by the following call:

```
>> z_aut := ZWA(5,alphabet,iv,
>>                   table(a=ma,b=mb),fv)
```

$$\begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix},\quad a=\begin{pmatrix}0&2&0&0&0\\0&1&0&0&0\\0&0&0&1&0\\0&0&0&1&0\\0&1&0&0&0\end{pmatrix},$$

$$b=\begin{pmatrix}0&0&2&0&0\\0&0&1&0&0\\0&0&0&0&1\\0&0&0&0&1\\0&0&1&0&0\end{pmatrix},\quad \begin{pmatrix}2\\1\\0\\0\\1\end{pmatrix}$$

Weighted automata are encoded by means of sparse matrices. The computer algebra system MuPAD provides an efficient storage of this kind of matrices with polynomials representing column-vectors. Thus, although zeros are displayed, they are not really stored. Furthermore, fast algorithms dealing with matrices are implemented. As an example, let us cite the star of matrices which is computed with a divide-and-conquer algorithm [10]. After creation, we can for instance compute the weight of the word $baabab ab$ through this $\mathbb{Z}$-automaton:

```
>> ZWA::weight(z_aut,[b,a,a,b,a,b,a,b])

                                  2
```

3.3 Rational Expressions

In 1961, Schützenberger proved the famous equivalence between the behaviours of weighted automata and the rational series [23]. The main result of this theorem is finite writings of behaviours called rational expressions. A R-expression over an alphabet A is inductively defined as follows:

- $a \in A$ and $r \in R$ are R-expressions,
- if E, F and G (such that $(G, \varepsilon)^{\circledast}$ exists) are R-expressions, then $E + F$, $E.F$ and G^* are also R-expressions.

We define in the set of R-expressions the null term function c. It associates at each R-expression E a value in R which is the weight of the empty word in E i.e. (E, ε). It can be recursively computed by:

$$c(a) = 0 \qquad c(r) = r \qquad c(F^*) = c(F)^{\circledast}$$
$$c(F \cdot G) = c(F) \otimes c(G) \quad c(F + G) = c(F) \oplus c(G)$$

The set of definition of the function c is called rational R-expressions. For example, the $\mathbb{Z}$-expression $E_1 = 2 \cdot (a + b \cdot a^* \cdot b)^*$ is rational as its null term $c(E_1) = 2 \otimes (0 \oplus 0 \otimes 0^{\circledast} \otimes 0)^{\circledast}$ is well defined and equal to 2. The size of a rational expression E, denoted $|E|$ is the size of the syntactical tree of E which is the number of scalars, letters and operations appearing in E. Hence, the size of the rational $\mathbb{Z}$-expression E_1 is $|E_1| = 11$.

In the following, we will introduce a first manipulation of rational expressions. We want to create the set of rational $\mathbb{Z}$-expressions over the semiring $\mathbb{Z}$ and an alphabet constituted of two letters a and b. For this goal, we use the domain `Dom::RationalExpression` in MuPAD-Combinat. We precise as arguments in the constructor, the alphabet and the semiring of weights $\mathbb{Z}$:

```
>> ZExp := Dom::RationalExpression(
>>                          alphabet, Z):
```

We adopt the following formalism in the writing: `a,r, a+b,a*b,a^n` and `star(a)` denoting respectively the rational R-expressions $a \in A$, $r \in R$, $a + b$, $a.b$, $a.a\ldots a$ (n times) and a^*. Therefore, the creation of the rational $\mathbb{Z}$-expression $2 \cdot (a + b \cdot a^* \cdot b)^*$ is realized by:

```
>> E1 := 2*star(ZExp(a)+ZExp(b)*
>>                star(ZExp(a))*ZExp(b))

         2 star(a+ (b star(a)) b)
```

The implementation of rational expressions requires a suitable structure allowing efficient computations. For this reason, we have chosen as internal data structure a generalization of syntactical trees of arithmetical expressions called ZPC. This algorithmic structure can be computed in a linear time from a rational expression and stands halfway between rational expressions and equivalent weighted automata.

4 ZPC-structure

In [25], the ZPC-structure has been proposed to represent efficiently boolean rational expressions. Next in [7], it has been extended to the multiplicity case. This structure can be constructed in a linear time with respect to the size of the rational expression. Indeed, from a rational expression it is based on two similar trees, the first tree and the last tree whose transitions are decorated with scalars belonging to the semiring R. These copies of the syntactical tree are joined by links meaning Cauchy product or star operation. The reader should refer to [7] for formal definitions and proofs.

As in [7], a node in a tree is written ν and the root is denoted by ν_0. If the arity of ν is two, ν_l and ν_r represent respectively its left and right descendant. When its arity is one, its unique descendant is ν_s. The relation of descendance over the nodes of a tree is denoted $\preceq$. It is the transitive closure of the relations $\nu_l \preceq \nu$, $\nu_r \preceq \nu$

or $\nu_s \preceq \nu$ according to the various cases. For a tree whose edges are labelled by scalars of the semiring R, we define the function of cost π. Suppose that $\nu' \preceq \nu$, the cost $\pi(\nu, \nu')$ is the weight of the path from ν to ν'. For any node ν, the rational R-expression E_ν denotes the subexpression resulting from ν as root and $c(\nu)$ its null term.

The first tree TF_E is deduced from the syntactical tree of E. Its nodes are written φ and its edges are directed from the root φ_0 to the leaves. When a node φ is labelled by "$\cdot$", we associate the weight $c(\varphi_l)$ to the edge from φ to φ_r. If a node φ is labelled by "$*$", we assign the weight $c(\varphi_s)^{\circledast}$ to the edge from φ to φ_s. Any edge not marked in previous steps will be marked by 1.

The last tree TL_E is also deduced from the syntactical tree E where each node is written ρ. Edges in TL_E are directed from leaves to the root ρ_0. For each node ρ labelled by "$\cdot$", we endow the weight $c(\rho_r)$ to the edge from ρ_l to ρ. If a node ρ is labelled by "$*$", the weight $c(\rho_s)^{\circledast}$ is assigned to the edge from ρ_s to ρ. Just like the first tree TF_E, any edge not marked in previous steps will be marked by 1.

The follow links connect the last tree TL_E to the first tree TF_E. Let ρ be a node in TL_E and φ its corresponding node in TF_E. For each node ρ labelled by "$\cdot$", we set a link from ρ_l to φ_r except if ρ_l is labelled by a scalar. For any node ρ labelled by "$*$", we create a link from ρ_s to φ. The ZPC-structure of the rational $\mathbb{Z}$-expression E_1 is shown in Figure 2.

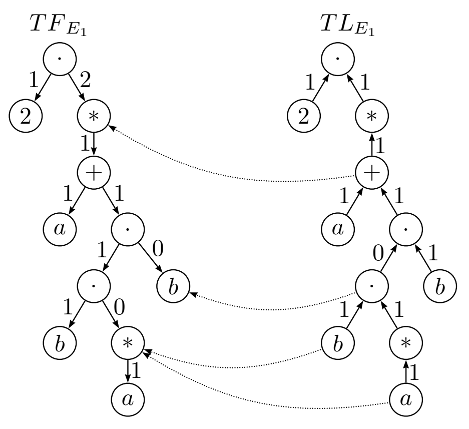

Figure 2: The ZPC-structure of $2 \cdot (a + b \cdot a^* \cdot b)^*$.

According to Figure 2, we can notice that each tree has eleven nodes since 11 is the size of the rational $\mathbb{Z}$-expression E_1. On the first tree TF_{E_1}, the scalar 2 is associated to the edge relating the root to the node labelled by "$*$" as $c(2) = 2$. The library gives the possibility to visualize this sharp structure, with the method `ZPCstructure`:

```
>> ZExp::ZPCstructure(E1)
```

```
1  table(
2    11 = ["*", 1, 10],  table(
3    10 = ["star", 9],     (11, 1) = 1,
4    9 = ["+", 2, 8],      (11, 10) = 2,
5    8 = ["*", 7, 5],      (10, 9) = 1,
6    7 = ["*", 3, 6],      (9, 8) = 1,
7    6 = ["star", 4],  ,   (9, 2) = 1,  ,
8    5 = b,                (8, 7) = 1,
9    4 = a,                (8, 5) = 0,
10   3 = b,                (7, 3) = 1,
11   2 = a,                (7, 6) = 0,
12   1 = 2                 (6, 4) = 1
13 )                     )
14
15    table(
16      (10, 11) = 1,
17      (1, 11) = 1,
18      (9, 10) = 1,   table(
19      (8, 9) = 1,      4 = [9, 10],
20      (2, 9) = 1,  ,   3 = [7, 5],
21      (5, 8) = 1,      2 = [3, 6],
22      (7, 8) = 0,      1 = [4, 6]
23      (6, 7) = 1,    )
24      (3, 7) = 1,
25      (4, 6) = 1
26    )
```

The library creates for each rational expression a set of dynamic tables coding the syntactic tree, the first, the last trees and the follow links respectively. Each entry of the first table corresponding to the syntactic tree represents a node. A node is a letter (as `2 = a` in line 11), a scalar (`1 = 2` in line 12) or again an operation (`7 = ["*", 3, 6]` in line 6). The entry associated to an operation is a list containing the label of the node and its descendants. For example, `7 = ["*", 3, 6]` means that the node 7 with the label "$\cdot$" is the predecessor of the nodes 3 and 6. The two tables that follow store the weights of edges appearing in the first and the last tree. In line 9, the writing `(8, 5) = 0` sets that the weight of the edge joining the nodes 8 and 5 in the first tree TF_{E_1} is 0. Finally, the last table retains all follow links of the ZPC-structure. The syntax `1=[4, 6]` in line 22 codes the first link connecting the node 4 of the last tree TL_{E_1} to the node 6 of the first tree TF_{E_1}.

5 On the algorithms

This library offers several handlings over expressions as rational operations, semirings conversions and checking equality. We can also compute the weights of recognized words and construct equivalent weighted automata. An important feature in symbolic computations is to be able to convert weighted automata to rational expressions. It is done by actions on the geometry of weighted automata. We are currently working on these effective algorithms. This result will enrich the library by supplying new manipulations as shuffle, hadamard and infiltration products of rational expressions. In the sequel, we focus on two algorithms of conversions to weighted automata and computing weights of recognized words.

5.1 Expressions versus automata

The transformation of an expression into an automaton has given rise to a very rich literature. In fact, recent researches in the theory deal with the conversion of a rational expression into a weighted automaton. In [5], a Glushkov automaton is built in a cubic time. Then, a generalization of the work of Antimirov [3] is detailed in [17]. Finally in [16], the notion of weighted position automaton is presented via an extended definition of Thompson automata, and in [7], the ZPC-structure is introduced for the multiplicity case. The main advantage of the ZPC-structure is to allow a construction of the so-called position automaton in a quadratic time. For its efficiency, we have implemented the algorithm and the data structures proposed in [7]. Moreover, for pedagogical goals we chose to include Thompson construction with an efficient algorithm [16]. A convenient practice using our library is to write the different constructions of weighted automata from rational expressions. The following MuPAD code computes the position $\mathbb{Z}$-automaton equivalent to E_1:

```
>> pos1 := ZExp::automaton(E1)
```

$$\begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix}, \quad a = \begin{pmatrix}0 & 2 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0 & 0\end{pmatrix},$$

$$b = \begin{pmatrix}0 & 0 & 2 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0 & 0\end{pmatrix}, \quad \begin{pmatrix}2\\ 1\\ 0\\ 0\\ 1\end{pmatrix}$$

The method returns obviously an object belonging to the set of weighted automata in MuPAD-Combinat:

```
>> domtype(last(1))

  Dom::WeightedAutomaton(Dom::Integer)
```

It is not difficult to observe that this position automaton is equal to the $\mathbb{Z}$-automaton drawn in Figure 1.

5.2 Computing weights of words

The classical way to compute weights of recognized words for a rational expression is first to construct an automaton $\mathcal{A} = (\lambda, \mu, \gamma)$ of dimension n, where the best complexity is quadratic, then to evaluate the weight of a word w through the automaton $\mathcal{A}$ with a matrix product $\lambda\mu(w)\gamma$ in $\mathrm{O}(|w| \times n^2)$. This comlexity is due to the multiplication of a row vector to a matrix for each letter of the word w. Thus, the best time to recognize a word w including the construction of the equivalend automaton is $\mathrm{O}(|w| \times |E|^2)$. However, the problem of the weighted recognition has been resolved in $\mathrm{O}(|w| \times |E|)$ using a generalization of Thompson automata without ε-transition removal [16]. This time complexity is also reached in [2] with ZPC-structures. We describe in [2] a novel efficient algorithm computing the weight of a word w for a rational expression E, without going through the construction of an equivalent weighted automaton. Using the corresponding ZPC-structure of E, the algorithm assigns coefficients to the nodes of the first and the last trees. For each letter w_i in the word w, there is a bottom-up traversal of the last tree that depends on the coefficients of edges and leaves labelled by letters. Next, via the follow links, some nodes of the first tree are provided with weights, and a top-down walk of the first tree yields new coefficients for some leaves labelled by letters. Hence, the computation of (E, w) is done in $\mathrm{O}(|w| \times |E|)$. Proofs and details are given in [2]. This algorithm has been integrated to the library with the method `weight`. The weight of the word $baababab$ for the rational $\mathbb{Z}$-expression E_1 is computed by:

```
>> ZExp::weight(E1,[b,a,a,b,a,b,a,b])

                    2
```

6 Symbolic proofs

In the sequel, we give just an idea to the reader about possible manipulations offered by the library. For this purpose, we suggest an example to illustrate an automatic demonstration. Indeed, we want to prove computationaly the following relation:

$$(a - b)^* + (a + b)^* = 2 \cdot (a + b \cdot a^* \cdot b)^*$$

First, we create a second rational $\mathbb{Z}$-expression E_2 denoting $(a - b)^* + (a + b)^*$:

```
>> E2 := star(ZExp(a)-ZExp(b))+
>>                 star(ZExp(a)+ZExp(b)):
```

Next, we check simply the equality of these rational $\mathbb{Z}$-expressions:

```
>> ZExp::areEqual(E2,E1)

                              TRUE
```

In what follows, we will detail the algorithms intervening in the proof of this relation. As stressed previously, rational expressions are represented in the library by a sharp structure. Its use allows efficient computations with rational expressions such as the construction of equivalent automata. From ZPC-structures, the method `areEqual` builds the $\mathbb{Z}$-automata associated to the expressions E_1 and E_2. Then, it tries to test the equivalence of these automata. Figures 3 and 5 show both the ZPC-structure and the $\mathbb{Z}$-automaton of $(a-b)^* + (a+b)^*$.

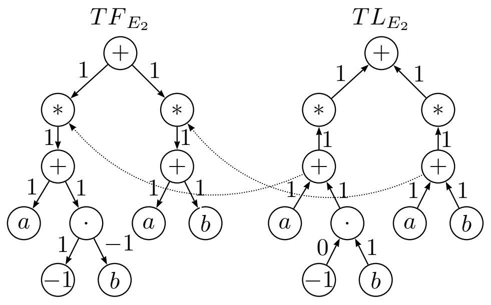

Figure 3: The ZPC-structure of $(a-b)^* + (a+b)^*$.

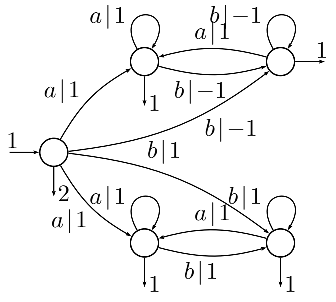

Figure 4: The position $\mathbb{Z}$-automaton of $(a-b)^* + (a+b)^*$.

```
>> pos2 := ZExp::automaton(E2)
```

$$\begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix}, a = \begin{pmatrix}0 & 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1 & 0\end{pmatrix},$$

$$b = \begin{pmatrix}0 & 0 & -1 & 0 & 1\\ 0 & 0 & -1 & 0 & 0\\ 0 & 0 & -1 & 0 & 0\\ 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 1\end{pmatrix}, \begin{pmatrix}2\\1\\1\\1\\1\end{pmatrix}$$

The equivalence of weighted automata checks whether their respective behaviours are identical. It is dependent upon the semiring R, and realizable if R is a principal ideal domain (as $\mathbb{Z}$ in our example) or a division ring. In this case, there is a minimization algorithm (in the number of states) which constructs an effective isomorphism between reduced linear representations. It is clarified in [11] and generalizes the Schützenberger's algorithm [4, 12]. When the boolean semiring is considered, the equivalence is verified with the Moore's algorithm [22]. For some exotic semirings as MinPlus, the equivalence is undecidable [14]. However, a heuristic is made under the twins property [20]. Therefore, to show the relation $(a-b)^* + (a+b)^* = 2 \cdot (a + b \cdot a^* \cdot b)^*$ amounts checking that the $\mathbb{Z}$-automaton obtained after minimization of the difference $pos_1 - pos_2$ is the empty automaton:

```
>> ZWA::minimization(pos1-pos2)
```

$$(\,), a = (\,), b = (\,), (\,)$$

The figures below illustrate the minimal $\mathbb{Z}$-automata corresponding to E_1 and E_2 with a generalization of the Schützenberger's algorithm:

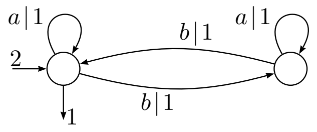

Figure 5: Minimal $\mathbb{Z}$-automaton of $2 \cdot (a + b \cdot a^* \cdot b)^*$.

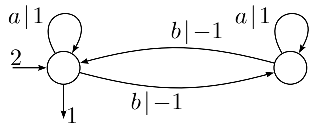

Figure 6: Minimal $\mathbb{Z}$-automaton of $(a-b)^* + (a+b)^*$.

7 Benchmarking

In this section we give some benchmarks and discuss some features of the library. For the moment, the efficiency of the software is acceptable. Table 1 gives an idea about the time and space complexities of the expression conversion algorithm. The number n represents the size of rational $\mathbb{Q}$-expressions. Table 2 presents some benchmarks about the boolean determinization algorithm such that n is the dimension of boolean automata. Clearly, Vaucanson is more efficient than our environment. The main reason of this observation is the use of C++ programming language in Vaucanson, whereas MuPAD language is used to develop our environment. Thus, a compiler is a bonus for the efficiency of our developments. In the future, we should focus our efforts to improve the performances of our software. To this end, we shall use the dynamic modules in MuPAD allowing C/C++ programming.

Table 1: Duration in ms and space in Kb of the expression conversion algorithm.

n	5	10	15	20	25	30	40
Time	7	34	95	300	465	1324	3630
Space	2	3	3	4	5	5	8

Table 2: Duration in ms of the boolean determinization algorithm.

n	2	8	14	20	26	32	35
Mu-Co	4	35	94	203	344	484	640
Vauc	2	13	28	72	160	228	283

The software offers an easy and a flexible use. Indeed, the user should not worry about the programming. One can introduce inputs in an intuitive way and execute appropriate functions. But with Vaucanson the user must write C++ code, learn Vaucanson syntax and know all its pre-compiled headers. We believe that this kind of experimentations over rational expressions is not achievable by a simple user and requires a good knowledge of C++. MuPAD language ensures the modularity of the software as it associates to each class of objects a domain that encapsulates methods handling these objects. The most interesting feature in our environment is its genericity. This characteristic has been particularly important for us during the design of the software. In fact, the software enables to create any particular semiring in a generic way. This powerful functionality enforces the genericity of the system by creating any type of weighted automata or rational expressions over created semirings. However, just few classical semirings are available with Vaucanson. Code is freely available with MuPAD-Combinat. This choice allows sharing code, experience and knowledge between researchers, developers and students but also improving the quality and the readability of code. MuPAD-Combinat installation is quick and simple, whereas Vaucanson needs a heavy one requiring particular compilers and lasting many hours in modern computers. Moreover, MuPAD-Combinat uses an adapted tool for multi-developers softwares called SVN[2] or Subversion managing the changes within their source code trees. It is a powerful method allowing to many developers to work on the same source code. Each developer checks out a copy of the current version of the source code from the SVN repository in order to work on his personal copy separately from other developers. When some changes are done, the developer commits them back to the SVN repository. The SVN server is then able to merge all the changes made by the developers. Finally, we point out that one of the greatest disadvantages of the use of Vaucanson is the lack of documentation contrary to our environment.

Table 3: Some benchmarks about MuPAD-Combinat and Vaucanson softwares.

Project	Use	Genericity	Efficiency
Mu-Co	Call functions	Excellent	Good
Vauc	Write programs	Good	Excellent

8 Conclusion

Designing a symbolic software aroses from the dream of having within the same system different theories over weighted automata, rational expressions, transducers and grammars. Our aim is to offer to the users a simple and a convenient interface to work with such objects by means of a computer algebra system. This framework is intended to be useful for research and academic communities which need a suitable and a computational toolbox.

References

[1] Abbad, H., Laugerotte, É. *Symbolic computations of weighted automata*. In Jordan International Conference for Computer Science and Engineering JICCSE04, Amman, Jordan, 2004.

[2] Abbad, H., Laugerotte, É. *Tree structures for rational expressions*. to be submitted to Internat. J. Foundations Comput. Sci.

[3] Antimirov, V. *Partial derivatives of regular expressions and finite automaton constructions*. Theoretical Computer Science, v.155, p.291-319, 1996.

[4] Berstel, J., Reutenauer, C. *Rational series and their languages*. EATCS Monographs on Theoretical Computer Science, Springer, 1988.

[2]available at the offi cial Subversion web site subversion.tigris.org or again SourceForge.net.

[5] Caron, P., Flouret, M. *Glushkov construction for multiplicities*. In 5th Int. Conf. on Implementations and Applications of Automata CIAA, Lecture Notes in Computer Science, Springer-Verlag, v.2088, p.67-79, 2001.

[6] Champarnaud, J.-M., Hansel, G. *Automate, a computing package for automata and finite semigroups*. J. Symbolic Comput, v.12, n.2, p.197-220, 1991.

[7] Champarnaud, J.-M., Laugerotte, É., Ouardi, F., Ziadi, D. *From regular weighted expressions to finite automata*. Internat. J. Foundations Comput. Sci, v.15, p.687-699, 2004.

[8] Char, B.-W., Geddes, K.-O., Gonnet, G.-H., Leong, B., Monagan, M.-B., Watt, M. *Maple Reference Manual*. WATCOM Publications Limited, 1988.

[9] Claveirole, T., Lombardy, S., O'Connor, S., Pouchet, L.-N., Sakarovitch, J. *Inside Vaucanson*. Lecture Notes in Computer Science, Springer-Verlag, v.3845, p.117-128, 2005.

[10] Duchamp, G., Hadj Kacem, H., Laugerotte, É. *Algebraic elimination of epsilon-transitions*. Discrete Math. and Theoret. Comput. Sci, v.7, p.51-70, 2005.

[11] Duchamp, G., Laugerotte, É., Luque, J.-G. *Extending the scalars of minimization*. Research report IGM 2001-01, Institut Gaspard Monge, 2001.

[12] Flouret, M., Laugerotte, É. *Noncommutative minimization algorithms*. Information Processing Letters, v.64, p.123-126, 1997.

[13] Hivert, F., Thiéry, N.-M. *MuPAD-Combinat, an open-source package for research in algebraic combinatorics*. Séminaire Lotharingien de Combinatoire, p.1-70, 2004.

[14] Krob, D. *The equality problem for rational series with multiplicities in the tropical semiring is undecidable*. Lecture Notes in Computer Science, v.623, p.101-112, 1992.

[15] Kuich, W., Salomaa, A. *Semirings, automata, languages*. Springer, 1986.

[16] Laugerotte, É., Ziadi, D. *Weighted Word Recognition*. Fondamenta Informaticae, v.83, n.3, p.277-298, 2008.

[17] Lombardy, S., Sakarovitch, J. *Derivatives of regular expression with multiplicity*. Theoret. Comput. Sci, v.332, p.141-177, 2001.

[18] Majewski, M. *MuPAD Pro computing essentials*. Springer, 2004.

[19] Matz, O., Miller, A., Potthoff, A., Thomas, W., Valkema, E. *Report on the Program AMoRE*. Report 9507, Inst. für Informatik u. Prakt. Mathematik, CAU Kiel, 1995.

[20] Mohri, M. *Finite State Transducers in Language and Speech Processing*. Computational Linguistics, v.23, n.2, p.269-312, 1997.

[21] Mohri, M., Pereira, F.-C.-N., Riley, M. *A Rational Design for a Weighted Finite-State Transducer Library*. Lecture Notes In Computer Science, v.1436, p.144-158, 1997.

[22] Moore, E.-F. *Gedanken-experiments on sequential machines*. Automata study, Annals of mathematics studies, Princeton University Press, Princeton, N. J, v.34, p.129-153, 1956.

[23] Schützenberger, M.P. *On the definition of a family of automata*. Inform. and Contr, v.4, p.245-270, 1961.

[24] Wolfram, S. *Mathematica : A System for Doing Mathematics by Computer*. Addison-Wesley Publishing Company, 1991.

[25] Ziadi, D., Ponty, J.-L., Champarnaud, J.-M. *Passage d'une expression rationnelle à un automate fini non-déterministe*. Bull. Bel. Math. Soc, v.4, p.177-203, 1997.