

Automata Reduction

Vivien Delmon

Technical Report n°0830, November 2008
revision 1996

A deterministic automaton can be minimized efficiently into an automaton that has a minimal number of states. The reduction algorithm presented here produces an automaton with a minimal number of states from a non deterministic automaton with weights defined in a field.

This algorithm computes the base of the vector space generated by the series represented by the automaton and produces an automaton with a number of states equal to the dimension of this base. To find this base the algorithm has to solve a system of linear equations, this step requires the semiring to be a field. We also want to run our algorithm on fields that are not commutative which forbids the use of classical solvers for these systems.

This report shows how we deal with these different constraints. We also present a modified algorithm to work with series over \mathbb{Z} semiring which is not a field but has some sufficient properties.

Un automate déterministe peut être minimisé de manière efficace et donne un automate dont le nombre d'états est minimal. L'algorithme de réduction présenté dans ce rapport permet de construire un automate dont le nombre d'états est minimal à partir d'un automate non déterministe dont les poids sont définis sur un semi-anneaux qui est un corps.

L'algorithme calcule pour cela la base de l'espace vectoriel engendré par la série représenté par l'automate, ce qui permet de construire un automate dont le nombre d'états est égal à la dimension de cette base. L'algorithme se base sur la représentation matricielle des automates et nous amène à résoudre un système d'équations linéaires, ce qui force le semi-anneau de notre série à avoir les propriétés d'un corps. Nous voulons aussi que notre algorithme fonctionne sur des séries dont le semi-anneaux est un corps non commutatif ce qui nous empêche d'utiliser les techniques classiques de résolution de systèmes linéaires.

Ce rapport montre comment passer outre ces difficultés. Nous verrons enfin comment adapter notre algorithme pour qu'il fonctionne sur \mathbb{Z} qui n'est pas un corps mais qui possède des propriétés suffisantes.

Keywords

Vaucanson, automata reduction



Laboratoire de Recherche et Développement de l'Epita
14-16, rue Voltaire – F-94276 Le Kremlin-Bicêtre cedex – France
Tél. +33 1 53 14 59 47 – Fax. +33 1 53 14 59 22

lrde@lrde.epita.fr – <http://www.lrde.epita.fr/>

Copying this document

Copyright © 2008 LRDE.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant Sections being just "Copying this document", no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is provided in the file COPYING.DOC.

Contents

Introduction	4
1 Preliminaries	5
1.1 Weighted Automata	5
1.2 The Matrix form	7
2 Algorithm	8
2.1 Construct the transformation matrix	8
2.2 Transform the automaton	9
2.3 Evaluating a word	10
2.4 Reduce Algorithm	10
2.5 Example	10
3 Implementation	11
3.1 Graph to matrix	11
3.1.1 μ BLAS	11
3.1.2 Standard Template Library	12
3.2 Reduce	12
3.3 Real semiring and precision	12
3.4 A field concept	13
4 Applications and Future Work	14
4.1 Comparing weighted automata	14
4.2 Adapt our algorithm for the ring \mathbb{Z}	14
4.3 Add new division rings to VAUCANSON	14
Conclusion	15
Bibliography	16

Introduction

The VAUCANSON library is a finite state machine platform designed to manipulate automata and transducers. Automata are present in lots of branches of computer sciences like model checking, natural language processing, database theory and we often want to compute the smallest automaton in order to save memory as well as execution time.

There exists different ways to find the smallest automaton that recognizes a rational expression. For instance, if we have a deterministic finite automaton (DFA) defined over the Boolean semiring we can use either the Hopcroft minimization ([Hopcroft, 1971](#)) or the Moore minimization ([Moore, 1956](#)). The complexity of these algorithms is polynomial in the number of states of the automaton. But usually a Boolean automaton is not deterministic and we have to call the determinize algorithm to obtain a DFA from a non-deterministic finite automaton (NFA). The complexity of determinize is exponential in the number of states.

The reduce algorithm presented in this report was first described by [Sakarovitch \(2003\)](#) and aims at finding the smallest representation of a \mathbb{K} -rational expression with \mathbb{K} a field (or division ring). It allows us to perform new operations in VAUCANSON like the minimization of automata with weights in a field. Some other operations are provided by combination of this algorithm with some other algorithms of VAUCANSON like the equivalence of two automata with weights in a field. Another interesting point is the complexity of this algorithm which is polynomial in the number of states of the automaton.

In this report, we present this algorithm and explain how it works and how we implemented it in VAUCANSON.

Chapter 1

Preliminaries

This chapter explains what is an automaton and introduces some notations we use in this report.

1.1 Weighted Automata

Weighted automata are automata in which the transitions are labeled with weights in addition to usual alphabet symbols. For various operations to be well defined, the weight set needs to have the algebraic structure of a semiring. To define a semiring we need the definition of a monoid:

Definition 1 A system $(\mathbb{K}, \otimes, \bar{1})$ is a monoid if:

- \otimes is associative: $\forall a, b, c \in \mathbb{K} (a \otimes b) \otimes c = a \otimes (b \otimes c)$
- $\bar{1}$ is the identity symbol for \otimes

We can now define a semiring as:

Definition 2 A system $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ is a right semiring if:

- $(\mathbb{K}, \oplus, \bar{0})$ is a commutative monoid with $\bar{0}$ as identity element for \oplus ,
- $(\mathbb{K}, \otimes, \bar{1})$ is a monoid with $\bar{1}$ as identity element for \otimes ,
- \otimes right distributes over \oplus : $\forall a, b, c \in \mathbb{K}, (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- $\bar{0}$ is an annihilator for \otimes : $\forall a \in \mathbb{K}, a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}$.

Left semirings are defined in a similar way by replacing right distributivity with left distributivity. $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ is a semiring if both left and right distributivity hold. Thus, more informally, a semiring is a ring that may lack negation.

The algebraic structure of semiring is enough to define a weighted automaton but our algorithm only works if this algebraic structure is a division ring.

Definition 3 A system $(\mathbb{K}, \oplus, \ominus, \otimes, \oslash, \bar{0}, \bar{1})$ is a division ring if:

- $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ is a semiring.
- $\forall a \in \mathbb{K} \exists \ominus a \in \mathbb{K}, a \oplus (\ominus a) = 0$ $a \oplus (\ominus b)$ is also denoted $a \ominus b$
- $\forall a \in \mathbb{K} \setminus \{\bar{0}\} \exists a^{-1} \in \mathbb{K}, a \otimes a^{-1} = 1$ $a \otimes b^{-1}$ is also denoted $a \oslash b$

Definition 4 A weighted automaton $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ over the semiring \mathbb{K} is a 7-tuple where Σ is the finite alphabet of the automaton, Q the finite set of states, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states, $E \subseteq Q \times \Sigma \times K \times Q$ the finite set of transitions, $\lambda : I \rightarrow \mathbb{K}$ the initial weight function mapping I to \mathbb{K} , and $\rho : F \rightarrow \mathbb{K}$ the final weight function mapping F to \mathbb{K} .

Given a transition $e \in E$, we denote by $l[e]$ its label, $w[e]$ its weight, $src[e]$ its source state and $dst[e]$ its destination state.

A path $\pi = e_1 \dots e_k$ in A is an element of E^* with consecutive transitions:

$$\forall i \in [2..k], dst[i-1] = src[i]$$

We extend to paths by setting:

$$\begin{aligned} l[\pi] &= l[e_1] \dots l[e_k] \\ w[\pi] &= w[e_1] \otimes \dots \otimes w[e_k] \\ src[\pi] &= src[e_1] \\ dst[\pi] &= dst[e_k] \end{aligned}$$

Given a string $x \in \Sigma^*$ we denote by $P(x)$ the set of paths from I to F labeled with x :

$$P(x) = \{\pi \in E^* \mid src[\pi] \in I, dst[\pi] \in F, l[\pi] = x\}$$

The output weight associated by A to an input string $x \in \Sigma^*$ is:

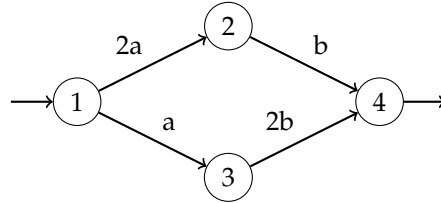
$$A.x = \bigoplus_{\pi \in P(x)} \lambda(src[\pi]) \otimes w[\pi] \otimes \rho(dst[\pi])$$

If $P(x) = \emptyset$, $A.x$ is defined to be $\bar{0}$. Note that weighted automata over the Boolean semiring are equivalent to the classical unweighted finite automata.

1.2 The Matrix form

An automaton can be stored with a set of matrices. A matrix is associated to each letter of the alphabet and stores the weights from any states to another. Weights on initial and final transitions are stored separately in two vectors that represents λ and ρ .

For instance the following automaton over the real semiring $(\mathbb{R}, +, \times, 0, 1)$:



Could be defined by the following matrix system:

$$I = [1 \ 0 \ 0 \ 0] \quad F = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \mu(a) = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \mu(b) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

To evaluate the string "ab" we multiply successively the initial vector by the matrix of a , the matrix of b and the final vector, the result gives us the output of the automaton.

$$I \times \mu(a) \times \mu(b) \times F = 4$$

Chapter 2

Algorithm

The goal of our algorithm is to find the minimal number of states to represent an automaton \mathcal{A} . When using a matrix system to represent \mathcal{A} , the problem is the same as finding the canonical base of the vector space engendered by the system representing \mathcal{A} and expressing \mathcal{A} in this base.

2.1 Construct the transformation matrix

Algorithm 1 transform_matrix

```
1: Matrix  $m$ 
2: Queue  $q \leftarrow I$ 
3: while  $q \neq \emptyset$  do
4:    $x \leftarrow q.pop$ 
5:   for  $k \leftarrow 0$  to  $m.nb\_row$  do
6:      $fnn\_x = \min\{j | x_j \neq 0\}$ 
7:      $fnn\_mk = \min\{j | m_{k,j} \neq 0\}$ 
8:     if  $x = 0$  or  $fnn\_x < fnn\_mk$  then
9:       break
10:    end if
11:    if  $fnn\_x = fnn\_mk$  then
12:       $coef = (x_{fnn\_x}) \otimes (m_{k,fnn\_mk})$ 
13:       $x \leftarrow x \ominus coef \otimes m_{k,*}$ 
14:    end if
15:  end for
16:  if  $x = 0$  then
17:     $m.insert(k, x)$ 
18:    for all letter  $l$  do
19:       $q.push(x \otimes \mu(l))$ 
20:    end for
21:  end if
22: end while
```

This method uses the Gaussian elimination and produces an echelon system which represents the base.

Definition 5 A sequence of k vectors (v^1, \dots, v^k) is an echelon system if:

$\forall i \in k$

$$(1) v_i^i = \bar{1}$$

$$(2) \forall j < i \quad v_j^i = \bar{0}$$

During the first step of Algorithm 1, we compute the base of the vector space engendered by the automaton \mathcal{A} . For this operation, we enumerate A^* the concatenation of any number of letters in our alphabet and we compute a prefix-closed subset of A^* .

We use the queue q to generate words of length 1, then words of length two etc.

$$A = \{a, b\}$$

$$A^* = \{1_{\mathbb{K}}, a, b, aa, ab, ba, bb, \dots\}$$

Then we try to find the coordinate of each word in the base m that we are constructing. We remove each part that we can express in our base from the vector representing the word we are treating (lines 6 to 14). If there is still some part that we cannot express in our base we add the rest of the vector to the base we are constructing (line 17). Finally we add vectors characterising our word with the concatenation of each letter of our alphabet (line 19) to the queue q .

For instance, if the current word is "ab" with the alphabet $A = \{a, b\}$ its vector is $I \otimes \mu(a) \otimes \mu(b)$. We multiply this vector by $\mu(a)$ to form the vector that characterises the word "aba" and push it in the queue, then we multiply it by $\mu(b)$ to form the vector that characterises the word "abb" and push it in the queue.

2.2 Transform the automaton

Now that we have our transformation matrix we have to solve some linear systems. The new input vector I' , final vector F' and letter matrices $\mu'(l)$ are defined by these three equations:

$$I'M = I$$

$$F' = MF$$

$$\forall l \in A, M\mu(l) = \mu'(l)M$$

Algorithm 2 change_base

```

1: Matrix  $m$ 
2: Vector  $input$ 
3: Vector  $output = 0$ 
4: for  $k \leftarrow 0$  to  $m.nb\_row$  do
5:    $fnn\_input = \min\{j | input_j \neq 0\}$ 
6:    $fnn\_mk = \min\{j | m_{kj} \neq 0\}$ 
7:   if  $fnn\_input = fnn\_mk$  then
8:      $coef = input_j \otimes m_{kj}$ 
9:      $input \leftarrow input \ominus coef \otimes m_{k*}$ 
10:     $output_j = coef$ 
11:   end if
12: end for

```

F' is the easiest to find. It is a simple multiplication of a matrix by a vector. For I' we have to express it in the new base defined by M using the Algorithm 2. Finally for each letter l , we express each line of $M\mu(l)$ using the Algorithm 2.

2.3 Evaluating a word

The word “ab” in the first system was evaluated by:

$$I \otimes \mu(a) \otimes \mu(b) \otimes F$$

with some simple replacements we can see that the results are the same in our new system:

$$\begin{aligned} I &= I'M \Rightarrow I'M \otimes \mu(a) \otimes \mu(b) \otimes F \\ M\mu(l) &= \mu'(l)M \Rightarrow I' \otimes \mu'(a) \otimes \mu'(b) \otimes MF \\ MF &= F' \Rightarrow I' \otimes \mu'(a) \otimes \mu'(b) \otimes F' \end{aligned}$$

2.4 Reduce Algorithm

The two steps we have just seen form the `left_reduce` algorithm which is part of the total reduce algorithm. The reduce algorithm is composed of a left and a right reduction. The right reduction is equal to the transpose of the left reduction of the transposed automaton:

$$\begin{aligned} \text{reduce}(\mathcal{A}) &= \text{left_reduce}(\text{right_reduce}(\mathcal{A})) \\ \text{reduce}(\mathcal{A}) &= \text{left_reduce}(\text{transpose}(\text{left_reduce}(\text{transpose}(\mathcal{A})))) \end{aligned}$$

The proof of this algorithm and more details about the mathematical background can be found in the draft of a chapter for the “Handbook of weighted automata” written by [Sakarovitch \(2008\)](#).

2.5 Example

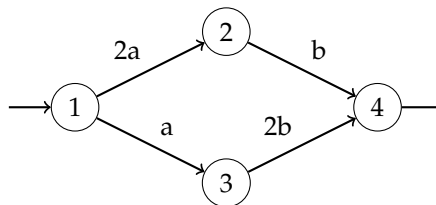


Figure 2.1: Before reduction

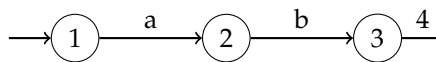


Figure 2.2: After reduction

Depending on the order in which we perform the left and the right reductions the weights are pushed either toward the input states or toward the output states.

Chapter 3

Implementation

In this chapter we explain some choices we had to do during the implementation of the reduce algorithm in VAUCANSON. Firstly we speak about the storage of an automaton in a matrix system and how to choose in which way we want to perform the reduction and finally what problems we have with algebraic structures.

3.1 Graph to matrix

In VAUCANSON, automata are stored in a graph structure. Since the reduce algorithm is based on the matrix form we have to convert the automaton into a matrix system.

When we manipulate automata represented by matrices there is often lots of holes in all our matrices. Remember that we have one matrix of $n \times n$ for each letter of our alphabet where n is the number of states in our automaton. We quickly conclude from this fact that we need sparse matrices to represent the automaton.

We first tried to use μ BLAS (Joerg Walter, 2002) but we will see in 3.1.1 what limits appeared during the implementation. Finally we will use sparse matrix home made based on the standard template library.

3.1.1 μ BLAS

μ BLAS is a C++ template class library that provides BLAS level 1, 2, 3 functionality for dense, packed and sparse matrices. The design and implementation unify mathematical notation via operator overloading and efficient code generation via expression templates.

The good point with this library is that it implements solver for linear systems and μ BLAS is part of Boost which is already a dependency of VAUCANSON. It seemed to be a really good choice for our algorithm but some problems occurred when we tried to use it.

The first one is that, from their documentation we cannot find any way to specify what objects holes in the matrix have to represent and in VAUCANSON our semiring are really different one from the other. For instance, in a tropical semiring the $\bar{0}$ is $\pm\infty$ in this case a matrix full of $\pm\infty$ should be full of holes.

The second problem is that our semiring are not always commutative and it implies that classical solvers cannot always help us.

We conclude that we want a sparse storage but our operations are so specific that we have to implement it by ourself.

3.1.2 Standard Template Library

The Standard Template Library offers different data structures. We choose to store our matrices in a `std::vector` of `std::map`. This choice is the same as the one μ BLAS programmers did for their sparse matrix and we did not try any other. We think that `std::map` of `std::map` could also be a good choice but it implies more complicated code to suppress only one `std::vector`. The code to keep our `std::map` empty in the right place is already complicated and obfuscate the code when we perform basic operation such as multiplying two matrices.

Other algorithms like ε -removal are using automaton stored in matrices but it is not exactly the same form. For instance, one of the implementation of ε -removal use a matrix to represent the automaton only composed of ε -transitions and the other construct a matrix of rational expressions to regroup every transitions from a state to another in a single cell of the matrix. An implementation of automaton in a matrix form is not the solution since there are these differences but an internal library to manipulate sparse matrices should be useful.

3.2 Reduce

The reduce algorithm is a functor that transforms the automaton in a matrix form during its construction and provides three methods `left_reduce`, `transpose` and `release`. The first one performs the left reduction, the second one transposes our automaton and the last one returns the automaton in its graph form. The reduce function delegates its work to this functor and let the user choose in which order he wants to perform left and right reduction.

```
template<typename A, typename AI>
Element<A, AI>
reduce(const Element<A, AI>& a,
       misc::direction_type dir = misc::right_left)
{
    Element<A, AI> output(a.structure());
    reducer<A, AI> r(a, output);
    if (dir == misc::right_left)
        r.transpose();
    r.left_reduce();
    r.transpose();
    r.left_reduce();
    if (dir == misc::left_right)
        r.transpose();
    r.release();
    return output;
}
```

3.3 Real semiring and precision

We tried our algorithm on the only division ring implemented in VAUCANSON which is the real ring. This implementation is based on either float or double. In these two types it is not possible to have an absolute precision and we had to overload our equality operator to overcome this problem by deciding that two float are equal if the difference between them is inferior to an epsilon. A better solution is to use a dedicated library that provides support for arbitrary precision like [GMP \(2008\)](#).

3.4 A field concept

Currently in VAUCANSON, the only thing we know about the weights of an automaton is that the algebraic structure is a semiring but we do not have any way to know if it is a ring or a field. In our algorithm the algebraic structure representing the weights must be a field and provides the negation and the division operator. For our implementation we just added these two operators to the real semiring but no static mechanisms are testing if these operators are defined in our semiring. Since all our operators are wrapped there is an error at compile time if we try to use reduce on a semiring that does not provide these operators but this error is not really explicit and we really need to provide division ring or ring abstraction in the library.

Algebra which is the name of the part in VAUCANSON that provides structures and mathematical concepts should also provide these abstractions to allow specialisation of certain algorithms considering some properties of the semiring.

Conclusion

In VAUCANSON the problem of deciding if an automaton is equivalent to another one is really important. When we are testing the different functionalities of our library we have some reference automata and we want to test if the result we obtain is the good one. Comparing the automaton to see if it is exactly the same is sometime too restrictive and an equivalent automaton could work as well. The reduction algorithm is not the answer to all our tests of equivalence since it only works on automata defined over division rings. In some semirings the equivalence relation is undecidable like in the semiring $M = (\mathbb{N}, \min, +)$ [Krob \(1994\)](#)

During the implementation of this algorithm we have seen that VAUCANSON needs some concepts that are not handled by the current concept system like the property that the semiring is in fact a division ring. A system that allows us to precise some properties of the semiring could help us to choose the right specialisation for certain types of semirings and should be study.

Bibliography

Béal, M.-P., Lombardy, S., and Sakarovitch, J. (2005). On the equivalence of \mathbb{Z} -automata. In Caires, L., F. Italiano, G., Monteiro, L., Palamidessi, C., and Yung, M., editors, *ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 397–409. Springer.

Béal, M.-P., Lombardy, S., and Sakarovitch, J. (2006). Conjugacy and equivalence of weighted automata and functional transducers. In Grigoriev, D., Harrison, J., and Hirsch, E. A., editors, *CSR 2006*, volume 3967 of *Lecture Notes in Computer Science*, pages 58–69. Springer.

GMP (2000-2008). Gnu multiple precision arithmetic library. <http://gmplib.org/>.

Hopcroft, J. E. (1971). An $n \log n$ algorithm for minimizing the states in a finite automaton. In *In The Theory of Machines and Computation*, pages 189–196. Academic Press.

Joerg Walter, M. K. (2000-2002). *Basic Linear Algebra*. Boost. <http://nwalsh.com/tex/texhelp/bibtex-15.html>.

Krob, D. (1994). The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, (4).

Moore, E. F. (1956). Gedanken-experiments on sequential circuits automata studies.

Sakarovitch, J. (2003). *Éléments de théorie des automates*. Éditions Vuibert. Table of Contents, preface and introductions to chapters available at <http://perso.enst.fr/~jsaka/ETA/>.

Sakarovitch, J. (2008). *Handbook of weighted automata*, chapter Rational and recognisable power series. <http://www.infres.enst.fr/~jsaka/ENSG/MPRI/Files/References/JS-HWA.pdf>. Draft.