Using Set Operations to Create Powerful New XML Languages

Roger L. Costello
January 5, 2014

This paper shows how to create powerful new XML languages by applying set operations to existing XML languages.

The technique shown in this paper is quite different from approaches employed today, which create new XML languages by extending or restricting an XML Schema. The approach employed today is grammar-based whereas the technique shown in this paper is set-based. Sets are the foundation of mathematics, so by using set operations to create new XML languages we have at our disposal the entire power of mathematics!

Using Set Union to Create New XML Languages

Suppose person #1 creates an XML language consisting of a list of Book elements:

```
<xs:group name="Books">
   <xs:sequence maxOccurs="unbounded">
      <xs:element name="Book">...</xs:element>
   </xs:sequence>
</xs:group>
```

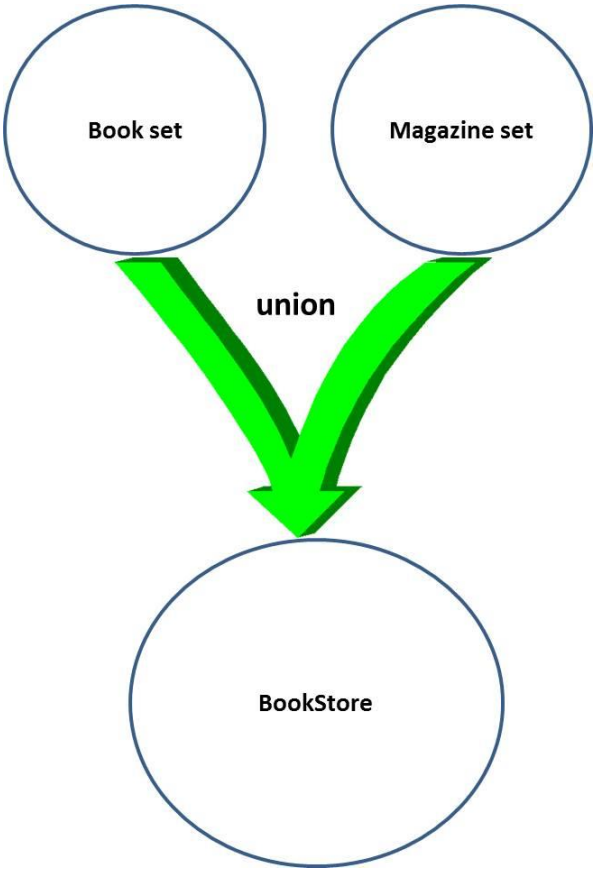
Person #2 creates an XML language consisting of a list of Magazine elements:

```
<xs:group name="Magazines">
   <xs:sequence maxOccurs="unbounded">
      <xs:element name="Magazine">...</xs:element>
   </xs:sequence>
</xs:group>
```

Now, person #3 can reuse them to create a BookStore XML language that consists of either a list of Book elements or a list of Magazine elements:

```
<xs:element name="BookStore">
   <xs:complexType>
      <xs:choice>
         <xs:group ref="Books" />
         <xs:group ref="Magazines" />
      </xs:choice>
   </xs:complexType>
</xs:element>
```

Neat! We have created a new XML language by doing a union of two existing XML languages. Here is a graphic to show what we've accomplished:



Using Set Intersection to Create New XML Languages

The previous section showed how to create new XML languages by applying set union to existing XML languages. In this section we apply set intersection to create new XML languages. I must admit that I was stumped at how to intersect XML languages. Fortunately, Rick Jelliffe provided the answer:

> *The mechanism is simply to validate twice, first*
> *against one schema and second against the other.*

Ah, yes, it seems so obvious now. Thanks Rick.

So if we have XML language 1 and XML language 2, then the intersection of them is obtained by creating an XML instance document and validating it twice, once against XML language 1 and once against XML language 2. The XML instance document is valid only if it conforms to XML language 1 and XML language 2.

Let's take an example.

But before getting into the XML syntax, let's describe the example in a somewhat abstract fashion so we can focus on the concepts. Suppose language 1 consists of strings with *n* **a**'s followed by *n* **b**'s followed by an arbitrary number of **c**'s. The language is a set. We can describe the language (set) using the set-builder notation that we learned in school:

$$\mathbf{L_1 = \{a^n b^n c^m \mid n = 1, 2, \ldots, m = 1, 2, \ldots\}}$$

Language 2 consists of strings with an arbitrary number of **a**'s followed by *n* **b**'s followed by *n* **c**'s. Here's a description of the language using set-builder notation:

$$\mathbf{L_2 = \{a^m b^n c^n \mid n = 1, 2, \ldots, m = 1, 2, \ldots\}}$$

The intersection of these two language may be understood as follows: L_1 requires every **a** have a matching **b**, and L_2 requires every **b** have a matching **c**, so the intersection requires every **a** have a matching **b** and every **b** have a matching **c**; that is, an equal number of **a**'s, **b**'s, and **c**'s:

$$\mathbf{L_1 \cap L_2 = \{a^n b^n c^n \mid n = 1, 2, \ldots\}}$$

This is quite exciting. Here's why: both L_1 and L_2 are context-free languages but the intersection is a context-sensitive language. Recall that context-sensitive languages are more powerful than context-free languages. So by intersection we are elevated to a more powerful language. That is exciting.

Okay, that's the abstract example. Now let's see the real-world example.

Recently I learned that the KML (XML) language has a <Track> element that must consist of *n* <when> elements followed by *n* <gx:Coord> elements, like so:

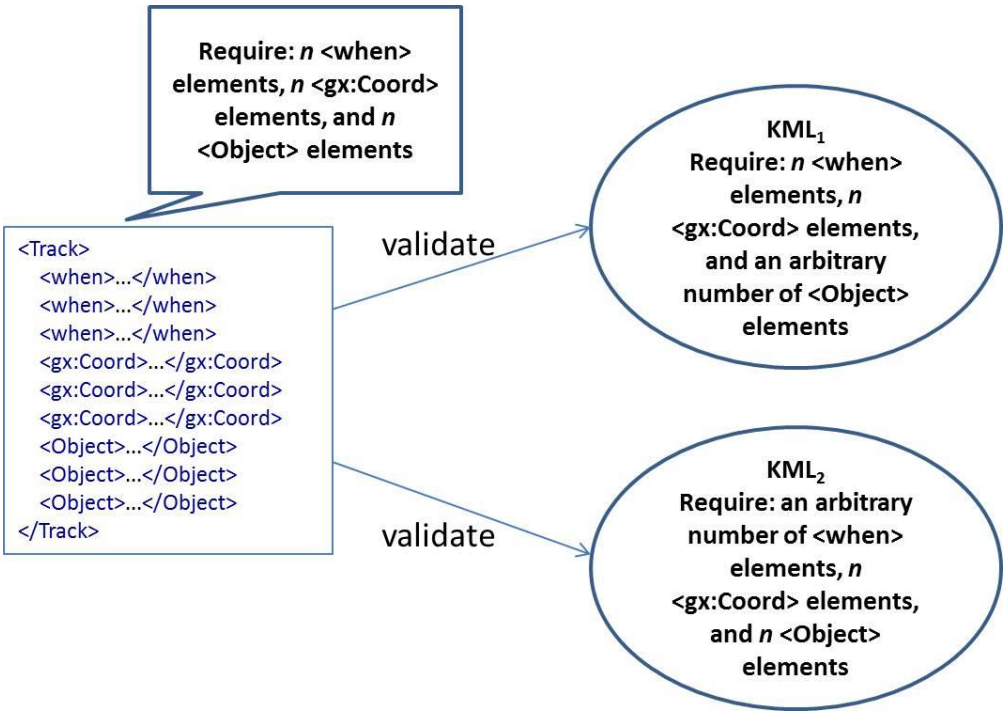
```
<Track>
  <when>...</when>
  <when>...</when>
  <when>...</when>
  <gx:Coord>...</gx:Coord>
  <gx:Coord>...</gx:Coord>
  <gx:Coord>...</gx:Coord>
</Track>
```

In that XML the <Track> element has 3 <when> elements followed by 3 <gx:Coord> elements.

Suppose person #4 creates a dialect of KML like so: he extends the <Track> element with an arbitrary number of <Object> elements. So the <Track> element is required to have *n* <when> elements, *n* <gx:Coord> elements, and an arbitrary number of <Object> elements. I will refer to this dialect of KML as KML_1.

Person #5 also creates a dialect of KML: he also extends <Track> with <Object> elements. However, he requires every <gx:Coord> element have a matching <Object> element, and permits the number of <when> elements to be arbitrary. So the <Track> element consists of an arbitrary number of <when> elements, *n* <gx:Coord> elements, and *n* <Object> elements. I will refer to this dialect of KML as KML_2.

Person #6 wants his KML documents to have <Track> elements that consist of *n* <when> elements, *n* <gx:Coord> elements, and *n* <Object> elements (equal number of all three elements). He can accomplish this by intersecting KML_1 and KML_2. Now hold onto your hat because this is incredible: to intersect KML_1 and KML_2 person #6 has to do nothing! He simply validates his XML instance documents against both KML_1 and KML_2. If an instance document conforms to both KML_1 and KML_2 then the <Track> element will necessarily contain *n* <when> elements, *n* <gx:Coord> elements, and *n* <Object> elements. Here is a graphic to illustrate the validation that is performed to achieve the intersection of KML_1 and KML_2:



If both validations come back positive, then the XML instance document is the intersection of KML_1 and KML_2.

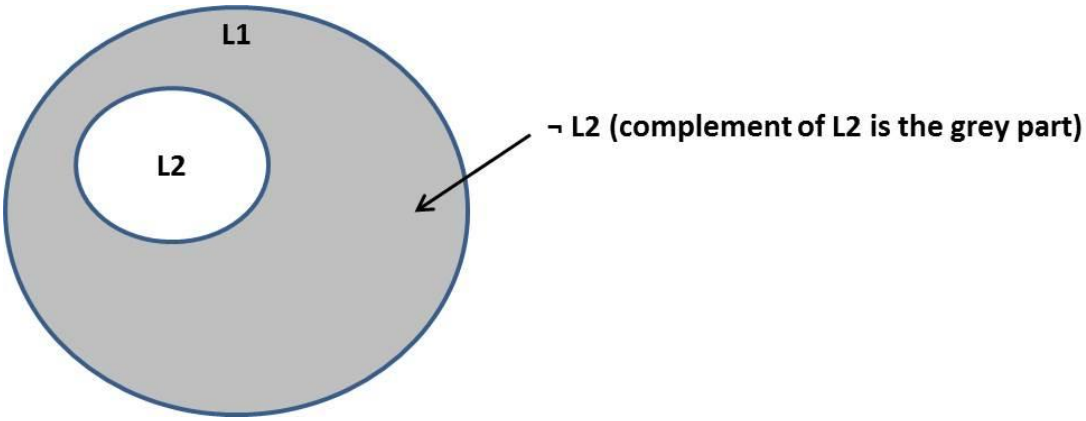
In the previous section we implemented set union by creating a small XML Schema component that contains a choice:

```
<xs:element name="BookStore">
  <xs:complexType>
    <xs:choice>
      <xs:group ref="Books" />
      <xs:group ref="Magazines" />
    </xs:choice>
  </xs:complexType>
</xs:element>
```

However, we now see that was unnecessary. To achieve set union we simply validate an XML instance against both schemas (the Books schema and the Magazines schema): if either one of the validations comes back positive, then the XML instance document is the union of the Books schema and the Magazines schema.

Using Set Complement to Create New XML Languages

Taking the complement of a set means that we want everything that is *not* in the set. Suppose we have two languages, where L2 is a subset of L1. The complement of L2 is the part outside of it:



So, the complement of L2 is the set of XML instance documents that successfully validate against L1 but don't validate against L2. In other words, validate the XML instance documents against both L1 and L2; the XML instance should pass validation against L1 and fail validation against L2.

For example, suppose the L1 is a BookStore that consists of Books and optionally Magazines. L2 is a BookStore that consists only of Magazines. The complement of L2 is a BookStore that consists exclusively of Books.

One might say that L1 (Bookstore with Books and Magazines) is the "universe" and complement is applied to a language that is a subset (BookStore with just Magazines) of the universe. So the complement is everything inside the universe but outside the subset.

Thanks to Dimitre Novatchev for this section of the paper. Very cool stuff Dimitre, thanks!

Using Other Set Operations to Create New XML Languages

We have seen how to create new XML languages using three set operations: union, intersection, and complement. There are other powerful set operations, such as reverse and concatenation. This article describes the other set operations:

http://en.wikipedia.org/wiki/Formal_language#Operations_on_languages

Powerful XML Languages

In a previous section we saw that the intersection of two context-free languages produced a context sensitive languages. That is very cool.

The field of formal languages provides us with these remarkable results:

1. The intersection of two context-free languages always has a context-sensitive grammar.

2. The intersection of three context-free languages is more powerful than the intersection of two of them.

To recap: by treating XML languages as sets we can perform set operations on them and create enormously powerful XML languages.

Strategy for Multiple Validations

We've seen that creating new XML languages via set operations involves multiple validations. So we need a good strategy for performing multiple validations. Once again, Rick Jelliffe provides us with the solution:

> *ISO DSDL was designed to allow you to specify this kind of thing in nice tractable layers.*

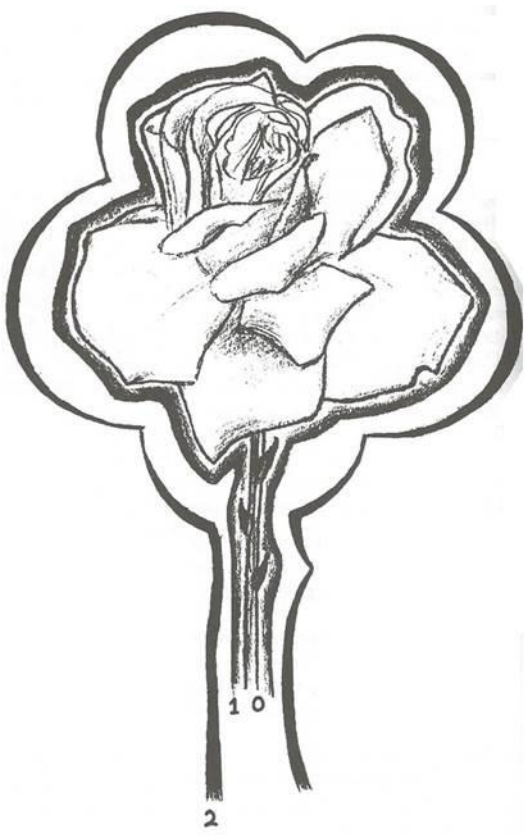
A Beautiful Metaphor for Grammars: Outlining a Rose

Recall that a Type 0 grammar (phrase structure grammar) is more powerful than a Type 1 grammar (context-sensitive grammar) which is more powerful than a Type 2 grammar (context-free grammar) which is more powerful than a Type 3 grammar (regular grammar).

"So what?" you ask. Well, the more powerful the grammar, the better it can define more complicated boundaries between correct and incorrect XML languages. Below is a beautiful metaphor that describes this. It comes from the fabulous book *Parsing Techniques* by Dick Grune et al.

Outlining a Rose

Imagine drawing a rose. It is approximated by increasingly finer outlines. In this metaphor, the rose corresponds to the language (imagine the sentences of the language as molecules in the rose); the grammar serves to delineate its silhouette. A regular grammar only allows us straight horizontal and vertical line segments to describe the flower; ruler and T-square suffice, but the result is a course and mechanical looking picture. A CF grammar would approximate the outline by straight lines at any angle and by circle segments; the drawing could still be made using the classical tools of compass and ruler. The result is stilted but recognizable. A CS grammar would present us with a smooth curve tightly enveloping the flower, but the curve is too smooth: it cannot follow all the sharp turns, and it deviates slightly at complicated points; still, a very realistic picture results. An unrestricted phrase structure grammar can represent the outline perfectly. The rose itself cannot be caught in a finite description; its essence remains forever out of our reach.



Real-World Examples of $a^n b^n$

The classic example of a context-free grammar is $a^n b^n$. That is, *n* occurrences of **a** followed by an equal number of occurrences of **b**. There are many real-world examples of data that has the requirement of *n* occurrences of something followed an equal number of occurrences of something else. See this web page for many examples:

http://cs.stackexchange.com/questions/19485/is-an-bn-an-artificial-grammar-or-does-it-occur-in-the-real-world