XML proposal for automata description

The Vaucanson group

July 28, 2006

Abstract

This paper presents an XML description format for the automaton representation. We introduce the proposal through some examples that enlight characteristic features of the format, with a progressive complexity. Finally, we briefly focus on implementation concerns.

Introduction

Conceiving a universal automaton exchange format aims at providing the community with a communication tool for the connection of the various programs that deal with automata and transducers. This system is used in the Vaucanson platform. Someone can load an automaton in Vaucanson from a XML file, or store an existing one from Vaucanson into an XML file.

1 Overview

As it will be described further, we use an XSD file [7] for the description format of this XML proposition, since it allows context sensitive declarations. This report first presents an XML representation of a classical Boolean automaton. Then this example will be fleshed out, so as to deal with transducers and more general automata with multiplicity.

The automaton description is structured in two parts. The `<label-type>` tag provides some automaton type definitions. This can be a Boolean automaton, or a weighted one with the ability to specify the weight type. It can also have some alphabet specifications, etc. The `<content>` tag provides the definition of the automaton "structure".

The visual representation of automata involves a very large amount of information. This is why two different types of information are distinguished in this proposition, described in the following two tags. The `<geometry>` data correspond to the embedding of the automaton in a plane. They represent the way the automaton is placed in it. The tag consequently contains information such as the coordinates of the states, or the directions and types of the transitions. The `<drawing>` data contain the definition of attributes that characterize the graphical aspects of the automaton's elements. Therefore, this tag contains information like the color of the states or the style of the transitions.

The proposed policy expects these properties to be checked by the program, and that it is not complicated nor more costly than to test whether an announced property is indeed fulfilled.

2 Simple examples with default types

2.1 A Boolean automaton

As a first example, the automaton of Figure 1 is represented in Figure 2. This Boolean automaton recognizes the set of words over the alphabet $\{a, b\}$ that contain at least one b.

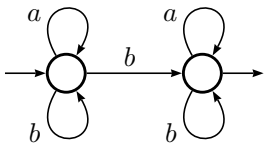

Figure 1: The automaton B_1

```
<automaton>
  <content>
    <states>
       <state name="s0"/>
       <state name="s1"/>
    </states>
    <transitions>
       <transition src="s0" dst="s0" label="a"/>
       <transition src="s0" dst="s0" label="b"/>
       <transition src="s0" dst="s1" label="b"/>
       <transition src="s1" dst="s1" label="a"/>
       <transition src="s1" dst="s1" label="b"/>
       <initial state="s0"/>
       <final state="s1"/>
    </transitions>
  </content>
</automaton>
```

Figure 2: The XML description of the automaton B_1

2.2 A Boolean transducer

The XML proposition can also be used to represent transducers. The example of Figure 3 gives the quotient by 3 of a binary number. It is represented in Figure 4.

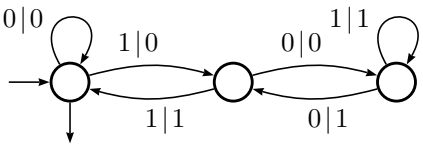

Figure 3: Transducer T giving the quotient by 3 of a binary number

```
<transducer>
  <content>
    <states>
       <state name="s0"/>
       <state name="s1"/>
       <state name="s2"/>
    </states>
    <transitions>
       <transition src="s0" dst="s0" in="0" out="0"/>
       <transition src="s0" dst="s1" in="1" out="0"/>
       <transition src="s1" dst="s0" in="1" out="1"/>
       <transition src="s1" dst="s2" in="0" out="0"/>
       <transition src="s2" dst="s2" in="1" out="1"/>
       <transition src="s2" dst="s1" in="0" out="1"/>
       <initial state="s0"/>
       <final state="s0"/>
    </transitions>
  </content>
</transducer>
```

Figure 4: The XML description of T

2.3 Naive description of the `<content>` tag

The `<content>` tag aims at describing the structure of the automaton. It has two children, both mandatory and expected in a specific order. These tags enable definitions of states and transitions.

The first tag, `<states>`, introduces the declaration of the set of states of the automaton. A state has three attributes: a `name` (which is mandatory and has to be unique), a `label` and a `number`. The latter can be used to put an ordering on states, or to add special integer data to the state.

The second tag, `<transitions>`, introduces the declaration of the set of transitions. The initial and final *transitions* are represented as children of `<transitions>`. Effectively, an initial state s can be seen as a transition which destination is s. This transition can have a `label` or a `number` in some cases, so it seems to be logical to have the list of initial states in the `<transitions>` tag. Similarly, the final states can be found at the same place in the XML description.

It is mandatory for a `<transition>` to have two attributes: `src` and `dst`, representing source and destination state names of the transition. In the case of an `<initial>` or `<final>` transition, the only mandatory attribute is `state`, referring to the initial or final state the transition belongs to.

There is no limitation of the format for the content of attributes, as it is a non-restricted string. For example, a user can store a rational expression in the `label`, `in` or `out` attributes of a transition. The use of these attributes depends on the structure of automaton one defines. When omitting them, the XSD grammar proposes the empty word as the default value.

At this point, most of automata can be easily described. These examples use only a part of the XML description that we present, but allow the reader to understand the basis of this format and to easily deal with a great amount of automata and transducers.

3 Description of the format

This part describes in details some tags of the XML format. Firstly, the tags `<automaton>` and `<transducer>` will be introduced. Then, the two main tags, namely `<label-type>` and `<content>`, will be described. Eventually, the main `<session>` tag that holds all the other ones will be presented.

3.1 The `<automaton>` and `<transducer>` tags

As one can see in the previous examples, Figure 2 and Figure 4, these tags specify the type(s) of the object(s) contained in the XML session. The content of an object is then specific, and it is linked to the type of tag that is used. In any case, an attribute "name", present in these tags, allows to bring an explicit name to an automaton and to store it in a XML file.

3.2 The `<label-type>` tag

In many cases, automata are graphs whose transitions are labeled by symbols called letters, taken in a set called alphabet. In full generality, this label can be a polynomial, or even a rational series, over a monoid with coefficients taken in a semiring. The `<label-type>` tag allows to refer to this semiring and this monoid.

In the automaton described in Figure 2, no specific information is given on the type of the automaton. The proposal comes with a set of predefined types, in order to limit amount of needed declarations for widely used structures. When the document starts with the `<automaton>` tag and when the `<label-type>` tag is omitted, the default automaton type is a Boolean automaton, on the standard alphabet (all the letters of the alphabet, including capitalized ones, and digits). Concerning the `<transducer>` tag without any `<label-type>` tag, the default transducer is Boolean with two monoids built on the same standard alphabet. These default types will be described further in the XML format.

3.2.1 The `<monoid>` tag

There are cases for which the default alphabet proposed to build the monoid doesn't fit. For this reason, this tag enables the user to determine the basic symbols set that she wants to use as an alphabet in the labels of the transitions.

For instance, in the current state, the automaton of Figure 1 is defined with the default alphabet. It could be better to set an alphabet that only contains the letters a and b, so as to prevent the user from possible errors subsequent to the first definition. So, the resticted alphabet would be defined as shown in Figure 5.

```
<label_type>
  <monoid>
     <generator value="a"/>
     <generator value="b"/>
  </monoid>
</label_type>
```

Figure 5: Setting $\{a, b\}$ alphabet

Similarly, one can also set a restriction on the alphabet like in Figure 6.

The user can set the string denoting the empty word with the attribute *identity_symbol* of the monoid. This way, the empty word symbol can always be different from any character of the used alphabet.

```
<label_type>
  <monoid generators="digits" type="free">
    <generator value="0"/>
    <generator value="1"/>
  </monoid>
</label_type>
```

Figure 6: Example of a restriction

To create a transducer based on a free monoid product, it is necessary to declare the two monoids in order to determine the two needed alphabets. A suitable example can be found in the Figure 12. It represents the type of a default transducer.

The XSD description of this tag is a little complicated. Some elements allow the proposition to be extensive and to fully describe any monoid type. This is why the first elements of the `<monoid>` tag are a choice. It can be one or more monoid types to allow some complex definitions (Figure 12 for an example), or it can be one or more generator types to describe the letters composing the alphabet of the monoid. But both types cannot exist in the same monoid type description. Concerning transducers, only two monoids can be defined under the main `<monoid>` tag so as to remain a free monoid product.

The monoid attributes are:

- `type`
  This is used to set the type of the monoid. Choices are `unit`, `free` or `product`.
- `generators`
  This attribute sets a global restriction to the alphabet. The current possibilities are letters, digits, pair or weighted.
- `identity_symbol`
  Used to set an empty word symbol.

The XSD description of the `<generator>` tag is:

- `value`
  This is used to add one letter in the alphabet. One can put several `<generator>` tags in a monoid description so as to have bigger alphabets.
- `range`
  This allows to set a fixed range without being obliged to add all the symbols one by one. For instance, the range ASCII sets the alphabet on the ASCII characters.

3.2.2 The `<semiring>` tag

The XML proposition enables a full description of the automaton type. It consequently proposes a way to write weighted automata or transducers seen as a weighted automaton with its weights in $Rat(B^*)$.

To describe a weighted automaton, the `<label-type>` tag provides a set of customizable tags to specify the type of multiplicities. The example of Figure 7 shows how to turn the automaton B_1 into a weighted automaton with weights in $\mathbb{Z}$ – so it counts the number of b in a word.

```
<automaton>
  <label-type>
     <semiring set="Z"/>
  </label-type>
  <content>
  ...
  </content>
</automaton>
```

Figure 7: The XML description of the $\mathbb{Z}$-automaton B_1

The `<semiring>` tag can be described with two attributes:

- `set`
  The set on which the automaton is built. The possible sets are $\mathbb{B}$, $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{N}$ and *ratSeries* (which will be discussed later).
- `operations`
  The type of operations that can be performed on this set. The possibilities are *numerical*, *boolean*, *tropicalMin* or *tropicalMax*.

For instance, describing the tropical semiring $(\mathbb{Z}, max, +)$ is achieved with:

```
<semiring set="Z" operations="tropicalMax">
```

All the content definition previously defined in Figure 2 is still totally compatible with a weighted automaton, and can remain unchanged.

Two different ways are proposed to set the weight of a transition. One can directly store the multiplicity in the `label` attribute, or use the dedicated `weight` attribute. These attributes can indistinctly be used in a `<transition>`, an `<initial>` or a `<final>` tag. When omitting the `weight` attribute, the XSD grammar proposes the identity of the semiring as the default value.

The `<semiring>` tag proposes some solutions for the transducers, like the example of Figure 8. It describes the right transducer for binary addition seen as a weighted automaton with weight in $Rat(B^*)$. `<monoid>` and `<semiring>` tags can recursively be defined, in order to describe a complex type. Only the `<label-type>` is shown in the example Figure 8 so as to remain clear.

```
<transducer>
  <label_type>
    <monoid generators="digits" type="free">
      <generator value="0"/>
      <generator value="1"/>
      <generator value="2"/>
    </monoid>
    <semiring set="ratSeries">
      <monoid generators="digits" type="free">
        <generator value="0"/>
        <generator value="1"/>
      </monoid>
      <semiring operations="numerical" set="B"/>
    </semiring>
  </label_type>
  <content>
    ...
  </content>
<transducer>
```

Figure 8: Right transducer for binary addition

The beginning of the XSD description of the `<semiring>` is a sequence that contains two elements, namely the monoid and the semiring. This allows a recursion in the definition of an automaton structure.

3.3 The `<content>` tag

For automata and transducers, the `<content>` tag has the same structure. The following is the description of this tag. Bold elements are mandatory. The special tags `<geometry>` and `<drawing>` can be at any place of the document. They are ignored here, but more information can be found in Section 4.

The first tag, `<states>`, gives the possibility to fully describe the states of an automaton and their content. The `<states>` tag is composed of:

- `<state>`
  This tag represents one state. There must be as many of these tags as there are states in the automaton. Such a tag has the following elements and attributes:
  - `name`
    The name to the state.
  - `label`
    Optional label for a state.
  - `number`
    Used if one wants to set an order on the states.

With the second tag, `<transitions>`, one can describe the transitions. It is composed of:

- `<transition>`
  This tag describes the content of one transition of the automaton. It is composed of:

  - `src`
    The state that is the source of the transition.
  - `dst`
    The state that is the destination of the transition.
  - `name`
  - `label`
  - `weight`
    Optional name, label and weight for the transition.

- `<initial>`
  This tag describes the content of an initial state of the automaton:
  - `state`
    The state that is initial.
  - `label`
  - `weight`
    Optional label and weigth for the entering transition.

- `<final>`
  This tag describes the content of a final state of the automaton. Its strucure is the same as the `<initial>` tag's one.

Concerning transducers, the `<content>` tag follows the same structure as for automata description, although a noticeable difference is the extension for transition definitions. Two new attributes are proposed for transducer description: `in` and `out`, respectively corresponding to the input and the output of a transition. These two attributes are proposed in addition to the classical `label` and `weight` attributes, that can still be used for transducer description. Of course, they can be used indistinctly in `<transition>`, `<initial>` or `<final>`.

3.3.1 About the labels

There is a specificity about the attribute "label" of the `<transition>` tag. Most of the time, there isn't any problem since its value is a single character. But one can prefer to set a rational expression denoting the language that must be recognized to pass through the given transition. Its type is a non-restricted string, but there is a grammar to follow so as to be correctly understood by VAUCANSON. It is presented in Figure 9, The empty word is represented by '`1`', and the absorbent element of the semiring by '`0`'. These are default values in VAUCANSON.

```
exp ::= '(' exp ')'
    |   exp '+' exp
    |   exp '.' exp
    |   exp exp
    |   exp '*'
    |   weight ' ' exp
    |   exp ' ' weight
    |   0
    |   1
    |   word
```

Figure 9: Grammar of the rational expression

Priority for operators is, from the most important to the least important:

- '*' (star), to star a series.
- ' ' (space), to weight a series either on the right or on the left.
- '.' (dot), to concatenate two series.
- '+' (plus), to do the union of two series.

The `in` and `out` attributes follow the same rules for transducers.

The VAUCANSON group is aware that with some special alphabets and a special empty word, some errors can occur when parsing some complicated labels. It would be preferable to design another XML description to represent a rational expression instead of a non-limited string, and it could be one of the further works of the VAUCANSON group.

3.4 The <session> tag

A way to manipulate many automata would be to combine them in a single document. The proposal offers this feature through the `<session>` tag. An unlimited number of automata or transducers can be combined in a single XML document. A XML session can also be named. An application can be found in Figure 10.

```
<session name="session1">
  <automaton name="a1">...</automaton>
  <transducer name="t1">...</transducer>
  <transducer name="t2">...</transducer>
</session>
```

Figure 10: A session with several automata

3.5 The cascade of default options

The `<label-type>` tag has two children: the `<monoid>` tag and the `<semiring>` tag. None of these tags is mandatory, and both have different values according to the root tag. Figure 11 shows the equivalent XML code if one omits the `<label-type>` tag when declaring an automaton. Similarly, Figure 12 shows the default type for transducers. The `operations` attributes is set to `"numerical"`, which means that usual laws over $\mathbb{B}$ shall be applied.

```
<label_type>
  <monoid type="free" generators="letters">
     <generator range="ascii"/>
  </monoid>
  <semiring set="B" operations="numerical"/>
</label_type>
```

Figure 11: Default type for an automaton

```
<label_type>
  <monoid type="product">
     <monoid type="free" generators="letters">
       <generator range="ascii"/>
     </monoid>
     <monoid type="free" generators="letters">
       <generator range="ascii"/>
     </monoid>
  </monoid>
  <semiring set="B" operations="numerical"/>
</label_type>
```

Figure 12: Default type for a transducer

4 The visualization tags

The visual representation of automata involves a very large amount of information. On the one hand, the `<geometry>` is context sensitive with data such as state coordinates or transition type. It gives some information about the way the automata are set in the plane only. On the other hand, the `<drawing>` tag gives some graphical information about the way the automata of a session must be drawn.

Let us note that these tags can be used at any level of the document. In this case, the defined properties are applied to the tag in which they are defined, and to its children. It is possible to define some properties in a tag and to locally override them in a child tag. For example in Figure 13, the filling color of the states is globally set to *black*, and the color of state s_1 is locally set to red. As a result, s_0 and s_2 will be black, and s_1 will be red.

```
<transducer>
  <drawing stateFillColor="black"/>
  <content>
     <states>
        <state name="s0"/>
        <state name="s1">
            <drawing stateFillColor="red"/>
        </state>
        <state name="s2"/>
      ...
</transducer>
```

Figure 13: Example of overriding drawing properties

4.1 The <geometry> tag

The `<geometry>` tag is context sensitive. If it is a child of the `<state>` tag, the only two properties that can be set are the position, `x` and `y`, of the state. These values can only be numeric.

If it is a child of `<transition>`, `<initial>` or `<final>`, two attributes can be set. Firstly, the `transitionType` attribute, that assigns the type of the transition (*line*, *arcL*, *arcR*, *curve*). Then, the `direction` attribute, that can be used to assign the

direction angle of a loop, for instance. This attribute is numeric.

The example of Figure 14 sets a global offset for the document, and then places the states in the plane. It also sets the type of the transition as a left arc.

```
<automaton>
  <geometry x="-2" y="-2"/>
  <content>
     <states>
        <state name="s0"><geometry x="0" y="0"/>
        </state>
        <state name="s1"><geometry x="3" y="0"/>
        </state>
     <transitions>
        <transition src="s0 dst="s1" label="a">
          <geometry transitionType="arcL">
        </transition>
     </transitions>
</automaton>
```

Figure 14: Setting geometry properties

4.2 The `<drawing>` tag

The `<drawing>` tag contains the definition of attributes that characterize the actual drawing of the graph. Most of them are indeed implicit and provided by drawing programs; the format only provides the possibility to make them explicit. A lot of different properties, that have been taken from the options proposed by Vaucanson-G, can be used in the `<drawing>` tag at many places in the proposal.

Since it's not possible to exhaustively name all needed attributes users may need, the proposal offers a limited set of properties. For example, *stateFillColor* or *transitionStyle* usage are shown in Figure 15. These attributes use a string representation to describe their values.

One of the powerful features of XSD files is the *anyAttribute* modifier. This modifier allows the user to easily extend the main XSD, and then use its own attributes and still be compliant with the grammar. The `<drawing>` tag contains a *anyAttribute* modifier in the proposal, so the grammar is not limited to a specific set of drawing properties.

```
<transducer>
  <geometry x="-5" y="0"/>
  <drawing stateFillColor="black" transitionStyle="dashed"/>
  <content>
     <states>
        <state name="s0">
            <drawing stateFillColor="red"/>
        </state>
      ...
</transducer>
```

Figure 15: Setting drawing properties

5 From DTD to XSD

The most important difference with our previous proposal [3] is the change from a DTD (Document Type Definition) document to an XSD (XML Schema Description) Schema.

It is desirable to keep the description of automata simple when describing widely used structures while giving the possibility to describe the most complex ones. For XML, this simplification enables to have default types, in order to omit `<label-type>` tag when describing common Boolean automata or transducers.

The problem then arises when describing an automaton or a transducer, the default values for the `<label-type>` tag must of course be different. This is not possible with a DTD description. The use of an XSD overcomes this difficulty, since it is possible to define different properties for a same element, according to the embracing context. It is so possible to locally change the behavior of a tag, and make it context-sensitive. With this feature, default values for the `<label-type>` tag are achieved, whether it is a child of `<transducer>` or `<automaton>`.

For information about the XSD Schema are available on the *World Wide Web Consortium* website [5].

6 Conclusion

For the past year we experimented the proposal made at CIAA'04 in the Vaucanson platform. This version 0.3 comes as a result of this experiment, with simplifications where possible. Thus, the Vaucanson platform deals with numerous automata types, and it is important to be able to define precisely the type of the automaton in addition to its content.

This proposal comes as a combination of two needs, shorten declaration of widely used structure and make possible definitions of complex types. We hope to have proposed a description format that fulfills, at least partially, both needs.

References

[1] Gamma E., Helm R., Johnson R., and Vlissides J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[2] Lombardy S., Régis-Gianas Y., and Sakarovitch J., Introducing Vaucanson *Theoretical Comput. Sci. 328* (2004), 77–96. Journal version of *Proc. of CIAA 2003, Lect. Notes in Comp. Sc. 2759*, (2003), 96–107 (with R. Poss).

[3] Claveirole T., Proposal: an XML representation for automata, Technical report, LRDE (2004).

[4] `http://xml.apache.org/xerces-c/`

[5] `http://www.w3.org/XML/Schema`

[6] `http://vaucanson.lrde.epita.fr/XML`

[7] `http://www.lrde.epita.fr/dload/vaucanson/vaucanson.xsd`