Release notes for Vaucanson

Contact: vaucanson@lrde.epita.fr

This document describes major updates to the Vaucanson project.

Vaucanson 0.7 May 17, 2005

- Vaucanson works with GNU C++ 3.2, 3.3, 3.4 and ICC 8.1.
- Transducers seen as automata over a free monoid product are now available.

  Until now, transducers in Vaucanson could only be seen as automata with multiplicity in a series. It is now possible to manipulate transducers seen as automata over a free monoid product.

  – Two context headers have been written: `fmp_transducer` and `z_fmp_ransducer` (fmp stands for Free Monoid Product), which can be used in the same way as previous context headers.

  On top of the classical types defined in all context headers, they define several types needed when manipulating this type of transducers:

ALGEBRAIC SETS	ELEMENTS OF SET	LOW LEVEL IMPLEMENTATIONS
first_alphabet_t		
second_alphabet_t		
first_monoid_t	first_monoid_elt_t	first_monoid_elt_value_t
second_monoid_t	second_monoid_elt_t	second_monoid_elt_value_t

In addition to that, the contexts provide the user with the following functions:

```
// Create an empty automaton.
automaton_t new_automaton(first_alphabet_t, second_alphabet_t);

// Create a couple of words that can directly be given to
// add_series_edge() for instance.
series_set_elt_t make_couple(first_alphabet_t, second_alphabet_t,
                             std::string, std::string);
```

  – Some new algorithms have also been added:

    * `fmp_to_realtime` and `realtime_to_fmp`: Enables the user to switch from one view of transducers to another.
    * `normalized_composition`: Enables the composition of normalized and sub-normalized transducers seen as automata over a free monoid product. (edges with labels of types (a,b), (1,a) and (a,1)).

- New XML I/O system.

The former XML system based on a DTD grammar was replaced by a system based on XSD schema specification. The new system is backward-compatible with old XML documents, just change namespace reference to `http://vaucanson.lrde.epita.fr` and remove DTD node.

This new specification comes with full transducer support, and a set of default types for both automata and transducer. To declare a transducer on free monoid product, do as the following:

```
<transducer xmlns=‘‘http://vaucanson.lrde.epita.fr‘‘>
  <content>
    <states>
      <state name=‘‘s0‘‘/>
    </states>
    <transitions>
      <transition src=‘‘s0‘‘ dst=‘‘s0‘‘ in=‘‘a‘‘ out=‘‘b‘‘/>
    </transitions>
    <initials/>
    <finals/>
</transducer>
```

```
To load an automaton (for example on Z semiring) from a stream
(containing the XML document), do as the following::

#include <vaucanson/xml/XML.hh>
#include <vaucanson/z_automaton.hh>
using namespace vcsn;
using namespace vcsn::z_automaton;
using namespace vcsn::xml;
automaton_t a = new_automaton(alphabet_t());
std::cin >> automaton_loader(a, io::string_out(), xml::XML());


To dump automaton as an XML document, do::

#include <vaucanson/tools/xml_dump.hh>
...
tools::xml_dump(std::cout, a, "A name");
...

Or use the automaton_saver() function.
```

- Big cleaning in the graph implementation.

  A lot of superfluous operations were done in the former implementation. Cleaning the code provided huge performance improvement. The actual implementation has the same interface than the former one.

- Update context headers.

  More granularity has been added to context headers. To create one, you need to include desired files from the include/vaucanson/contexts directory in a specific order. Please refer to the sources for more details.

- A long-standing bug in the core of the library was corrected.

  Until now, some compiler optimisation that was believed to happen did not happen. Because of this, all Element instances had an overhead of at least a few bytes in their memory footprint, while in most cases it was not necessary. The code was rewritten to allow for this expected optimisation.

Vaucanson 0.6.1 October 26, 2004

- Vaucanson works with GNU C++ 3.2, 3.3, 3.4 and ICC 8.1.

- Vaucanswig is no longer enabled by default.

  To enable the compilation of Vaucanswig, run:

  ```
  ./configure --enable-vaucanswig
  ```

  Be warned: this compilation takes several hours on a modern computer.

- The demos in `src/demos/xml` were updated.

  The demo formerly named `algorithms` is now compiled for various semirings under the following names:

  `b`: Boolean semiring.

  `z`: usual semiring on Z.

  `r`: usual semiring on R.

  `z_max_plus`: tropical semiring with (max, +) on Z.

  `z_min_plus`: tropical semiring with (min, +) on Z.

  Furthermore, additional algorithms may be called from this demo: `transpose` and `eval`. A list of states may be provided to the `aut_to_exp` algorithm, thus allowing the elimination of states to be performed in a specified order.

  An additional `src/demo/xml/samples` directory was created, with some XML samples, and some programs that can generate XML samples.

- `expand()` definitively replaces `verbalize()`.

  The `verbalize()` function does not exist anymore. It is replaced by `expand()`, which was introduced in Vaucanson 0.6. A short description of `expand()` can be found below, in the description of Vaucanson 0.6.

Vaucanson 0.6 July 18, 2004

- Big cleanings in Vaucanson XML.

  A big work was done around Vaucanson XML, which should be fairly more usable now. To save an automaton in a XML representation, just include `vaucanson/xml/static.hh` and do:

  ```
  stream << vcsn::automaton_saver(auto,
                                  vcsn::io::string_out (),
                                  vcsn::xml::xml_loader ());
  ```

  To reload the automaton, you may do the opposite operation:

  ```
  stream >> vcsn::automaton_loader(auto,
                                   vcsn::io::string_out (),
                                   vcsn::xml::xml_loader ());
  ```

  Of course, if you want to get rid of `vcsn::io`, `vcsn::xml`, etc. you may do:

  ```
  using namespace vcsn::xml; using namespace vcsn::io;
  ```

Many more examples and utilities can be found in the directory `src/demos/xml`. Just browse the sources!

- Few more examples.

  A few programs were written for the CIAA 2004 conference. It is not an extraordinary or complex code, but it demonstrates how quick and easy it may be to use Vaucanson for assembling some algorithms and building simple automata. It may also be a good introduction to learn Vaucanson by practice.

  These examples are in `src/demos/ciaa`.

- New features in context headers.

  A new context header for automata over the tropical semiring with the "min" and "+" operators was added. Furthermore, some new functions and typedefs are declared in each header.

  If you want to use the context "foo", then include `vaucanson/foo.hh`. You will get the following types in the namespace `vcsn::foo`:

ALGEBRAIC SETS	ELEMENTS OF SET	LOW LEVEL IMPLEMENTATIONS
alphabet_t		letter_t
monoid_t	monoid_elt_t	monoid_elt_value_t
semiring_t	semiring_elt_t	semiring_elt_value_t
series_set_t	series_set_elt_t	series_set_elt_value_t
automata_set_t	automaton_t	automaton_impl_t
series_set_t	rat_exp_t	rat_exp_impl_t

  As the user, you will probably only be interested in `alphabet_t`, `automaton_t` and `rat_exp_t`, respectively for alphabets, automata and rational expressions.

  For transducers, you will get the following extra types:

ALGEBRAIC SETS	ELEMENTS OF SET	LOW LEVEL IMPLEMENTATIONS
output_semiring_t	output_semiring_elt_t	output_semiring_elt_value_t
output_series_set_t	output_series_set_elt_t	output_series_set_elt_value_t

  In addition to that, a context provides the user with the following functions:

```
automaton_t new_automaton(alphabet_t); // Create an empty automaton.
rat_exp_t   new_rat_exp(alphabet_t); // Create an empty rational expres-
sion.
rat_exp_t   new_rat_exp(alphabet_t, std::string); // Create a rational
                // expression and initial-
ize it. Ex: new_rat_exp(a, "a+b*");
automaton_t standard_of(rat_exp_t); // Build the standard automa-
ton of an exp.
automaton_t thompson_of(rat_exp_t); // Build the thompson automa-
ton of an exp.
rat_exp_t   aut_to_exp(automaton_t); // Build an exp from an automaton.
```

  If you want more algorithms, just browse the `vaucanson/algorithms` directory, or look in the HTML documentation.

Currently, the following contexts are available:

  - `boolean_automaton`
  - `z_automaton`
  - `r_automaton`
  - `z_max_plus`
  - `z_min_plus`
  - `boolean_transducer`

- Many bug fixes.

  As usual many bugs were fixed. Especially the quotient should be correct now. There were also some fixes in the `standard_of` algorithm, and `derivatives_automaton` now works with expressions that have right weights. The closure was rewritten and should be faster now.

  Beside from the algorithms a big bug was fixed in the algebra core of Vaucanson, which used to cause some problems when one wanted to use different alphabets in different automata in the same program.

  Also, rational expressions which weights are implemented as floats or doubles now work correctly.

- `verbalize()` is deprecated, use `expand()`!

  A new `expand()` function was created. It performs a simple expansion of a rational expression. For example:

```
expand(a(a+b))      =       aa+ab
expand(a(a+b*))     =       aa+a.b*
expand(a(a+(a+b*))) =       aa+a.(a+b)*
expand(a+(a(a+b))*) =       a+(aa+ab)*
```

  To use this function, include `vaucanson/algorithms/krat_exp_expand.hh` and call the `vcsn::expand()` function on an `Element< Series, rat::exp<M, W> >` (or a `rat_exp_t` if you are using a context).

  While this function behave exactly as verbalize on series which have a finite support, the latter is deprecated. It will probably be removed in next releases.

- `krat_exp_print()` does not exist anymore.

  The file `vaucanson/algorithms/krat_exp_print.hh` was removed. To print a rational expression with no extra parenthesis, just use the `<<` operator onto a C++ stream:

```
std::cout << exp << std::endl;
```

  To make this operator behave as in previous versions, send the right format onto the stream:

```
std::cout << setpm (MODE_ALL) << exp << std::endl;
```

  (`setpm` and `MODE_ALL` are in the `vcsn::rat` namespace). To have an exhaustive list of the different manipulators and print modes, you may look in `vaucanson/algebra/implementation/series/rat/dump_visitor`

- `tools/usual.hh` was removed.

  A system of context headers is used since vaucanson 0.4.2, and therefore `usual.hh` became useless. Furthermore it was a bad idea to use it since it has many includes and may slow a lot a compilation.

  This include has been removed from the distribution. if you were using some of its definitions (e.g. `usual_automaton_t`) you now need to use the context headers (e.g. include `boolean_automaton.hh` and use `automaton_t` in namespace `vcsn::boolean_automaton`). You will also find all the macros that may be needed in `tools/usual_macros.hh`.

Vaucanson 0.5 March 24, 2004

- New XML Input/Output system.

  An XML Input/Output system have been added to Vaucanson. To use it, you will need the Apache Xerces C++ library version 2.3.*. To enable the test suite on the XML I/O system, you need to use the configure option `--enable-xml`.

  `doc/xml/:` You can find a minimal documentation here.

  `xml/:` DTD and XSL files and some scripts.

  `include/vaucanson/xml/:` Header files.

  `src/test/xml/:` Test files. Can be used as examples.

- Better documentation.

  An effort has been made to make the Doxygen documentation look better. The documentation is still incomplete, and some errors probably remain, but it should be far more usable now.

  Especially, the "Algorithm" section of the documentation should now be exhaustive.

- Instantiation of Element which set is dynamic may fail when you do not specify the set.

  To get more safety at runtime, trying to instantiate an Element which set is dynamic without initializing the set will fail at compile time. This ensures every Element you will manipulate has its corresponding set associated. This has two consequences:

  - Using `Element::set()` should provoke no more segmentation faults or similar undesired behavior.
  - The `bound()` method has no sense anymore and therefore has been removed. If you make calls to this method in your programs, just consider it returns true every time now and remove the call.

- Many renamings.

  To get a more consistent interface, a few methods have changed. Consequently, your old code designed for Vaucanson 0.4 may not work properly with Vaucanson 0.5. However, converting it to the new nomenclature should be straightforward:

  - All names containing the word "`serie`" are now written with "`series`" instead. For example, `serie_get()` is now `series_get()`.
  - All `serie_t` typedefs are now `series_elt_t`.
  - The method `value_get()` for the elements of series is now called `get()`.
  - The method `value_set()` for the elements of series is now called `assoc()`.
  - The convenience files `vaucanson_*.hh` in the `include/vaucanson` directory have been stripped from their leading "`vaucanson_`".
  - `hopcroft_minimization.hh` is now named `minimization_hopcroft.hh` to be consistent with `minimization_moore.hh`.

- New implementation for numerical semirings.

  It is now possible to use rational numbers as an implementation in numerical semirings. In order to do so, the header `<vaucanson/algebra/concrete/semiring/rational_number.hh>` must be included. All you need to do then, is to declare a variable as follows:

  ```
  vcsn::Element<semiring_t, vcsn::algebra::RationalNumber> q(num, denom);
  ```

`semiring_t` can be any numerical semiring.

The usual operators have been overloaded, and you can get the integer or double value of the fraction by using `to_int()` or `to_double()`. You can also access the numerator with `num()` and denominator with `denom()`.

- Many more tests.

  The test suite has been extended and improved. Generic tests are now instantiated on many more types. Also, existing tests have been enriched with extra checks and non-existing tests have been written. Some bugs have been discovered, fixed, and now have their corresponding regression test.

  As a result, running "`make check`" should take more time than before, but now trusting a successful check sequence is less hazardous. Note that it is still possible to disable some tests by removing the test directory and running the configure script again.

  Tests have shown that using some rational expressions which weight are implemented with double can be potentially dangerous. This requires more checks before we can fix it. For the moment try to avoid doing that.

- Rewriting of the `minimization_moore()` algorithm.

  The minimization algorithm named `minimization_moore()` has been rewritten. It should be more readable and more reliable now.

Vaucanson 0.4 October 29, 2003

- Addition of an automata I/O subsystem.
- Nearly-complete SWIG bindings for algorithms in Vaucanswig.
- Preliminary documentation for Vaucanswig.
- Addition of a Bitset class which behave almost like a `std::set<int>`.
- Addition of a Window class to permit easy text manipulation.
- Addition of a generic search algorithm able to skip characters in the input stream.

Vaucanson 0.3 July 11, 2003

- More documentation.
- Addition of SWIG modules (Vaucanswig).
- Noticeable performance boost thanks to a working unification of references to structural elements.

Vaucanson 0.2 July 02, 2003

- First public release.
- Rewrite of the `fundamental` module.
- New graph structure to replace the old, legacy `ManyLinks` implementation.
- Rewrite of most algorithms.
- Implementation of algorithms on rational expressions.
- Initial Doxygen documentation efforts.

Vaucanson 0.1 January, 2002

- Initial release.