XML proposal for automata description

The VAUCANSON group

October 27, 2005

Abstract

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

1 Introduction

The aim of conceiving a universal automata exchange format is to provide the community with a communication tool that could be used for the connexion of the various softwares that deal with automata and transducers.

The idea of establishing an XML format for automata has been discussed at the CIAA conferences for several years. At CIAA'04, a special session was organized on the subject and two proposals were presented, one by our group (see [3]). We come again at CIAA'05 with a new proposal, which is an evolution of the former but has undergone profound modifications based on our experience in using this format as an input-output format for the VAUCANSON platform.

The most important difference with our previous proposal is the change from a DTD (Document Type Definition) to an XSD Schema for the description of the format. We explain later the reason for this change.

2 Overview

The description of automata is structured in two parts. The `<type>` tag provides automaton type definition, like Boolean automaton, or weighted ones with the ability to specify weight type, alphabet specification, etc. The `<content>` tag provides the definition of the automaton "structure".

The visual representation of automata involves a very large amount of informations. The `<geometry>` data corresponds to the embedding of the automaton in a plane (with informations such as state coordinates or edge type for a transition). The `<drawing>` data contains the definition of attributes that characterize the actual drawing of the graph (such as label position or state color for instance).

3 Description of the format

3.0.1 A first example

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

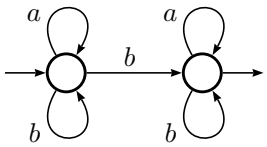

Figure 1: The automaton B_1

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

Figure 2: The XML description of the automaton B_1

3.1 The content tag

The `<content>` tag aims to describe the structure of the automaton. It has two children, mandatory and supposed to appear in a specific order. These two tags allow definitions of states and transitions.

The first tag is `<states>`, representing start 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 data to the state.

The second tag is `<transitions>`, representing start declaration of the set of transitions. Let us note that the initial and final *transitions* are represented as children of `<transitions>`. It is mandatory for a `<transition>` to have

two attributes: `src` and `dst`, representing source and destination 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.

Let us note that 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. Let us note also that when omiting the `label` attribute the XSD grammar propose the identity of the monoid (*i.e.* the empty word) as the default value.

3.2 The type tag

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 `<type>` tag is omitted, the default automaton type is Boolean automaton, on the standard alphabet (all letters of the alphabet, capitalized or not, and digits).

3.2.1 Weighted automata

To describe a weighted automaton, the `<type>` tag provides a set of customizable tags to specify the type of multiplicities. The example of Figure 3 shows how to turn the automaton B_1 into a weighted automaton with weight in $\mathbb{Z}$ – so it counts the number of b in a word.

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

Figure 3: The XML description of the $\mathbb{Z}$-automaton B_1

The multiplicity semiring can be described with two attributes. The `set` indicates the set of weights, while the `operations` attributes indicates the corresponding operations. The possible sets are $\mathbb{B}$, $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{N}$ and *ratSeries* (which will be discussed later).

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 an edge. 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 omiting the weight attribute, the XSD grammar propose the identity of the semiring as default value.

3.2.2 Transducers

As already mentioned above, this proposal aims to limit amount of declarations for widely used structures. Description of transducers is now achieved through the <transducer> tag. The example of Figure 4, the right transducer for binary addition, is represented in Figure 5.

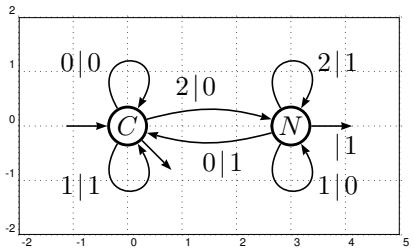

Figure 4: Right transducer for binary addition

```
<transducer>
  <content>
    <states>
       <state name="s0" label="C"/>
       <state name="s1" label="N"/>
    </states>
    <transitions>
       <transition src="s0" dst="s0" in="0" out="0"/>
       <transition src="s0" dst="s0" in="1" out="1"/>
       <transition src="s0" dst="s1" in="2" out="0"/>
       <transition src="s1" dst="s0" in="0" out="1"/>
       <transition src="s1" dst="s1" in="1" out="0"/>
       <transition src="s1" dst="s1" in="2" out="1"/>
       <initial state="s0"/>
       <final state="s0"/>
       <final state="s1" out="1"/>
    </transitions>
  </content>
</transducer>
```

Figure 5: The XML description of the right transducer for binary addition

The <content> tag follows the exact same structure as for automata description. Although a noticeable difference is the extension for transitions 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. They of course can be used

indistinctly in <transition>, <initial> or <final>.

In the example of Figure 5, the <type> tag is omitted. The XSD grammar propose also a default type for transducer: automaton over the free monoid product.

3.3 The geometry tag

The visual representation of automata involves a very large amount of informations. The <geometry> data corresponds to the embedding of the automaton in a plane (with informations such as state coordinates or edge type for a transition). The format provides the possibility to set these properties at any level of the document and to locally override them in a child tag.

The example of Figure 6 sets a global offset for the document, and then places a state in the plane.

```
<transducer>
  <geometry x="-2" y="-2"/>
  <content>
     <states>
        <state name="s0" label="C"><geometry x="0" y="0"/>
        </state>
        <state name="s0" label="N"><geometry x="3" y="0"/>
        </state>
      ...
</transducer>
```

Figure 6: Setting geometry properties

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

If it is a child of <transition>, <initial> or <final>, two attributes can be set. First, the edgeType attribute, that assign the type of the edge (*line*, *arcL*, *arcR*, *curve*). Then, the direction attribute, that can be used to assign the direction angle of a loop, for instance. Note that this attribute is numeric.

3.4 The drawing tag

The <drawing> data contains the definition of attributes that characterize the actual drawing of the graph (such as label position or state color for instance). Most of them are indeed implicit and provided by drawing programs; the format only provides the possibility to make them explicit. As the geometry tag, this tag can be used at any level of the document and be locally overridden in a child tag.

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 edgeStyle usage are shown in Figure 7. These attributes use a string representation to describe their values.

One of the powerful features of XSD Schema descriptions 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" edgeStyle="dashed"/>
  <content>
     <states>
        <state name="s0">
            <drawing stateFillColor="red"/>
        </state>
      ...
</transducer>
```

Figure 7: Setting drawing properties

4 Discussion

4.1 The complexity of the `type` tag

In section 3.2.1, we briefly introduced how the `<type>` tag can be used to specify weight types for a weighted automaton. The aim of the `<type>` tag is to provide a set of tags that allows full description of the automaton type.

4.1.1 Alphabet specification

The user may need to use an alphabet that is not necessarily the standard letter alphabet. For example, a restriction of the alphabet to $\{a, b\}$ is proposed in Figure 8.

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

Figure 8: Setting $\{a, b\}$ alphabet

4.1.2 Default types

The `<type>` tag has two children: the `<monoid>` tag and the `<semiring>` tag. Note that both of these tags are not mandatory, and have different values according to the root tag. Figure 9 shows the equivalent XML code if you omit the

`<type>` tag when declaring an automaton. Similarly, Figure 10 shows the default type for transducers. The `operations` attributes is setted to `"numerical"`, which means that usual laws over $\mathbb{B}$ shall be applied.

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

Figure 9: Default type for an automaton

```
<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"/>
</type>
```

Figure 10: Default type for a transducer

4.1.3 Advanced example

The power of the type tag is enlighted with the example of Figure 11. This example describes the right transducer for binary addition (Figure 4) 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. For the sake of space saving, the content part is omitted, but is *verbatim* the one proposed in Figure 5.

4.2 From DTD to XSD

The most important difference with our previous proposal is the change from a DTD (Document Type Definition) describing the tags for automata representation to an XSD 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 amounts to have default types, in order to omit `<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 `<type>` tag must of course be different. This is not possible with a DTD description. The use of a XSD overcomes this difficulty,

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

Figure 11: Right transducer for binary addition

since it is possible to define different properties for a same element, according to the embracing context. Is is so possible to locally alter the behavior of a tag, and make it context-sensitive. With this feature, default values for the `<type>` tag are achieved, whether it is a child of `<transducer>` or of `<automaton>`.

4.3 Convenient

4.3.1 Sessions

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, as shown in Figure 12.

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

Figure 12: Session of numerous automata

5 Conclusion

For the past year we experimented the proposal made at CIAA'04 in the Vaucanson platform. This new proposal comes as a result of this experiment, with

simplifications where it was 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 of these 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/`