Your first program in Vaucanson

May 17, 2005

Contents

1 Introduction

This document is designed to help beginners writing their first program in Vaucanson. For this purpose, the library provides some c++ headers that contain shortcuts for the basic usage of the library.

2 How to build an automaton

The following listing is a valid Vaucanson program:

```
1  #include <vaucanson/boolean_automaton.hh>
2  using namespace vcsn::boolean_automaton;
3  int main()
4  {
5    alphabet_t alphabet;
6    alphabet.insert('a');
7    alphabet.insert('b');
8    automaton_t a = new_automaton(alphabet);
9    hstate_t p = a.add_state();
10   hstate_t q = a.add_state();
11   a.set_initial(p);
12   a.set_final(q);
13   a.add_letter_edge(p, q, 'a');
14   a.add_letter_edge(q, p, 'b');
15   tools::dot_dump(std::cout, a, "automaton");
16 }
```

- ▷ **line 1 and 2**: To program standard boolean automata (the so-called acceptors), the user can include this shortcuts' header. The `using namespace` command makes all shorcuts directly available. Otherwise, the user has to prefix `vcsn::boolean_automaton::` to every types and functions.
- ▷ **line 5**: The first thing to do is to declare the alphabet we are working with. The type `alphabet_t` is predefined into the header. We obtain an object instance called `alphabet`.
- ▷ **line 6 and 7**: `alphabet` is an object instance that provides services. For example, we can insert 'a' and 'b' into the alphabet (other services can be consulted in the file `vaucanson/algebra/concept/a`
- ▷ **line 8**: The function `new_automaton` defined in the header takes an alphabet and returns a fresh empty automaton. Here, we store this automaton into the variable `a`.
- ▷ **line 9 and 10**: As an object instance, `a` provides services like the ability to create a new state. This state is characterized by a handler (concretely, a little integer). In Vaucanson, every handler for state has the `hstate_t` type.
- ▷ **line 11 and 12**: `a` can also change the status of its state. For example, the `set_initial` method mark a state as initial.
- ▷ **line 13 and 14**: We can define a transition between two states labelled by a letter using the method `add_letter_edge`. The other methods that the user can expect from an automaton are located in the file: `vaucanson/automata/concept/automata_base.hh`.
- ▷ **line 15**: Vaucanson provides different way to interact with the user. For example, we can use the `DOT` format to display the automaton 'a' with `dotty`.

2.1 How to compile a stand-alone program

In a shell, if your file is called `automaton.cc` and if vaucanson is installed on your system, type the following command:

```
% g++ automaton.cc -o automaton
```

Note: if your Vaucanson is not installed or if it is not installed into a standard location, add `-I the_vaucanson_directory/include` to your command.

To execute the program and to display the resulting automaton:

```
% ./automaton | dotty -
```

3 How to use standard algorithms

The second step is to test the algorithms of Vaucanson. For this purpose, the user can also include shortcut header. For example, in the following code, the program build an automaton and compute its associated deterministic automaton.

```
1  #include <vaucanson/boolean_automaton.hh>
2  #include <vaucanson/standard_algorithms.hh>
3  using namespace vcsn::boolean_automaton;
4  int main()
5  {
6    alphabet_t alphabet;
7    alphabet.insert('a');
```

```
8    alphabet.insert('b');
9    automaton_t a = new_automaton(alphabet);
10   hstate_t p = a.add_state();
11   hstate_t q = a.add_state();
12   hstate_t r = a.add_state();
13   a.set_initial(p);
14   a.set_final(r);
15   a.add_letter_edge(q, q, 'a');
16   a.add_letter_edge(p, q, 'a');
17   a.add_letter_edge(q, p, 'b');
18   a.add_letter_edge(q, r, 'b');
19   a.add_letter_edge(r, q, 'a');
20   a.add_letter_edge(r, r, 'b');
21   automaton_t a_det = determinize(a);
22   tools::dot_dump(std::cout, a_det, "det_automaton");
23 }
```

- **line 2**: This header is including several headers present in the `vaucanson/algorithms` directory.
- **line 21**: Algorithms are functions not object instance. So, they are called as functions.