VAUCANSON User's manual

The VAUCANSON group

July 28, 2006

Contents

Chapter 1

Installation

1.1 Getting Vaucanson

The latest stable version of the VAUCANSON platform can be downloaded from http://vaucanson.lrde.epita.fr/. The current development version can be retrieved from its Subversion[1] repository as follows:

```
user> svn checkout https://svn.lrde.epita.fr/svn/vaucanson/trunk vaucanson
```

1.2 Building Vaucanson

The following commands build and install the platform (the process takes an hour on a modern computer):

```
user> cd vaucanson-0.x
```

Then:

```
user> ./configure
user> make
root> make install
```

[1] Subversion can be found at http://subversion.tigris.org/.

Chapter 2

Vaucanson as a toolkit

VAUCANSON provides several programs that manipulate various types of automata. In this chapter we will learn how to use them. Actually there are 6 programs:

vcsn-b automata over the Boolean semiring $\mathbb{B}$;

vcsn-z automata over $(\mathbb{Z}, +)$;

vcsn-z-min-plus automata over $(\mathbb{Z}, min)$;

vcsn-z-max-plus automata over $(\mathbb{Z}, max)$;

vcsn-rt-tdc realtime transducers;

vcsn-tdc automata over free monoid products.

The first step to work with VAUCANSON toolkit is to choose on which type of automata you intend to work. Then use the proper program among those listed before.

All automata used in this chapter can be found in the **doc/manual/examples/** directory.

2.1 Boolean automata

This part demonstrates the use of the program `b`.

2.1.1 A first example

Let's consider the following Boolean automaton Figure 2.1. We will use VAUCANSON to compute the determinized automaton of A_1 and then minimize the resulting automaton.

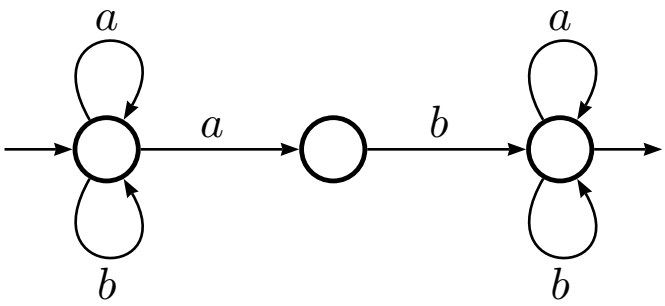


Figure 2.1: The automaton A_1

Determinization of A_1

To determinize a Boolean automaton, call the `determinize` function:

```
# vcsn-b determinize a1.xml > a1_det.xml
```

Now the file '`a1_det.xml`' contains the XML description of the determinized of the automaton A.

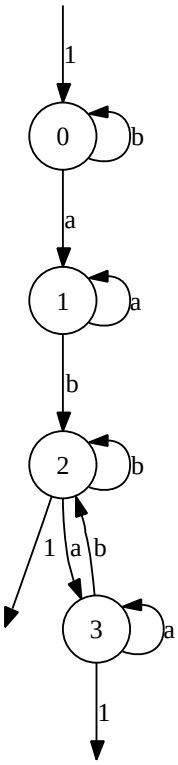
Visualizing

To get some information about the newly created automaton, call the `info` function:

```
# vcsn-b info a1_det.xml
States: 4
Transitions: 8
Initial states: 1
Final states: 2
```

Or use dotty to visualize our newly created automaton:

```
# vcsn-b display a1_det.xml
```



A { 4 states, 8 transitions, #I = 1, #T = 2 }

Minimizing

The minimal automaton can be computed the same way:

```
# vcsn_b minimize a1_det.xml > a1_min.xml
```

The commands can be composed with pipes from the shell, using '`-`' to denote the standard input.

```
# vcsn_b determinize a1.xml | vcsn_b minimize - > a1_min.xml
```

Evaluation

To evaluate whether a word is accepted:

```
# vcsn-b eval a1.xml 'abab'
1
# vcsn-b eval a1.xml 'bbba'
0
```

where 1 (resp. 0) means that the word is accepted (resp. not accepted) by the automaton.

2.1.2 Rational expressions and Boolean automata

VAUCANSON provides functions to manipulate rational expressions associated to Boolean automata. For instance, computing the language recognized by a Boolean automaton can be done using `aut_to_exp`:

```
# vcsn-b aut-to-exp a1.xml
(a+b)*.a.b.(a+b)*
# vcsn-b aut-to-exp a1_det.xml
b*.a.a*.b.(a.a*.b+b)*.(a.a*+1)
```

VAUCANSON provides several algorithms that build an automaton that recognizes a given language:

```
# vcsn-b standard "(a+b)*a.b.(a+b)*" | vcsn-b minimize -
```

computes the minimal automaton of '`(a+b)*ab(a+b)*`'.

2.1.3 Available functions

This section gives a brief definition of all functions that VAUCANSON provides for manipulating Boolean automata. All these algorithms are invoked using '`vcsn-b` *`algorithm-name`* `[`*`arguments`*`]`'. If the argument is replaced by '`-`' then the program will read an argument from the standard input. All algorithms dump their result to the standard output, except the "tests" functions that also return an exit status (0 if the test is successful, anything else otherwise).

In the following:

- *`a1`* and *`a2`* are two Boolean automata described in VAUCANSON XML format;
- `w` is a word, for example '`"aabb"`' if you are working on an alphabet that contains the letters '`a`' and '`b`';
- *`exp`* is a rational expression denoting a language;
- *`n`* is a nonnegative integer.

Input/output work with automata	
`define-automaton`	Define an automaton from scratch.
`edit-automaton` *`a1`*	Edit an existing automaton.
`info` *`a1`*	Print the number of states, transitions, initial and final states of *`a1`*.
`display` *`a1`*	Display the automaton using DOTTY.
`dump` *`a1`*	Dump the automaton to DOT format.
Tests and evaluation on automata	
`are-isomorphic` *`a1 a2`*	Test whether *`a1`* and *`a2`* are isomorphic.
`evaluation` *`a1 w`*	Test whether the word `w` is accepted by *`a1`*.
`is-deterministic` *`a1`*	Test whether *`a1`* is deterministic.
`is-empty` *`a1`*	Test whether *`a1`* accepts no word.

Generic algorithms for automata	
`accessible` *a1*	Extract the sub-automaton of accessible states of *a1*.
`co-accessible` *a1*	Extract the sub-automaton of co-accessible states of *a1*.
`trim` *a1*	Trim the automaton *a1*.
`transpose` *a1*	Compute the automaton accepting the mirror language of the one accepted by *a1*.
`closure` *[-bf] a1*	ε-removal algorithm. '`-b`' : backward closure '`-f`' : forward closure
`concatenate` *a1 a2*	Concatenate *a1* to *a2*.
`sum` *aut1 aut2*	Compute the sum of *a1* and *a2*.
`normalize` *aut1*	Compute an automaton with unique initial and final states, with ε-transitions.
`standardize` *aut1*	Compute an automaton with unique initial state without adding ε-transitions.
Generic algorithms for automata on letters	
`realtime` *[-bf] a1*	ε-removal algorithm and make every transition labeled by a letter. '`-b`' : backward closure '`-f`' : forward closure
`product` *a1 a2*	Compute the (Cartesian) product of *a1* and *a2*.
`power` *a1 n*	Compute the (Cartesian) product of *a1* by itself *n* times.
`quotient` *a1*	Compute the minimal automaton in bi-simulation with *a1*.
Algorithms specific to Boolean automata	
`determinize` *a1*	Compute the determinized automaton of *a1*.
`complement` *a1*	Compute an automaton that accepts the complement language of the one accepted by *a1*.
`minimize` *[-hm] a1*	Minimize the *deterministic* automaton *a1*. '`-h`' : use the Hopcroft algorithm '`-m`' : use the Moore algorithm
Conversion between automata and expressions	
`aut-to-exp` *a1*	Print a rational expression denoting the language accepted by *a1*.
`expand` *exp*	Partially expand rational expressions. For instance, expanding '`a(b+ab(a+b))`' will produce '`aab.(a+b)*+ab`'
`derived-term` *exp*	Compute the derived term automaton of *exp*.
`standard` *exp*	Compute the standard (Glushkov) automaton of *exp*.
`thompson-of` *exp*	Compute the Thompson automaton of *exp*.

2.2 Transducers

In VAUCANSON we distinguish two types of transducers, and therefore provides two programs:

tdc considering a transducer as a weighted automaton of a product of free monoid,

rt_tdc considering a transducer as a machine that takes a word as input and produce another word as.

Both views are equivalent and VAUCANSON provides algorithms to pass from a view to the other one.

2.2.1 Example

The realtime transducer T_1 (Figure 2.2) gives the quotient by 3 of a binary number and the transducer T_2 (Figure 2.3) adds 1 to a binary number.

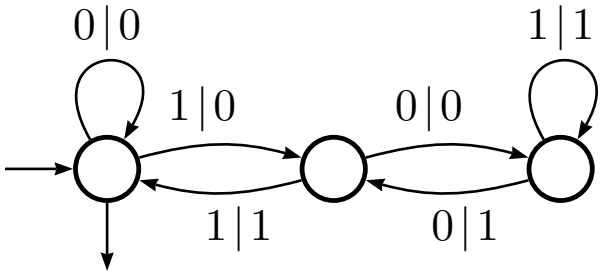


Figure 2.2: Realtime transducer T_1 computing the quotient by 3 of a binary number

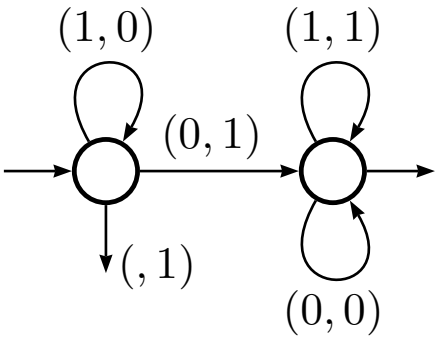


Figure 2.3: Transducer T_2 adding 1 to a binary number

Evaluation

```
# vcsn-rt-tdc evaluation quot_3_rt.xml '110'
0.1.0
```

Domain

The transducer T only accepts binary number which are divisible by 3 as input.

```
# vcsn-rt-tdc domain quot_3_rt.xml > divisible_by_3.xml
```

Now the file '`divisible_by_3.xml`' contains the description of a Boolean automaton that accepts only the numbers divisible by 3.

to-tdc

Each transucers can be transformed to the other type of transducer thanks to the `to-tdc` and `to-rt-tdc` functions.

```
# vcsn-rt-tdc to-tdc quot_3_rt.xml > quot_3.xml
# vcsn-tdc to-rt-tdc add1.xml > add1_rt.xml
```

Composing

```
# vcsn-tdc compose quot_3.xml add1.xml
```

2.2.2 Available functions

The following functions are available for both `vcsn-rt_tdc` and `vcsn-tdc` programs. To invoke them, run '*`program algorithm-name [arguments]`*'.

In the following:

- *`t1`* and *`t2`* are two transducers (either "realtime" or not) described in VAUCANSON XML format;
- `w` is a word, for example '`"aabb"`' if you are working on an alphabet that contains the letters '`a`' and '`b`';
- `a` is a Boolean automaton;
- *`t1_rt`* is a realtime transducer;
- *`t1_fmp`* is a transducer (seen as an automaton over a free monoid product).

Input/output work with transducers	
`define-automaton`	Define a transducer from scratch.
`edit-automaton` *t1*	Edit an existing transducer.
`info` *t1*	Print the number of states, transitions, initial and final states of *t1*.
`display` *t1*	Display the transducer using DOTTY.
Tests and evaluation on tranducers	
`are-isomorphic` *t1 t2*	Test if the two transducers are isomorphic.
`evaluation` *t1 w*	Compute the evaluation of *w* by *t1*.
`is-empty` *t1*	Test if *t1* realizes the empty relation.
Generic algorithm for tranducers	
`closure` *t1*	ε-removal algorithm.
`compose` *t1 t2*	Compute a tranducer realizing $f_2 \circ f_1$, where f_1 (resp. f_2) is the function associated to *t1* (resp. *t2*).
`domain` *t1*	Compute an automaton accepting all input accepted by the transducer t1.
`evaluation` *t1*	Compute the evaluation of w by t1.
`evaluation_aut` *t1*	Compute a Boolean automaton describing the words produced by the language described by a evaluated by t1.
`image` *t1*	Compute an automaton describing all output produced by the transducer t1.
`transpose` *t1*	Compute the transposed of the transducer t1.
`trim` *t1*	Compute the trimmed transducer of t1.
Algorithms for transducers	
`sub-normalize` *t1_fmp*	Compute the sub-nomalized transducer of *t1_fmp*.
`is-sub-normalize` *t1_fmp*	Test if *t1_fmp* is sub-normalized.
`composition-cover` *t1_fmp*	.
`composition-co-cover` *t1_fmp*	.
`b-compose` *t1_fmp t2_fmp*	Compose *t1_fmp* and *t2_fmp*, two unweighted normalized or sub-normalized transducers.
`to-rt-tdc` *t1_fmp*	Compute the equivalent realtime transducer of *t1_fmp*.
`intersection a`	Transform `a` in a fmp transducer by creating, for each word, a pair containing twice this word.

Algorithms for "realtime" transducers	
`realtime` *t1_rt*	Compute the realtime transducer of *t1_rt*.
`is-realtime` *t1_rt*	Test if *t1_rt* is realtime.
`to-tdc` *t1_rt*	Compute the equivalent fmp transducer of *t1_rt*.

2.3 Weighted automata

This part shows the use of the program `vcsn-z`, but all comments should also stand for the programs `vcsn-z_min_plus` and `vcsn-z_max_plus`.

2.3.1 Example

Let's consider the following $\mathbb{N}$-automaton, *i.e.* an automaton which label's weights are in $\mathbb{N}$:

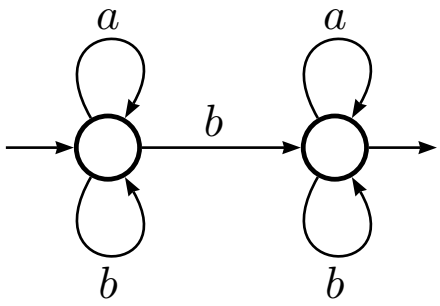


Figure 2.4: The automaton B_1

This time the evaluation of the word `w` by the automaton B_1 will produce a number, rather than simply accept or reject `w`. For instance let's evaluate '`abab`' and '`bbab`':

Evaluation

```
# vcsn-z eval b1.xml 'abbb'
3
# vcsn-z eval b1.xml 'abab'
2
```

The automaton B_1 "counts" the number of '`b`' contained in `w`.

Power

Now let's consider the B_1^n, where

$$B_1^n = \prod_{i=1}^{n} B_1, n > 0$$

```
# vcsn-z power b1.xml 4 > b1_4.xml
```

Now the file '`b1_4.xml`' contains the automaton B_1^4. Lets see what the evaluation of the words '`abab`' and '`bbab`' gives with this automaton:

```
# vcsn-z eval b1_4.xml 'bbab'
81
# vcsn-z eval b1_4.xml 'abab'
16
```

This time one can notice that the automaton B_1^4 returns the evaluation of B_1 at power 4.

Quotient

One drawback of doing successive products of an automaton is that it creates a lot of new states and transitions.

```
# vcsn-z power b1.xml 4 | vcsn-z info -
States: 16
Transitions: 97
Initial states: 1
Final states: 1
```

One way of reducing the size of our automaton is to use the "quotient" algorithm.

```
# vcsn-z power b1.xml 4 | vcsn-z quotient - | vcsn-z info -
States: 5
Transitions: 15
Initial states: 1
Final states: 1
```

2.3.2 Available functions

In this section you will find a brief definition of all functions for manipulating weighted automata. The following functions are available for both. They are called using `vcsn-z`, `vcsn-z_max_plus`, and `vcsn-z_min_plus` run as '*program algorithm-name* [*arguments*]'.

In the following:

- *a1* and *a2* are two weighted automata described in VAUCANSON XML format;
- *w* is a word, for example '`aabb`' if you are working on an alphabet that contains the letters '`a`' and '`b`';
- *exp* is a rational expression denoting a language;
- *n* is a nonnegative integer.

Input/output work with weighted automata	
`define-automaton`	Define an automaton from scratch.
`edit-automaton` *a1*	Edit an existing automaton.
`info` *a1*	Print the number of states, transitions, initial and final states of *a1*.
`display` *a1*	Display the automaton using DOTTY.
Tests and evaluation on weighted automata	
`are-isomorphic` *a1 a2*	Test if the two automata are isomorphic.
`evaluation` *a1 w*	Compute the evaluation of *w* by *a1*.
`is-empty` *a1*	.
Generic algorithms for automata	
`accessible` *a1*	Extract the sub-automaton of accessible states of *a1*.
`co-accessible` *a1*	Extract the sub-automaton of co-accessible states of *a1*.
`trim` *a1*	Trim the automaton *a1*.
`transpose` *a1*	Compute the automaton accepting the mirror language of the one accepted by *a1*.
`closure` *[-bf] a1*	ε-removal algorithm. '`-b`' : backward closure '`-f`' : forward closure
`concatenate` *a1 a2*	Concatenate *a1* to *a2*.
`sum` *a1 a2*	Compute the sum of *a1* and *a2*.
`normalize` *a1*	Compute an automaton with unique initial and final states, with ε-transitions.
`standardize` *a1*	Compute an automaton with unique initial state without adding ε-transitions.
Generic algorithms for automata on letters	
`realtime` *[-bf] a1*	ε-removal algorithm and make every transition labeled by a letter. '`-b`' : backward closure '`-f`' : forward closure
`product` *a1 a2*	Compute the (Cartesian) product of *a1* and *a1*.
`power` *a1 n*	Compute the (Cartesian) product of *a1* by itself *n* times.
`quotient` *a1*	Compute the $\mathbb{Z}$-quotient of *a1*.
Conversion between automata and expressions	
`aut-to-exp` *a1*	.
`expand` *exp*	Partially expand rational expressions.
`derived-term` *exp*	Compute the derived term automaton of *exp*.
`standard` *exp*	Compute the standard (Glushkov) automaton of *exp*.
`thompson-of` *exp*	Compute the Thompson automaton of *exp*.

2.4 Building your own automaton

Chapter 3

Vaucanson as a library

To be written.