Generic epsilon-removal

Vivien Delmon

Technical Report n^o0743, June 2007
revision 1470

Lots of algorithms on automata take an automaton without any ε-transition. For this reason ε-removal is a crucial point in VAUCANSON.

The current implementation is generic but slow and takes a lot of memory. Another version of this algorithm proposed by Sylvain LOMBARDY might take less memory and be faster than the current one. In addition, another algorithm described by Mehryar MOHRI a researcher of AT&T Labs might be faster too.

These two algorithms have been implemented in VAUCANSON and this report describes their implementations in VAUCANSON and gives some results.

Keywords
Epsilon removal, shortest-paths algorithms, finite-state machine

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

Contents

1 Introduction **4**

2 Preliminaries **5**
- 2.1 Spontaneous transition . . . 5
- 2.2 ε-removal . . . 5
  - 2.2.1 Backward ε-removal . . . 5
  - 2.2.2 Forward ε-removal . . . 6
- 2.3 Weighted automata . . . 7
  - 2.3.1 Some mathematic notions . . . 7
  - 2.3.2 Evaluating a world on a weighted automaton . . . 7
  - 2.3.3 A problem with ε-cycles . . . 8

3 Current algorithm based on star-matrix **9**
- 3.1 First step: Transitive closure . . . 9
  - 3.1.1 Initialization . . . 9
  - 3.1.2 Transitive closure . . . 10
- 3.2 Second step: Forward or Backward ε-removal . . . 11
- 3.3 Problems and Solutions . . . 11
- 3.4 Future work . . . 12

4 Future algorithm based on shortest-paths **13**
- 4.1 First step: Shortest-path . . . 13
- 4.2 Second step: Forward or Backward ε-removal . . . 13
- 4.3 Generic shortest-path algorithm . . . 14

5 Benches **15**
- 5.1 $(a + \varepsilon)^n$ . . . 15

6 Conclusion **16**

Chapter 1

Introduction

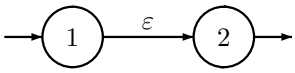
In VAUCANSON, lots of algorithms such as determinize or minimize, have to input an automaton without any ε-transition. Evaluating a word in an automaton without any ε-transition is faster. For these reasons, we often use ε-removal and we want a fast algorithm.
For these two reasons we had decided to improve the poor performance of the current algorithm. Two different methods had been tested, the first one was to add a strong connected components research. The second one was to replace the first part of the algorithm which computes the transitive closure. In the current algorithm we have a classical transitive closure on a matrix representing the automaton, we replaced it by a shortest-path algorithm.

Chapter 2

Preliminaries

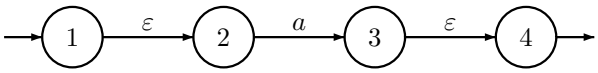
2.1 Spontaneous transition

A spontaneous transition permits to go from a state to another without reading any letter in the word which is checking. When we draw an automaton these transitions are labeled with an ε. For example, in the next automaton, we could be in the first state and in the second state in the same time due to ε-transition.



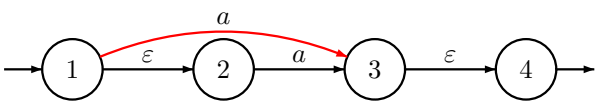
2.2 ε-removal

In this section we will see how removing ε-transitions in Boolean automata. We have two ways to do that the Backward and the Forward methods. We will apply these two methods on the automaton drawn below.

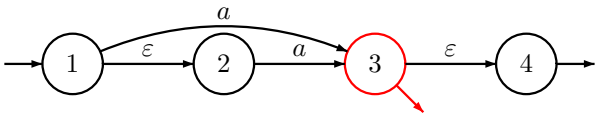


2.2.1 Backward ε-removal

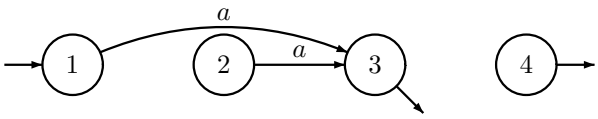
The first step is to add new transitions to pass through ε-transitions.



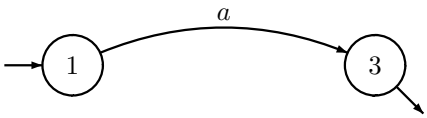
The second step is to add new final states before ε-transitions.



The third step is to remove all ε-transitions.

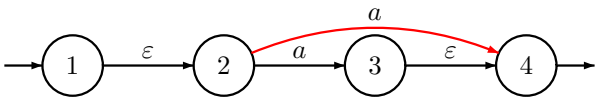


We could remove inaccessible states but it is not mandatory.

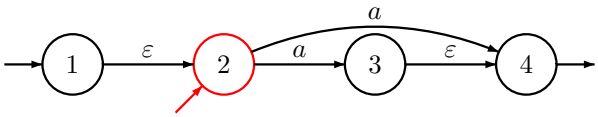


2.2.2 Forward ε-removal

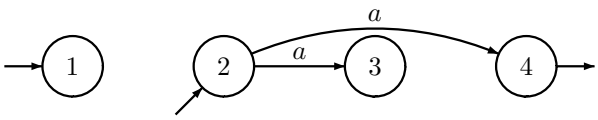
The first step is to add new transitions to pass through ε-transitions.



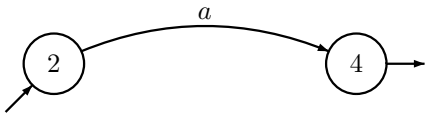
The second step is to add new initial states before ε-transitions.



The third step is to remove all ε-transitions.



We could remove inaccessible states but it is not mandatory.



2.3 Weighted automata

2.3.1 Some mathematic notions

Weighted automata are automata with weight on transitions. The algebraic structure of a semiring is required to well-define these weight.

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

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

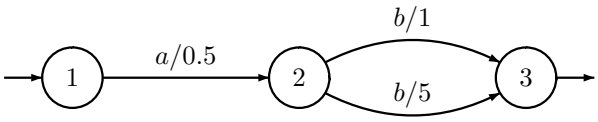
In this report, we will only work with the Real-Semiring $\mathcal{R} = (\mathbb{R}, +, \times, 0, 1)$ and the Tropical-Semiring $\mathcal{T} = (\mathbb{R}_+ \cup \{\infty\}, min, +, \infty, 0)$. But our algorithms work on other semirings.

Definition 2 *A weighted automaton* $\mathcal{A} = (\Sigma, Q, I, F, E, \lambda, \rho)$ *over a semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is defined as below:*

- Σ *is the finite alphabet of the automaton,*
- Q *is a finite set of states,*
- $I \subseteq Q$ *is the set of initial states,*
- $F \subseteq Q$ *is the set of final states,*
- $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ *is a finite set of translations,*
- $\lambda : I \to \mathbb{K}$ *the initial weight function mapping* I *to* $\mathbb{K}$,
- $\rho : F \to \mathbb{K}$ *the final weight function mapping* F *to* $\mathbb{K}$.

2.3.2 Evaluating a world on a weighted automaton

To evaluate the weight of a word through a weighted automaton, we perform the weight of each path labeled with this word, by using the $\otimes$ of the semiring between each transition weights. Then we use $\oplus$ of the semiring between the weight of each path.



In this automaton defined on the Real-Semiring the weight of the world "ab" is

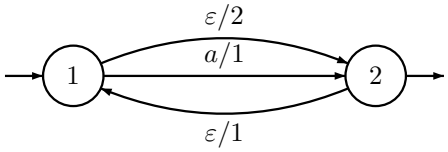
$$(0.5 \times 1) + (0.5 \times 5) = 3$$

and its weight on the Tropical-Semiring is

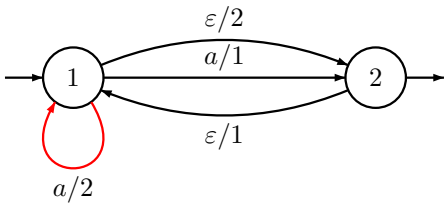
$$min((0.5 + 1), (0.5 + 5)) = 1.5$$

2.3.3 A problem with ε-cycles

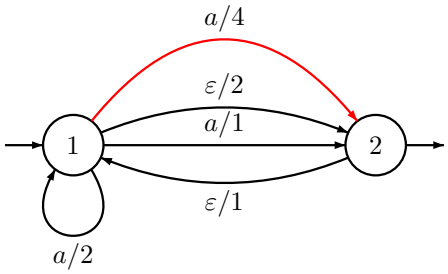
Now we have two good methods to suppress ε-transitions but we have another problem, these two "classical" methods do not work with a weighted automaton composed of ε-cycles. For example in the automaton below defined on the Tropical-Semiring, the previous algorithms will never finish.



Considering, the transition labeled with "a/1", this transition is followed by an ε-transition. We will add a transition to pass through this ε-transition.



The transition added is also followed by an ε-transition and we will also add a transition to pass through this ε-transition.



At this point, it is trivial that our algorithm will never finish. For these automata we have to find an algorithm which resolves these cases. For a human, it is easy to remark that if we follow the two ε-transitions infinitely at each turn we could go out of the automaton with a weight divided by two. The serie produced converges toward 1.

$$\sum_{k=2}^{\infty} \frac{1}{k} = 1$$

Chapter 3

Current algorithm based on star-matrix

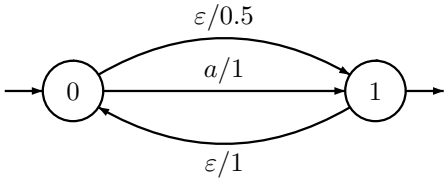
The current algorithm implemented in VAUCANSON works in two steps. During the first step we perform a transitive closure on the ε-transitions graph and then we perform a Forward or Backward closure on the automaton using the previous transitive closure.

3.1 First step: Transitive closure

3.1.1 Initialization

To perform the transitive closure just on the graph composed of ε-transitions, we construct a matrix of this graph. The complexity of this step is $\mathcal{O}(m)$ where m is the number of transitions in our automaton.

1: **for** each transition t **do**
2: **if** weight of ε in $t \neq 0$ **then**
3: $matrix$[src of t][dst of t] $\leftarrow$ weight of ε in t
4: **end if**
5: **end for**



	0	1
0	0	0.5
1	1	0

3.1.2 Transitive closure

After applying a transitive closure on a graph, if we had an ε-transition from a state p to a state q and an ε-transition from the state q to a state r, now we have a new ε-transition from the state p to the state r. The weight of this ε-transition is the product between weights of the two first transitions.
To apply this transitive closure we perform the star-matrix algorithm on our matrix. The complexity of star-matrix is $\mathcal{O}(n^3)$ where n is the number of states.

Algorithm 1 star-matrix

```
1:  for i ← 0 to matrix.size do
2:      w ← matrix[i][i]
3:      matrix[i][i] ← 1/(1−w)
4:      for j ← 0 to matrix.size do
5:          if j ≠ i then
6:              z ← matrix[j][i] ⊗ w
7:              matrix[j][i] ← z
8:              if z ≠ 0̄ then
9:                  for k ← 0 to matrix.size do
10:                     if k ≠ i then
11:                         matrix[j][k] ← matrix[j][k] ⊕ z ⊗ matrix[i][k]
12:                     end if
13:                 end for
14:             end if
15:         end if
16:     end for
17:     for j ← 0 to matrix.size do
18:         if j ≠ i then
19:             matrix[i][j] ← w ⊗ matrix[i][j]
20:         end if
21:     end for
22: end for
```

	0	1
0	0	0.5
1	1	0

$\xrightarrow{star}$

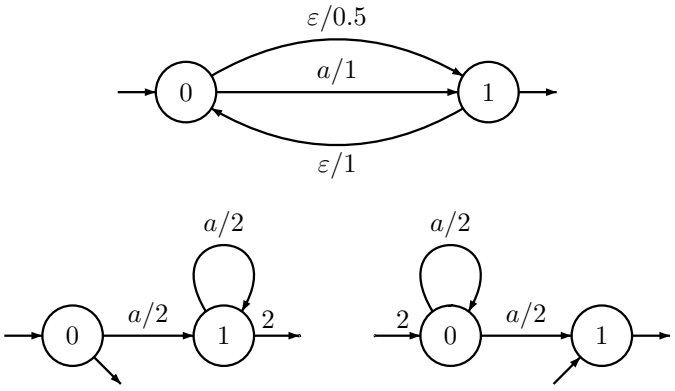
	0	1
0	2	1
1	2	2

3.2 Second step: Forward or Backward ε-removal

The second step is a classical Backward or Forward ε-removal as we explained in preliminaries but we search for ε-transition in the matrix and not in the automaton.

Algorithm 2 backward_remove

```
1:  for each_states s do
2:    vect_final[s] = final weight of s
3:  end for
4:  for each_states s do
5:    for each_transitions t to s do
6:      src ← src of t
7:      for i ← 0 to nb_states do
8:        w ← matrix[i][src]
9:        if w ≠ 0 then
10:         transition from k to s ⊗ w
11:       end if
12:     end for
13:     suppress t
14:   end for
15:   for i ← 0 to nb_states do
16:     w ← w ⊕ matrix[s][k] ⊗ vect_final[k]
17:   end for
18:   if w ≠ 0 then
19:     set final weight of s to w
20:   end if
21: end for
```



3.3 Problems and Solutions

This algorithm constructs a matrix which uses $n^2 \times s$ bytes where n is the number of states and s the size of the weight structure. With huge automaton we have a huge matrix even if our automaton have no ε-transition.
Sylvain LOMBARDY have proposed to find strong connected components before the first step and to apply star matrix just for these components. With this method we can remove ε-cycles and apply the "classical" algorithm which is really faster.

3.4 Future work

At the moment, we have a strong connected components algorithm which give us a list of set. Each set contains a strong connected component but we have some problems to create the matrix. To construct this matrix we must consider all ε-transitions between two states in the strong connected component and all ε-transitions incoming and outgoing from the component. But these two groups must be study separately. This step could be really costly in an automaton with large components.

Chapter 4

Future algorithm based on shortest-paths

4.1 First step: Shortest-path

Computing a shortest-path between all states gives the same results than a transitive closure but it is faster. Moreover, we use boost multi-index to simulate a sparse-matrix. It takes less memory and it could be good in some cases in which VAUCANSON had not enough virtual memory before. In this matrix we keep two values $dist$ is the distance between the two states and rel is the total weight added to $dist$ since the last time the state was extracted from the queue q.

4.2 Second step: Forward or Backward ε-removal

At the moment we use the second part of the first algorithm but it is not optimized. The version written during this seminar is not finished. Some bugs continues to trouble my nights. But with a new method we can gain time on this part, because currently we spend time to check is if a shortest path exists to apply our ε-removal. In the future version we will just apply the algorithm on shortest path calculated.

4.3 Generic shortest-path algorithm

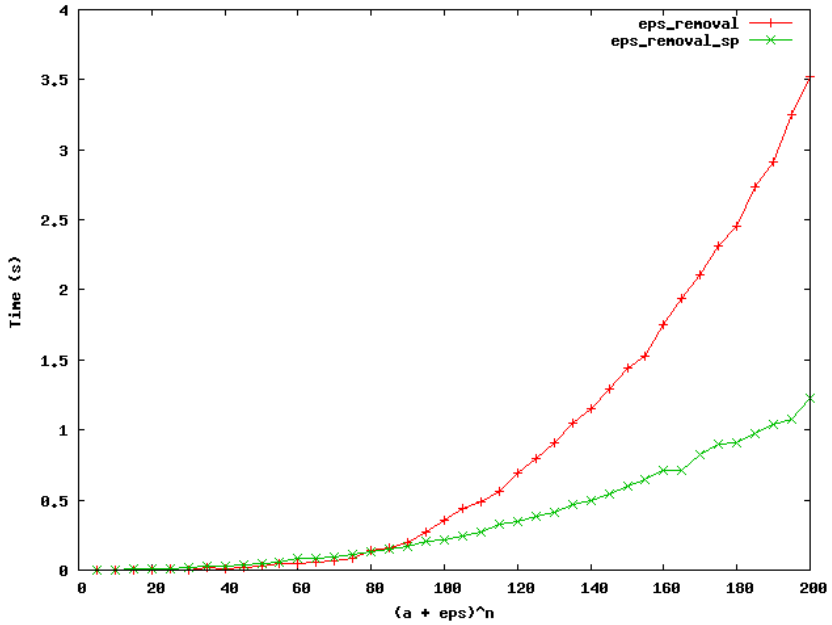
Algorithm 3 generic_shortest_path

```
1:  for each states s do
2:    shortest_path[s][s].dist ← 1̄
3:    shortest_path[s][s].rel ← 1̄
4:    push s into q
5:    while q not empty do
6:      curr ← front of q
7:      pop q
8:      R ← 0̄
9:      if shortest_path[s][curr].exist? then
10:       R ← shortest_path[s][curr].rel
11:       shortest_path[s][curr].rel ← 0̄
12:     end if
13:     for each ε-transitions t do
14:       dist ← 0̄
15:       if shortest_path[s][dst of t].exist? then
16:         dist ← shortest_path[s][dst of t].dist
17:       end if
18:       we = weight of ε in t
19:       if dist ≠ dist ⊕ we then
20:         if shortest_path[s][dst of t].exist? then
21:           create shortest_path[s][dst of t]
22:         end if
23:         shortest_path[s][dst of t].dist ← shortest_path[s][dst of t].dist ⊕ we ⊗ R
24:         shortest_path[s][dst of t].rel ← shortest_path[s][dst of t].dist ⊕ we ⊗ R
25:         if dst of t ∉ q then
26:           insert dst of t into q
27:         end if
28:       end if
29:     end for
30:   end while
31: end for
```

Chapter 5

Benches

5.1 $(a + \varepsilon)^n$



Regarding these results the algorithm based on shortest-paths is faster than the old one. It is due to transitive closure computed by shortest-paths method. Because in this version we use the second part of the old algorithm. The shortest-paths are stored in a hash table and a second part which computes backward or forward closure on reachable states only is not finished yet.

Chapter 6

Conclusion

We have included the algorithm based on shortest paths in VAUCANSON but some optimisations shall be added. The old algorithm with a research of strong connected components is almost finished. At this moment we can not choose between the two algorithms. Some benches shall be done and after them we will choose an algorithm.

References

Boost (2005). Boost documentation.

Mohri, M. (2000). Generic epsilon-removal algorithm for weighted automata. In *CIAA: International Conference on Implementation and Application of Automata, LNCS*.

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