Chomsky Hierarchy for XML Developers

Roger L. Costello
December 15, 2013

A few days ago Liam Quin posted this to the xml-dev list

> *I'd hope that most of the people here … are familiar with it [the Chomsky hierarchy].*

It just so happens that recently I have been learning about the Chomsky hierarchy. So I figured that I would have a go at explaining it, orienting the explanation to XML developers.

Context-Free Grammars

One way to create an XML document is by nesting components inside one another. For example, suppose I create a **name** component:

```
<name>
    <given>___</given>
    <surname>___</surname>
</name>
```

I could create an XML document containing just that component. Or, I could nest the **name** component in another component. Here I create an **author** component whose content is the **name** component:

```
<author>
    <name>
        <given>___</given>
        <surname>___</surname>
    </name>
</author>
```

The **name** component is defined independent of the **author** component. It is being used by the **author** component, but it could be used by other components as well. So, the **name** component is defined independent of any context. It is *context-free*.

I could create an XML document containing just that **author** component. Or, I could nest the **author** component in another component. Here I create a **book** component whose content is the **author** component:

```
<book>
    <author>
        <name>
            <given>___</given>
            <surname>___</surname>
        </name>
    </author>
</book>
```

The **author** component is defined independent of the **book** component. It is being used by the **book** component, but it could be used by other components. So, the **author** component is defined independent of any context. It is *context-free*.

I could create an XML document containing just that **book** component. Or, I could nest the **book** component in another component.

Okay, you get the idea. There are two key points: (1) Each component is created independent of context. (2) The components are nested.

So I have created an XML vocabulary that consists of nested, context-free components. I can use any of the schema languages (XSD, RNG, or DTD) to formally define the XML vocabulary.

What I have described above is a *grammar*. [A grammar is a set of elements and their arrangement.] In particular, I have defined a *context-free (CF) grammar.* A CF grammar is also known as a *Type 2 grammar*.

The main properties of CF grammars are: (1) their components are independent (context-free), and (2) the components nest, one within another.

The XML language itself is described using a CF grammar: http://www.w3.org/TR/REC-xml/#sec-notation

Regular Grammars

Suppose I want to create XML documents that contain data about a cellphone. I proceed as follows: I enumerate the service providers, then provide a place to put features data:

```
<Cellphone>
    <Ameritech>___</Ameritech>
    <AirTouch>___</AirTouch>
    <CellularOne>___</CellularOne>
    <BellSouthMobility>___</BellSouthMobility>
    <GTEWireless>___</GTEWireless>
    <SouthwesternBell>___</SouthwesternBell>
    <Features>
        ???
    </Features>
</Cellphone>
```

For <Features> I enumerate the features, then provide a place to put dimensions data:

```
<Cellphone>
    <Ameritech>___</Ameritech>
    <AirTouch>___</AirTouch>
    <CellularOne>___</CellularOne>
    <BellSouthMobility>___</BellSouthMobility>
    <GTEWireless>___</GTEWireless>
    <SouthwesternBell>___</SouthwesternBell>
    <Features>
        <Authentication/>
        <RingerAndEarpieceVolumeControl/>
```

```
            <BatteryStrengthIndicator/>
            <AudibleKeypadControls/>
            <DTMF-KeytoneSignalling/>
            <One-YearPartsAndLaborWarranty/>
            <StoresNamesAndNumbers/>
            <OneTouchDialing/>
            <CallerID/>
            <Dimensions>
                ???
            </Dimensions>
        </Features>
    </Cellphone>
```

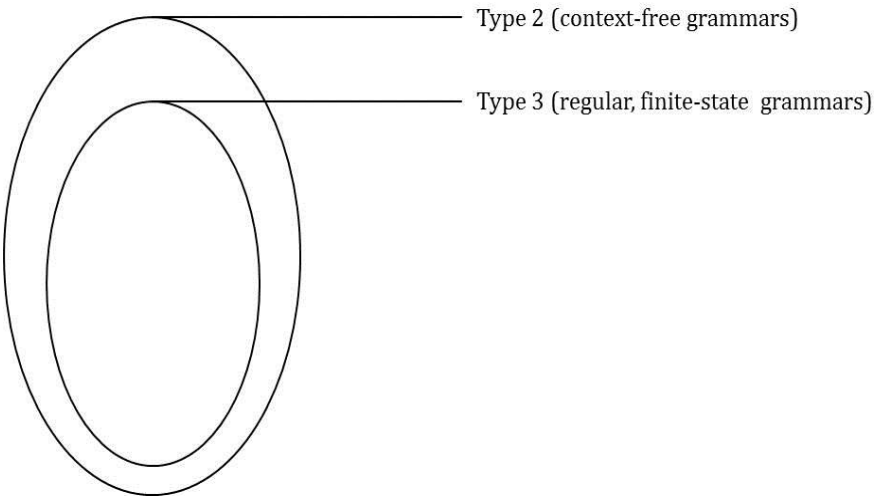
For <Dimensions> I enumerate the dimension data:

```
    <Cellphone>
        <Ameritech>___</Ameritech>
        <AirTouch>___</AirTouch>
        <CellularOne>___</CellularOne>
        <BellSouthMobility>___</BellSouthMobility>
        <GTEWireless>___</GTEWireless>
        <SouthwesternBell>___</SouthwesternBell>
        <Features>
            <Authentication/>
            <RingerAndEarpieceVolumeControl/>
            <BatteryStrengthIndicator/>
            <AudibleKeypadControls/>
            <DTMF-KeytoneSignalling/>
            <One-YearPartsAndLaborWarranty/>
            <StoresNamesAndNumbers/>
            <OneTouchDialing/>
            <CallerID/>
            <Dimensions units="inches">
                <height>6 5/8</height>
                <width>1 7/8</width>
                <thickness>1</thickness>
            </Dimensions>
        </Features>
    </Cellphone>
```

Do you see the pattern being used? There is an enumeration of elements, followed by a container. For the container there is an enumeration of elements, followed by another container. And so on.

Here is the key point: the container always comes last, after the enumeration of elements. This is called a *regular* or *finite state (FS) grammar*. FS grammars are also known as *Type 3 grammars*.

The preceding section talked about context-free (CF) grammars. A CF grammar can be designed to have an enumeration of elements, followed by a container. But, a CF grammar is not restricted to placing the container last. A CF grammar can place the container first, last, or in-between the enumeration elements. So a CF grammar can express any design that a FS grammar can, plus more. CF grammars are a superset of FS grammars. Here is a Venn diagram of the XML languages that these two grammars can produce:



Context-Sensitive Grammars

Whereas the components in CF grammars are independent of context, the components in context-sensitive (CS) grammars may depend on their context. Let's see what this means.

A few weeks ago Rick Jelliffe posted this to the xml-dev list:

> *Information architects reading might be interested that here in Australia we are increasingly having to deal with people's names from our vibrant neighbor Indonesia, where people commonly have a single name only (e.g. Munali). It is not a family name, not a surname, and not a second name.*

In the section on CF grammars I created this **`name`** component:

```
<name>
    <given>___</given>
    <surname>___</surname>
</name>
```

But now, given Rick's post, the content of <name> should actually depend on the context. I extend the <name> element by adding a **`country`** attribute, like so:

```
<name country="Indonesia">
```

or

```
<name country="United States">
```

For the former, the content of the **`name`** component must be just a name (e.g. Munali):

```
<name country="Indonesia">____</name>
```

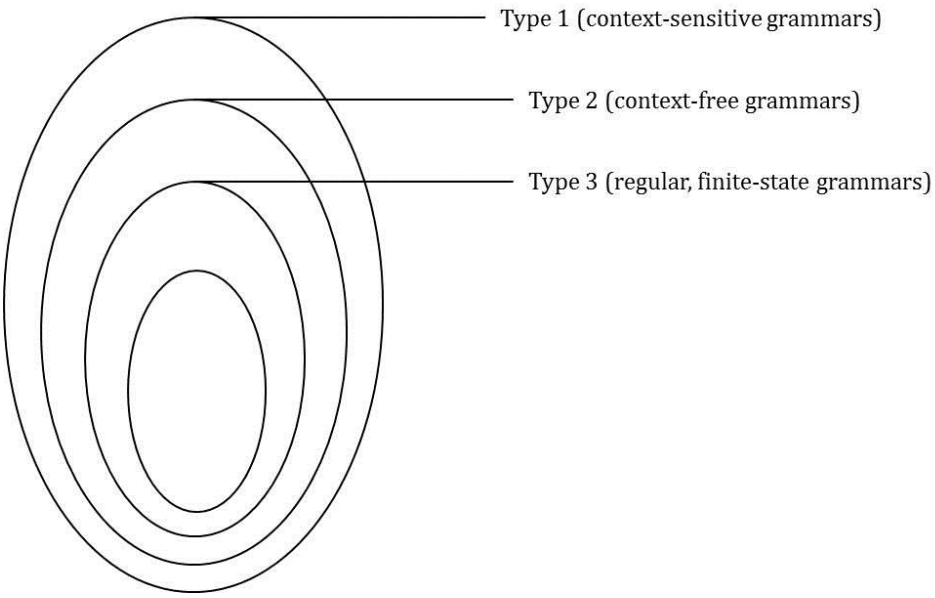
For the later, the **`name`** component has two elements, <given> and <surname>:

```
<name country="United States">
    <given>___</given>
    <surname>___</surname>
</name>
```

This kind of grammar is called a *context-sensitive (CS) grammar*. It is also known as a *Type 1 grammar*.

Interestingly, CS grammars cannot be expressed using XSD 1.0, RNG, or DTD. Those schema languages are not powerful enough. However, XSD 1.1 adds some meta-notation (e.g., the xs:assert element) that enables an additional layer of constraints to be added onto CF grammars. Also, Schematron is a very popular meta-language for expressing an additional layer of context-sensitive constraints.

CS grammars have all the power of CF grammars, plus more: a CF grammar is a CS grammar with zero context restrictions. Here is the updated Venn diagram:



Phrase-Structure Grammars

These are the most powerful grammars. They have no restrictions on the grammar.

In all the other grammars, the content of an element must expand the element (increase the information). For example, the **`author`** component is expanded by the **`name`** component:

```
<author>
    <name>
        <given>___</given>
        <surname>___</surname>
    </name>
</author>
```

With *phrase-structure (PS) grammars*, the information can expand or shrink. Let me give you an example of shrinking: If the BookStore has books and magazines but the sales are low, drop the magazines. That's an example of the BookStore losing/dropping information. Here's a pseudo-notation that expresses this:

`<BookStore> <books> <magazines> <low-sales>` → `<BookStore> <books> <low-sales>`

The part on the left-hand side of the arrow sets the context. Whenever that context occurs, drop the magazines. See how information is shrinking?

So with PS grammars information can expand or shrink. [That is really cool.]

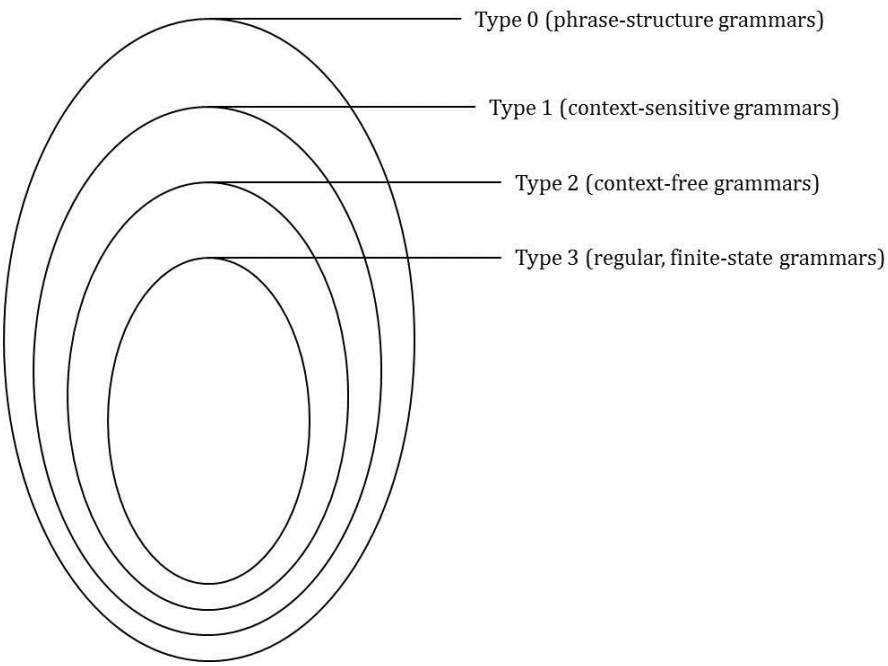
PS grammars are a superset of CS grammars. They are also known as *Type 0 grammars*.

Sorry, I really can't show you an XML document that illustrates PS grammars. In the book that I reference in the following section, the author makes this fascinating statement about PS languages:

> *Strangely enough no simple examples of such languages are known. Although the difference between Type 0 and Type 1 is fundamental and is not just a whim of Mr. Chomsky, grammars for which the difference matters are too complicated to write down; only their existence can be proved.*

Wow! That is an amazing statement. There is much more to be said about PS grammars. For me, this is the most interesting of the grammars. It marks the edge of the universe, with regard to languages that we humans can handle.

The Venn diagram that I have been showing is called the *Chomsky hierarchy*, named after the linguist Noam Chomsky, who first formally characterized these different types of grammars. Here is the full Chomsky hierarchy:



More Information on Grammars

This is a fantastic book: Parsing Techniques: A Practical Guide

In an individual's life one encounters just a few books that really changes one's world. For me, this is one of those books.