XML proposal for automata description

The VAUCANSON group

October 12, 2007

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 would enable someone to load an automaton in any automata handling program from a XML file, or to store an existing one from this program into an XML file.

1 Overview

As it will be described further, we use an XSD file (3) for the description format of this XML proposition, since it fits our needs. This report first presents an XML representation of a classical Boolean automaton and a classical transducer. Then the format of the proposal will be described in details so as to enable the reader to deal with more general automata with multiplicity.

The automaton description is structured in two parts. The `<labelType>` 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

The following examples aim at showing how some simple automata and transducers can be defined in XML. The description and details about each node tag will be given in the next sections.

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.

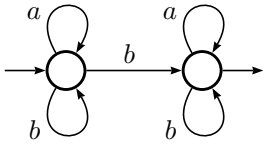

Figure 1: The automaton B_1

```
<automaton>
  <labelType>
    <monoid type="free" generators="letters">
       <generator range="implicitAlphabet"/>
    </monoid>
    <semiring set="B" operations="classical"/>
  </labelType>
  <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.

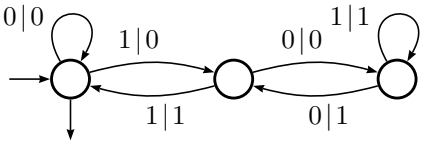

Figure 3: Transducer T giving the quotient by 3 of a binary number

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 the automaton states. A state has three attributes: a `name` (which is mandatory and has to be unique), a `label` and a `number`. The second one can be used in some special cases for which states have to store labels. 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.

The format has no limitation concerning 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 string 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 `<automaton>` tag will be introduced. Its two main tags, namely `<labelType>` and `<content>`, will also be described. Then, we will discuss the possible ways to describe transition labels, notably with the `<label>` tag. Eventually, the main `<session>` tag that holds all the other ones will be presented.

3.1 The `<automaton>` tag

As one can see in the previous examples Figure 2 and Figure 4, this tag is the main one and delimits the definition of one single automaton. Each automaton contains a

```
<automaton>
  <labelType>
    <monoid type="product">
       <monoid type="free" generators="letters">
         <generator range="implicitAlphabet"/>
       </monoid>
       <monoid type="free" generators="letters">
         <generator range="implicitAlphabet"/>
       </monoid>
    </monoid>
    <semiring set="B" operations="classical"/>
  </labelType>
  <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>
</automaton>
```

Figure 4: The XML description of T

`<labelType>` and a `<content>` tag. An attribute `name`, present in this tag, allows to bring an explicit name to an automaton when stored in an XML file.

3.2 The `<labelType>` 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 `<labelType>` 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 predefined type, in order to limit amount of needed declarations for widely used structures. When the document starts with the `<automaton>` tag and when the `<labelType>` 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). This default type will be described further in the proposition.

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 restricted alphabet would be defined as shown in Figure 5.

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

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

Similarly, one can also set a restriction on the alphabet like in Figure 6, where only digits are allowed.

```
<labelType>
  <monoid generators="digits" type="free" identitySymbol="e">
    <generator value="0"/>
    <generator value="1"/>
  </monoid>
</labelType>
```

Figure 6: Restriction and identity symbol

The user can set the string denoting the empty word with the attribute `identitySymbol` of the monoid. This way, the empty word symbol can always be different from any character of the used alphabet. But the XML format cannot check if the user uses the same character for both an element of the alphabet and the identity symbol. So, the user has to set properly these values, and the program using this format should check such properties.

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 3.

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 (**??** 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.

To sum up, the monoid attributes are:

- `type`
  This is used to set the type of the monoid. Choices are:
  - `unit`, when the monoid is built on only one element;

  - `free`, when dealing with a free monoid;
  - `product`, when the monoid is itself a monoid product.

- `generators`
  This attribute sets a global restriction to the alphabet. The current possibilities are:
  - `letters`, if the only possible values are taken from the alphabet;
  - `digits`, if one only wants digits;
  - `pair`, if the elements of the alphabets are pairs;
  - `indexed`, if each element of the alphabet has to be associated to something, like a weight for instance.
- `identitySymbol`
  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 obtain bigger alphabets.
- `range`
  This allows to set a fixed range without being obliged to add all the symbols one by one. Possible values are:
  - `implicitAlphabet`, that sets the alphabet on the lowercase and uppercase alphabet characters;
  - `digits`, that inserts in the alphabet the digit characters;
  - `ascii`, that sets the alphabet on the ASCII characters.

3.2.2 The `<semiring>` and `<numericalSemiring>` tags

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 `<labelType>` 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 that it counts the number of b in a word.

```
<automaton>
  <labelType>
     <semiring set="Z"/>
  </labelType>
  <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 values are *B, R, Z, N*, for the corresponding sets.
- `operations`
  The type of operations that can be performed on this set. The possibilities are:
  - `classical`, for the classical operations of a semiring, namely $(+, \times)$;
  - `minPlus`, for a semiring with its operations set to $(min, +)$;
  - `maxPlus` for $(max, +)$.
  - `minMax` for (min, max).

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

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

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 transducers. In the example of Figure 8, It is seen as a weighted automaton with its weights in $Rat(B^*)$. Only the `<labelType>` is shown so as to remain clear. To make this definition possible, the `<numericalSemiring>` tag has been defined. It can be used instead of the `<semiring>` one to introduce a more complex set. In this tag, one can define a monoid and a semiring so as to obtain a transducer.

```
<automaton>
  <labelType>
    <monoid generators="digits" type="free">
      <generator value="0"/>
      <generator value="1"/>
      <generator value="2"/>
    </monoid>
    <numericalSemiring>
      <monoid generators="digits" type="free">
        <generator value="0"/>
        <generator value="1"/>
      </monoid>
      <semiring operations="classical" set="B"/>
    </numericalSemiring>
  </labelType>
  <content>
    ...
  </content>
</automaton>
```

Figure 8: Right transducer for binary addition

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. It 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 weight for the entering transition.
- <final>
  This tag describes the content of a final state of the automaton. Its structure 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>.

3.4 Dealing with labels

3.4.1 About the label attributes

A label as an attribute of a transition or a state can be useful. It can keep an XML description of an automata human-readable and more simple to write. But this system comes with its limitations. Written as a string, a label automatically comes with a grammar to parse its content. Consequently, using attributes should only be done with very simple labels. If not, two programs using different grammars would surely be unable to parse a complicated label the same way. To avoid such errors, a label node has been defined.

3.4.2 The `<label>` tag

This proposal now possesses a strict definition for labels. The `<transition>` and `<state>` tags now have an inner element named `<label>` that enables to define the content of transitions or states. Its structure represents the grammar of the regular expressions that the label node can contain.

A label element can contain one of the following ones:

- `<star>`
  This tag is used to "starify" the expression inside it.
- `<sum>`
  This tag is used if the label to be defined is the union of two expressions. For this reason, it can contain two tags amongst those that define the label node.
- `<product>`
  This tag has the same properties as the `<sum>` one. It is used if the label is a product of expressions, *i.e* their concatenation. It is represented by "." in the explanation of the use examples.
- `<word>`
  First of the three possible terminal elements of the label grammar, this tag is used to define a word. It has a mandatory attribute `value` that must be valid elements taken from the automaton alphabet.
- `<zero>`
  This tag represents the zero word. The result of an evaluation on such an element will always return false.
- `<identity>`
  This tag represents the empty word. It is similar to a label with an empty string value, or with a value equal to the optional `identitySymbol` attribute of the `<monoid>` node.

A `weight` attribute can be added to each of these adds. It enables the user to set a weight to any element of the expression.

The XSD structure of the label has been designed so as to follow the grammar of the expected expressions. One can easily notice that this definition allows to define a complicated expression, recursively using the different tags.

In figure Figure 9, some examples of use of the `<label>` tag are shown. It is easy to understand the label values of the three examples, namely *a*, *(3 a+b)** and *a+(b.(a.c))*.

as for the `label` attribute, the same system has been enabled for the `in`and `out` ones. Indeed, one can define the input and the output of a transition using the `<in>` and `<out>` tags. If the labels are described this way in an XML file, one can now be sure that the labels will be perfectly understood, whatever the program that deals with.

```
<label>
  <word value="a"/>
</label>

<label>
  <star>
    <sum>
      <word value="a" weight="3"/>
      <word value="b"/>
    </sum>
  </star>
</label>

<label>
  <sum>
    <word value="a"/>
    <product>
      <word value="b"/>
      <product>
        <word value="a"/>
        <word value="c"/>
      </product>
    </product>
  </sum>
</label>
```

Figure 9: Examples of use of the `<label>` tag

3.5 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 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>
  <automaton name="a2">...</automaton>
  <automaton name="a3">...</automaton>
</session>
```

Figure 10: A session with several automata

3.6 The default value of the <labelType> tag

The <labelType> tag has two children: the <monoid> tag and the <semiring> tag. None of these tags is mandatory, and both have a default value. Figure 11 shows the equivalent XML code if one omits the <labelType> tag when declaring an automaton. In this case, the automaton alphabet is composed of all the letters of the alphabet. The operations attribute is set to *classical*, which means that usual laws over $\mathbb{B}$ shall be applied.

```
<labelType>
  <monoid type="free" generators="letters">
     <generator range="implicitAlphabet"/>
  </monoid>
  <semiring set="B" operations="classical"/>
</labelType>
```

Figure 11: Default type for an automaton

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 12, 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 12: Example of overriding drawing properties

4.1 The `<geometry>` tag

The `<geometry>` tag is context sensitive. It can have some different attributes according to the tag in which it is used. Four different tags exist.

- If this tag is a child of the `<state>` tag, the only two properties that can be set are the positions x and y of the state. These values can only be numeric.
- If it is a child of `<transitions>` or `<transition>`, three attributes can be set. Firstly, the `transitionType` attribute, that assigns the type of the transition. Values can be:
  - *line* for a straight transition
  - *arcL* and *arcR* for left and right arcs
  - *curve* for more complicated curved transitions. This value can be used for automata with a big amount of transitions that must not cross over each other.
  - *loop* for transitions of which destination state is its origin.

  Secondly, if the `transitionType` attribute is set to *loop*, the `loopDirection` can be set. Its value, between 0 and 360, indicates the angle of the transition.
  Lastly, one can set the position of the label on the transition. the `labelPosition` indicates the place where it should be on it. It is seen as a percentage, from the origin state to the destination state. For example, one may want to move the label of a transition near the origin state to keep clear the visualization of a big automaton. Setting the `labelPosition` to around 20 enables this.

- For `<initial>` and `<final>` tags. Another attribute, `typeShowing`, indicates if the initial(final) states have to be circled or to have an input(output) transition. If the choice is a transition, its angle can be set through the `transitionAngle` attribute. And as for transitions, the `labelPosition` attribute exists.
- For `<session>` and `<automaton>` tags, the global position of the automaton can be set.

The example of Figure 13 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 13: 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.

Since it's not possible to exhaustively name all needed attributes users may need, the proposal offers a very limited set of properties. For example, `stateFillColor` or `stateLineStyle` usage are shown in Figure 14. These attributes use a string representation to describe their values. Providing more attributes will be one of our future works.

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.

5 Using an XSD file

Some specifications of the XML format have led us to describe it using 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 `<labelType>` tag when describing common Boolean automata or transducers.

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

Figure 14: Setting drawing properties

The problem then arises when describing an automaton or a transducer, the default values for the `<labelType>` 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 `<labelType>` 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 (W3C).

6 Conclusion

For the past years we experimented the different proposals in the VAUCANSON platform. It begun with a first one in 2004 (5) and we presented our progress at CIAA'05 (1). This version 0.4 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.

The particularity of this version is the removal of the <<>transducer¿ tag and its corresponding default values. The proposal is consequently less readable, but it is more explicit and closer to the mathematical model of automata.

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.

(1) (5) (3) (W3C)

References

[1] Claveirole, T., Lombardy, S., O'Connor, S., Pouchet, L.-N., and Sakarovitch, J. (2005). Inside Vaucanson. In Springer-Verlag, editor, *Proceedings of Implementation and Application of Automata, 10th International Conference (CIAA)*, volume 3845 of *Lecture Notes in Computer Science Series*, pages 117–128, Sophia Antipolis, France.

[2] Lombardy, S., Régis-Gianas, Y., and Sakarovitch, J. (2004). Introducing Vaucanson. *Theoretical Computer Science*, 328:77–96.

[3] VAUCANSON group. Last version of the proposal XSD file. `http://www.lrde.epita.fr/dload/vaucanson/vaucanson_current.xsd`.

[4] Vaucanson group. Vaucanson XML system page. `http://vaucanson.lrde.epita.fr/XML`.

[5] Vaucanson group, T. (2004). Proposal: an XML representation for automata. Technical Report 0414, EPITA Research and Development Laboratory (LRDE), France.

[w3c] w3c. xsd information on the *World Wide Web Consorsium* website. `http://www.w3.org/XML/Schema`.