

XML and Web Application Programming

Overview

- **Schema languages for XML**
- **XML in programming languages**
- **Web application frameworks**

Part I

- **Schema languages for XML**
 - what is the essence of DTD / XML Schema / RELAX NG ?
 - expressiveness of schema languages
 - formal models, variations of *regular tree grammars*
 - foundation for Part II

Part I

▪ Schema languages for XML

“W3C XML Schema is one spec where your eyes will still twitch and your head still go buzzZZZ even when you read it for the tenth time. Except that you will no longer be surprised by surprises.”

– Michael Kay, editor of W3C's XSLT 2.0

Part II

- **XML in programming languages**
 - how can we integrate XML processing in programming languages?
- XML schemas as types in programming languages
- research projects:
 - XDuce
 - XACT



Part III

- **Web application frameworks**
 - why is Web programming so complicated?
 - Java Servlets and JSP
 - JWG
 - Google Web Toolkit



Part III

11.11 frameworks

» Wow ! Thanks for releasing another Java Web application development framework. The 31,918 frameworks currently available don't offer enough choice.

You know, I was just saying to my friend the other day that we really do not have enough Java web development frameworks. "Why," I said to him, "it's been three days since I read about a new web development framework on TSS ! Is the pace of innovation slowing in the Java world? Are we that far along the road to obsolescence ? Can death be far away ?"

But now I feel better. [...] when I saw the AJAX word I knew that the richness and agileness of the end-user experience would be exceeded only by the usability, scaleability, and, most important of all, testability of applications produced by the [`<useless thing name>`] .

You are some wild and crazy guys, and I, for one, am going to nominate [`<useless thing name>`] for the Web Application Development Framework of the Week. No, I'm serious ! You've earned it ! Stand up and take a bow ! «

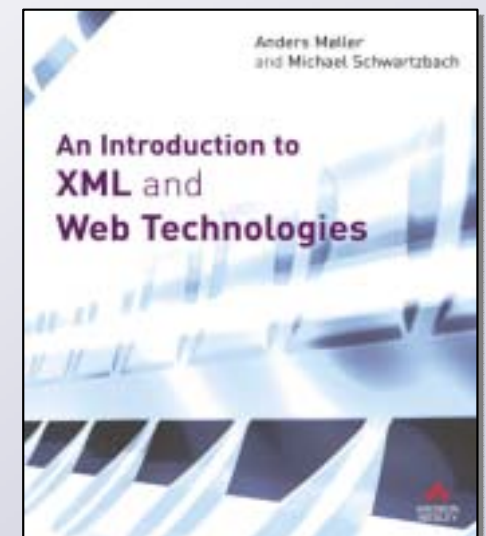
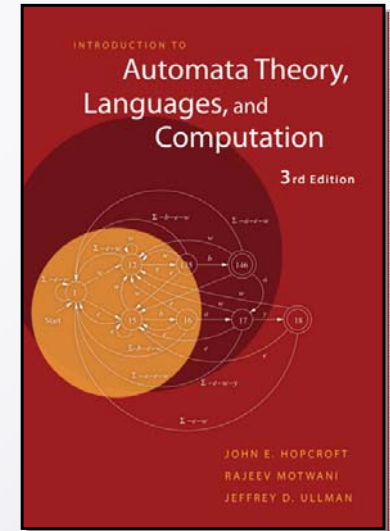
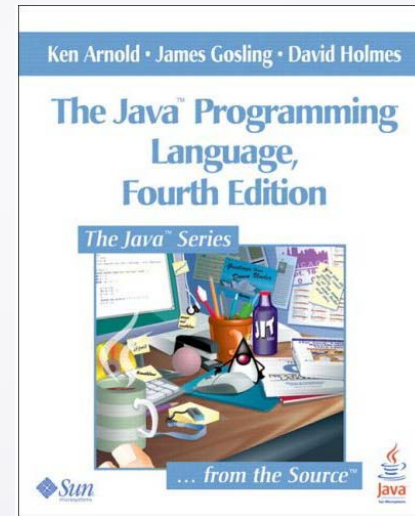
– on theserverside.com as a comment to a tool announcement

Objectives

- These topics illustrate a few areas in $(\text{XML} \cup \text{WWW}) \cap \text{Programming Languages}$
- At different stages:
 - **XML and schemas:** now well-understood, essential to other areas of XML/WWW
 - **XML programming:** many proposals, (still) no “silver bullet”
 - **Web programming:** new frameworks appear frequently, needs consolidation
- Lots of remaining research opportunities!

Prerequisites

- I assume that you have a basic knowledge of
 - Regular languages
 - Java (and ML)
 - WWW and XML/HTML



A tour of schema languages for XML: DTD, XML Schema, and RELAX NG



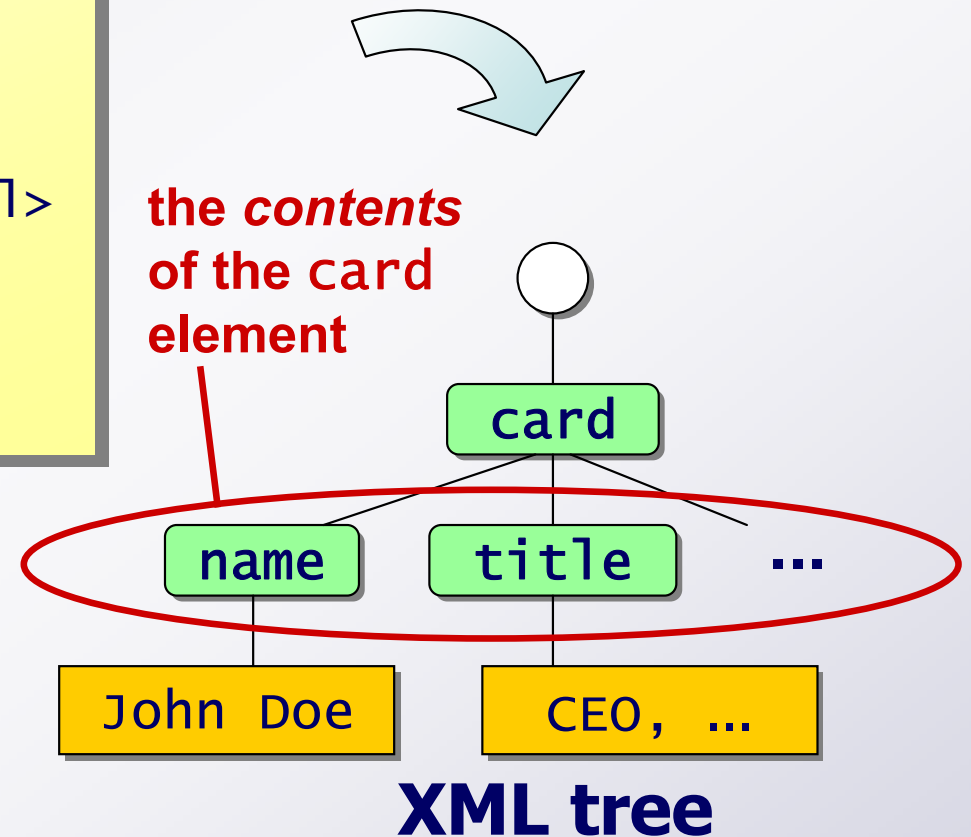
Overview

- Motivation
 - DTD
 - XML Schema
 - RELAX NG
- focus on language constructs that are essential to **expressiveness**

Example: XML for *business cards*

```
<card>
  <name>John Doe</name>
  <title>CEO, widget Inc.</title>
  <email>john.doe@widget.inc</email>
  <phone>(202) 555-1414</phone>
  <logo uri="widget.gif"/>
</card>
```

XML document



- An **XML-based language** is *a **set** of XML documents* (with some semantics)

Schemas for XML

Validation of XML documents:

- A schema describes the **syntax of an XML-based language**
- A schema language (DTD / XML Schema / RELAX NG / ...) is a formal notation for specifying schemas
 - like BNF for programming language syntax
- A schema processor **checks validity** of a document, relative to a schema

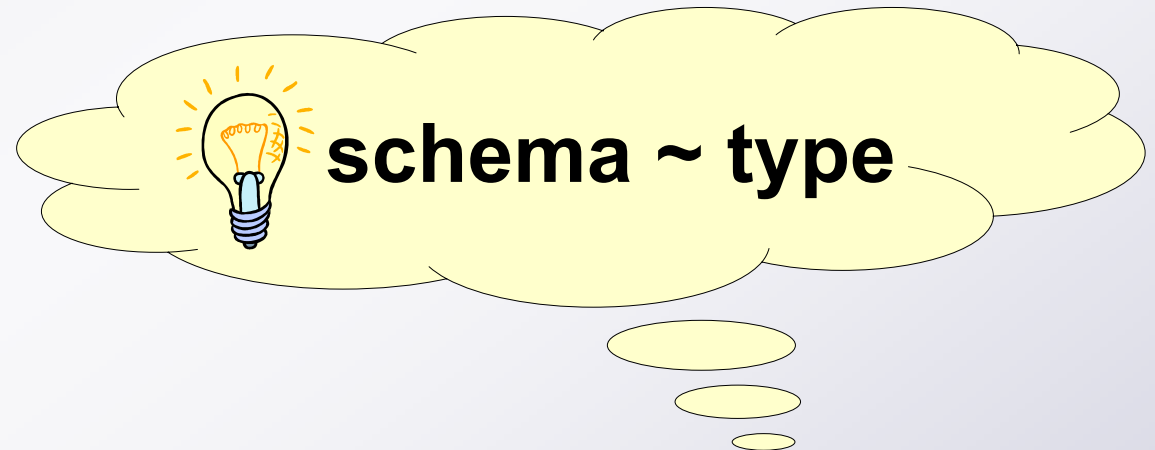
This presentation is based on...

- Anders Møller and Michael Schwartzbach, **An Introduction to XML and Web Technologies**, Addison-Wesley, 2006
- Makoto Murata, Dongwon Lee, Murali Mani, and Kohsuke Kawaguchi, **Taxonomy of XML schema languages using formal language theory**, *ACM Trans. Internet Techn.* 5(4): 660-704, 2005

XML in programming languages

- Many recent programming languages and APIs are designed for processing XML data

- Type checking?
Type inference??



- We need **formal models** of schemas and schema languages!



Overview

- Motivation
- **DTD**
- XML Schema
- RELAX NG

DTD – Document Type Definition

- Introduced with XML 1.0
- Focuses on elements and attributes
- Little control of attribute values and chardata
- No support for namespaces
- Widely used (still!)



Element declarations

```
<!ELEMENT element-name content-model >
```

where *content-model* is generally a restricted **regular expression** over element names and #PCDATA

Observations:

- the content model of a particular element only depends on its name!
- content models must be *deterministic*
- limited control over chardata

Attribute declarations

```
<!ATTLIST element-name attribute-definitions >
```

Each attribute definition consists of

- an attribute name
- an attribute *type* (CDATA, enum, ID, IDREF, ...)
- a *default declaration* (#REQUIRED, #IMPLIED, ...)

Observations:

- the requirements for a particular attribute only depends on its name and the name of the element!
- limited control over attribute values

Example

```
<!ELEMENT card (name,title,email,phone?,logo?)>
```

```
<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT title (#PCDATA)>
```

```
<!ELEMENT email (#PCDATA)>
```

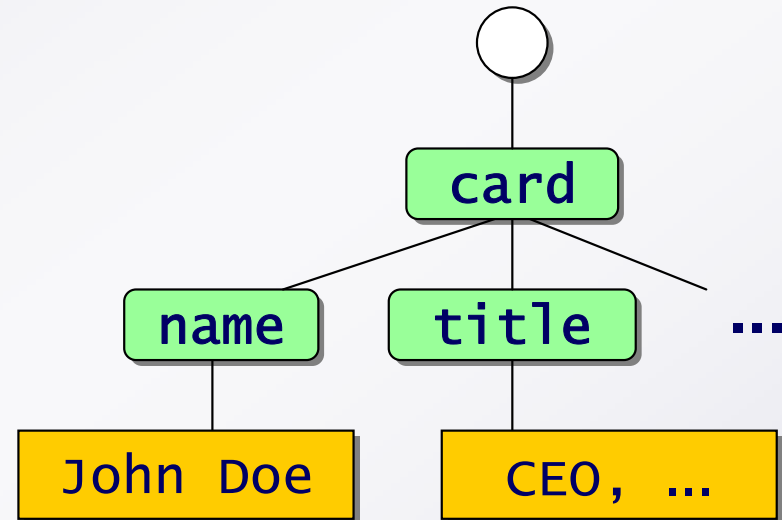
```
<!ELEMENT phone (#PCDATA)>
```

```
<!ELEMENT logo EMPTY>
```

```
<!ATTLIST logo uri CDATA #REQUIRED>
```

General observations

- DTD can only express “**local**” properties on XML trees!
- An XML document can be validated by checking the element nodes in any order



Overview

- Motivation
- DTD
- **XML Schema**
- RELAX NG

XML Schema

- W3C's design goal: make "a better DTD"...
- Successes:
 - Namespace support
 - Modularization
 - Data types
 - Type derivation mechanism
- Critique:
 - lots... it's an easy prey ☺



Types and declarations

- **Simple type definition:**
defines a family of Unicode text strings (i.e. no markup)
- **Complex type definition:**
defines a content and attribute model
- **Element declaration:**
associates an element name with a simple or complex type
- **Attribute declaration:**
associates an attribute name with a simple type

Example (1/2)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://businesscard.org"
        targetNamespace="http://businesscard.org">
```

```
  <element name="card" type="b:card_type"/>
```

```
  <element name="name" type="string"/>
```

```
  <element name="title" type="string"/>
```

```
  <element name="email" type="string"/>
```

```
  <element name="phone" type="string"/>
```

```
  <element name="logo" type="b:logo_type"/>
```

```
  <attribute name="uri" type="anyURI"/>
```

Example (2/2)

```
<complexType name="card_type">  
  <sequence>  
    <element ref="b:name"/>  
    <element ref="b:title"/>  
    <element ref="b:email"/>  
    <element ref="b:phone" minOccurs="0"/>  
    <element ref="b:logo" minOccurs="0"/>  
  </sequence>  
</complexType>
```

```
<complexType name="logo_type">  
  <attribute ref="b:uri" use="required"/>  
</complexType>
```

```
</schema>
```

Complicated structural features

- *Type derivation by restriction or extension*
 - *Substitution groups*
 - *Overloading with local declarations*
 - *...*
- many of these features are not essential to the formal expressiveness

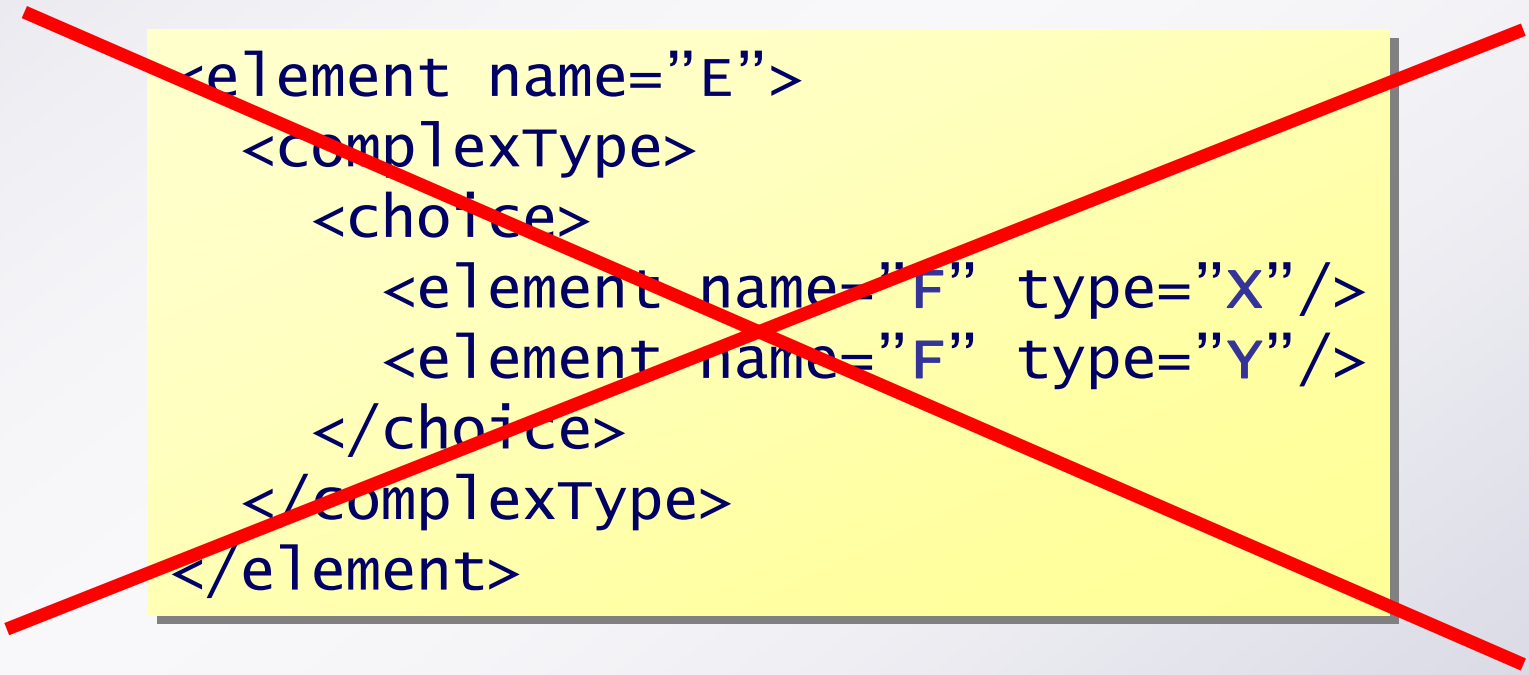
Overloading with local declarations

```
<element name="cardlist">
  <complexType><sequence>
    <element name="title" type="string"/>
    <element ref="x:card" .../>
  </sequence></complexType>
</element>

<element name="card">
  <complexType><sequence>
    <element name="title" type="x:title"/>
    ...
  </sequence></complexType>
</element>
```

The *Element Declarations Consistent* rule

*If a content model contains two or more element declarations with the **same element name** then they must have the **same type definition***



```
<element name="E">  
  <complexType>  
    <choice>  
      <element name="F" type="X"/>  
      <element name="F" type="Y"/>  
    </choice>  
  </complexType>  
</element>
```

Simplifies implementation,
central for formal modeling of schemas...

Unique interpretations

- For *some* applications, an important side-effect of validation is that *each **tree node** is assigned a **schema type*** (i.e. an interpretation)
- Aka. the “post-schema-validation info set”
- The EDC rule in XML Schema ensures ***unique*** interpretations!

What's being used in practice?

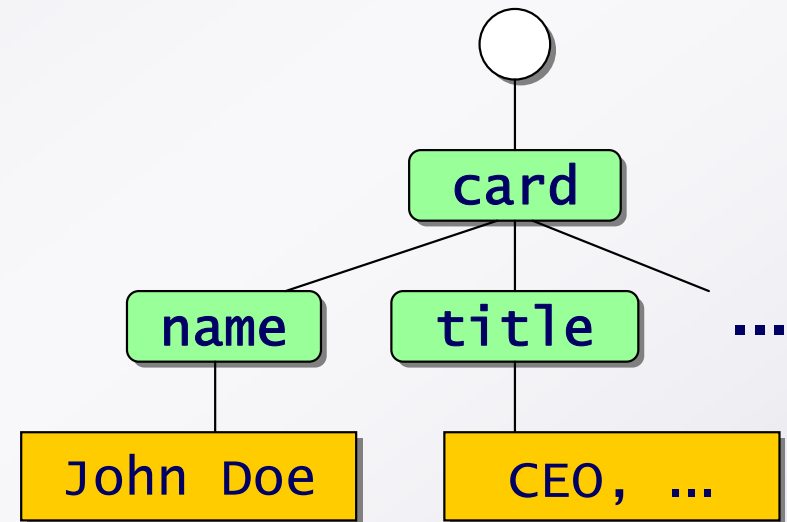
A study of 819 schemas shows:

- simple types and restrictions are heavily used!
- complex type extensions used in only 20%
- substitution groups used in only 6%
- only 27% pass the Schema Quality Check
 - of those, the element structure could in 85% be expressed using DTD! (i.e. no overloading)
 - in most of the remaining 15%, types only depend on the parent context!

– *Any good explanations?*

General observations

- The type associated with an element may (only) depend on the node itself and its ***ancestors***



- The crucial mechanisms are
 - overloading with local declarations
 - the EDC rule

Overview

- Motivation
- DTD
- XML Schema
- **RELAX NG**

RELAX NG

- OASIS + ISO competitor to XML Schema
- Designed for simplicity and expressiveness, solid mathematical foundation



Processing model

- For a valid instance document, the **root** element must match a designated **pattern**
- A ***pattern*** may match **elements**, **attributes**, or **character data**
- Element patterns can contain **sub-patterns** that describe contents and attributes

Patterns

- `<element name="..."> ... </element>`
- `<attribute name="..."> ... </attribute>`
- `<text/>`
- `<group> ... </group>` (concatenation)
- `<choice> ... </choice>` (union)
- `<zeroOrMore> ... </zeroOrMore>` (Kleene star)
- `<oneOrMore> ... </oneOrMore>`
- `<optional> ... </optional>`
- `<empty/>`
- ...

Example

```
<element name="card">
  <element name="name"><text/></element>
  <element name="title"><text/></element>
  <element name="email"><text/></element>
  <optional>
    <element name="phone"><text/></element>
  </optional>
  <optional>
    <element name="logo">
      <attribute name="uri"><text/></attribute>
    </element>
  </optional>
</element>
```

Grammars

- Pattern **definitions** and **references** allow description of *recursive* structures

```
<grammar ...>
  <start>
    ...
  </start>
  <define name="...">
    ...
  </define>
  ...
</grammar>
```

Grammar simplification

- The spec explains a simplification process
- §4.19:
 - every **e**lement is the child of a **define**
 - the child of every **define** is an **e**lement
(i.e. we may assume a 1-1 correspondence between named pattern definitions and element patterns)
- ...

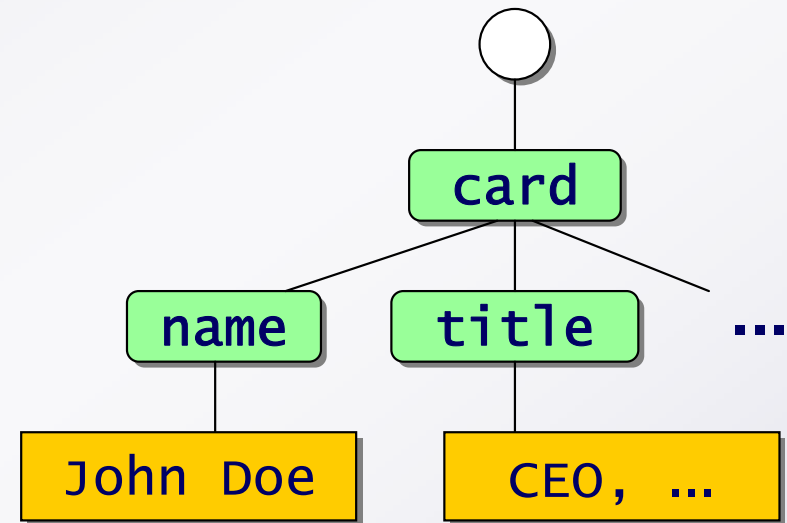
(we need this later...)

No surprising restrictions!

```
<element name="X">  
  <choice>  
    <group>  
      <element name="X">  
        <optional><ref name="P"/></optional>  
      </element>  
      <optional>  
        <attribute name="A"><text/></attribute>  
        <attribute name="B"><text/></attribute>  
      </optional>  
    </group>  
    <element name="X"><empty/></element>  
  </choice>  
</element>
```

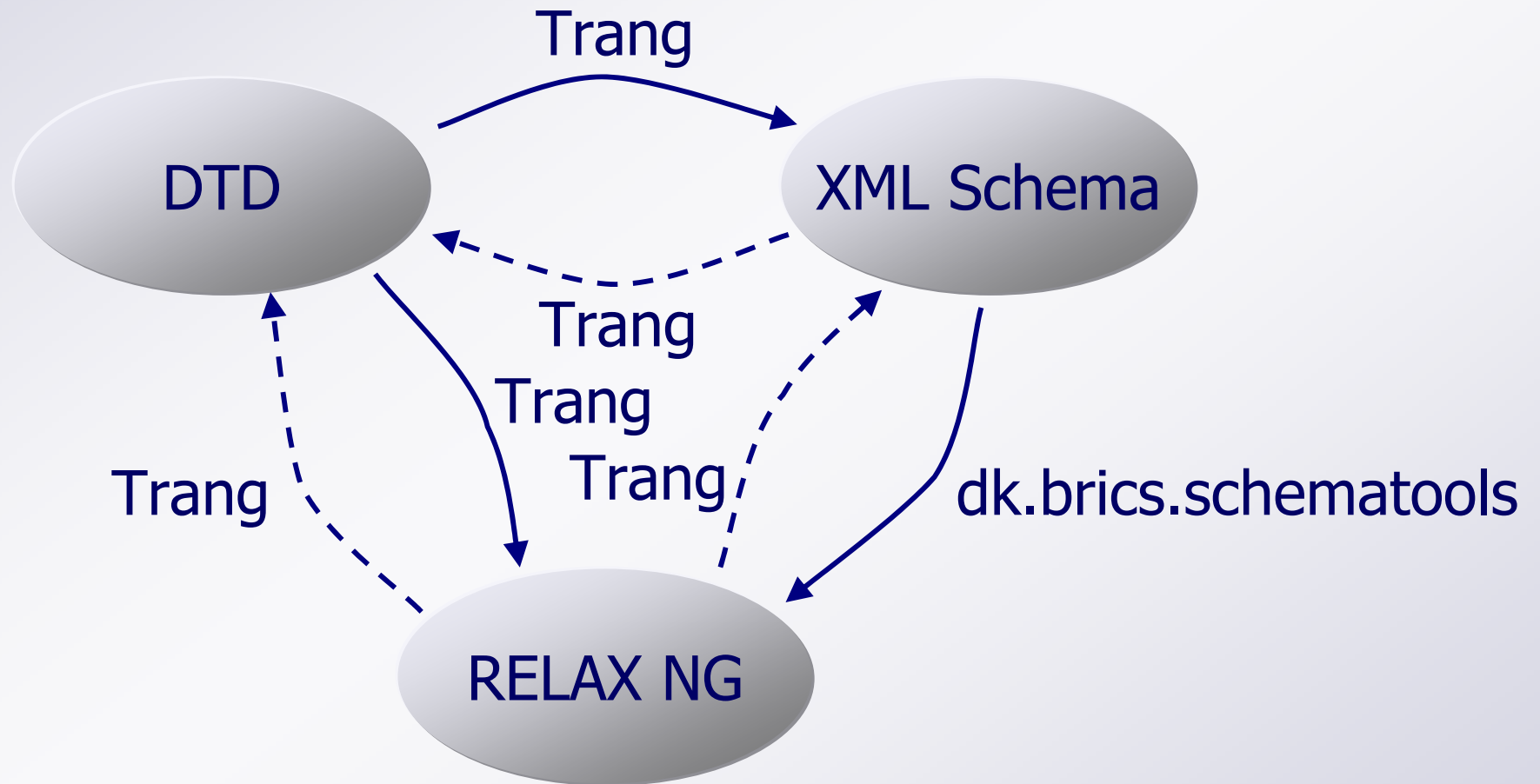
General observations

- RELAX NG schemas can express many **"non-local"** properties on XML trees!



- Validation is more difficult but can be performed in **linear time** (in the size of the XML tree)!
- Unique-interpretations** property fails!
- We shall later examine the **formal model** underlying the design of RELAX NG...

Translating schemas



Trang: <http://www.thaiopensource.com/relaxng/trang.html>
dk.brics.schematools: <http://www.brics.dk/schematools/>

Formal models of XML schemas: regular tree grammars and subclasses



Schemas as formal languages

- Let S be a schema (written in DTD, XML Schema, or RELAX NG)
- Define $\mathcal{L}(S)$ as the set of XML documents that are valid relative to S :

$$\mathcal{L}(S) = \{ X \mid X \text{ is valid relative to } S \}$$

Decision problems

[membership] The problem “given a schema S and an XML document X , is $X \in \mathcal{L}(S)$?” is (fortunately) decidable

How about the following?

[equality] “given two schemas S_1 and S_2 , is $\mathcal{L}(S_1) = \mathcal{L}(S_2)$?”

[subset] “given two schemas S_1 and S_2 , is $\mathcal{L}(S_1) \subseteq \mathcal{L}(S_2)$?”

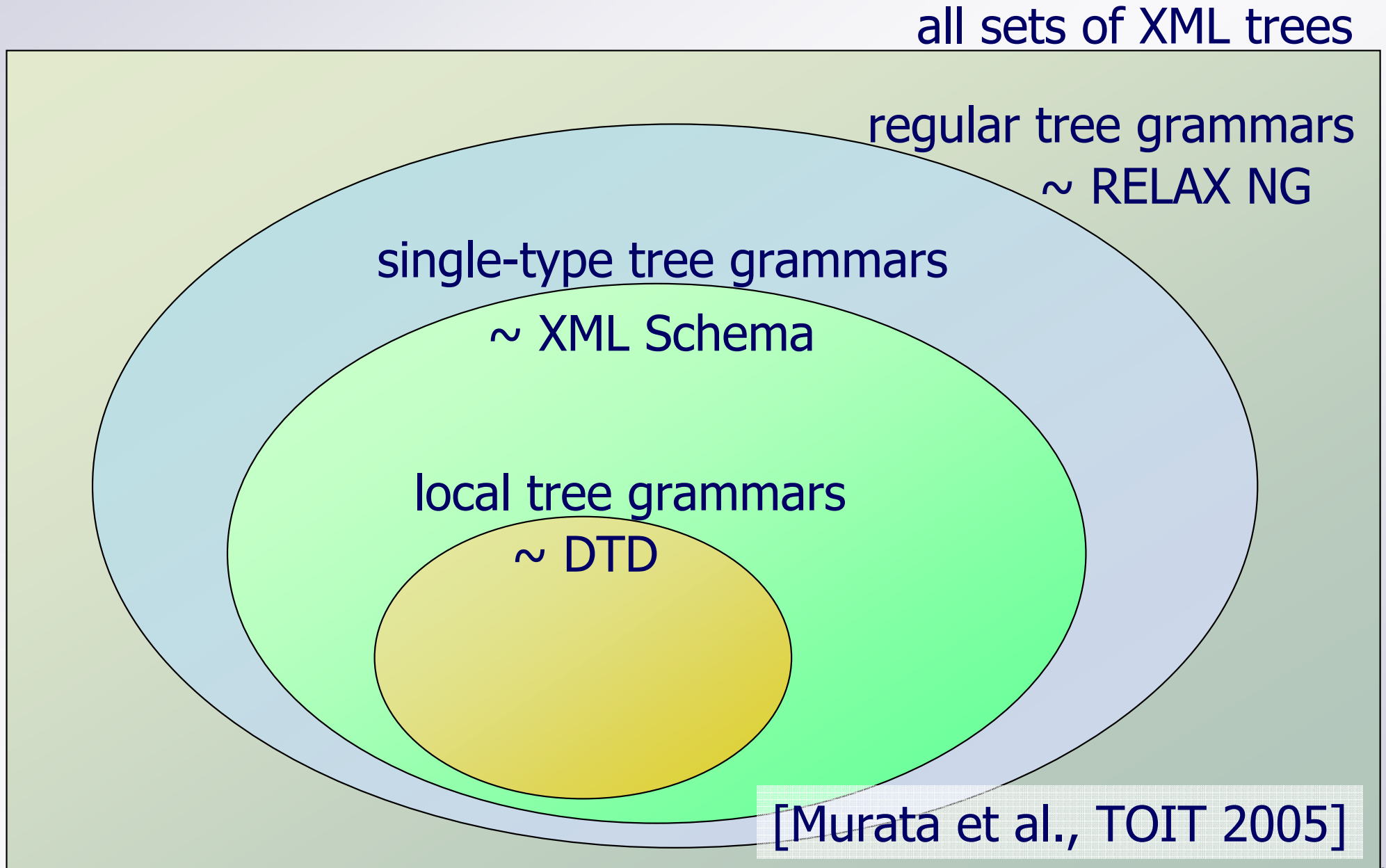
[emptiness] “given a schema S , is $\mathcal{L}(S) = \emptyset$?”

Closure properties

- Is DTD (or XML Schema or RELAX NG) closed under union?
... intersection?
... complement?

*All these decision problems and closure properties are relevant when considering **schemas as types** in programming languages!*

Overview

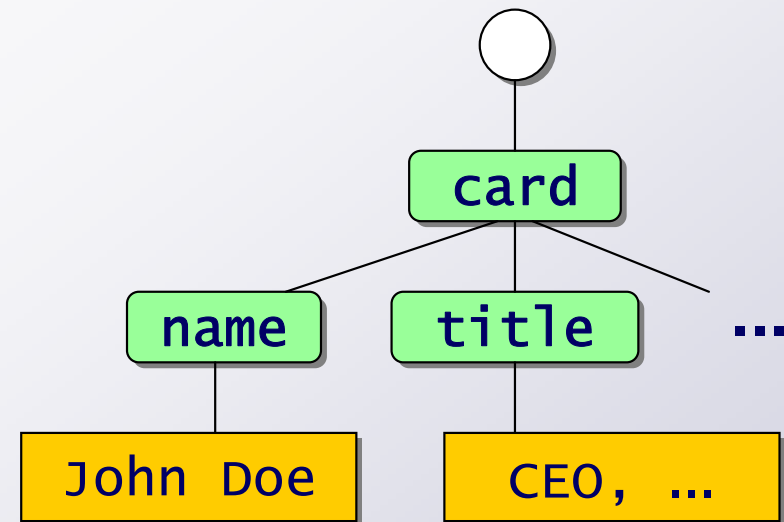


Regular languages

- You know about regular languages on strings... (finite automata, regular grammars, etc.)
- We shall now work with ***regular languages on trees***

Which kinds of trees?

- ***Labeled, ordered, unranked*** trees
 - each node has a *label* (i.e. the element name; we ignore attributes and character data...)
 - each node has an *ordered* sequence of child nodes
 - the child sequences may have *different lengths*



Context-free grammars (over strings)

- A **context-free grammar** is a 4-tuple $G = (N, T, S, P)$ where
 - N is a finite set of nonterminals
 - T is a finite set of terminals
 - S is a set of start symbols, where $S \subseteq N$
 - P is a finite set of production rules of the form $X \rightarrow \alpha$ where $X \in N$ and $\alpha \in (T \cup N)^*$
- The **language** of G , denoted $\mathcal{L}(G)$, is the set of strings over T that can be derived from a start symbol (with the usual definition of “derived”...)

Regular tree grammars

- A **regular tree grammar** is a 4-tuple $G = (N, T, S, P)$ where
 - N is a finite set of nonterminals
 - T is a finite set of terminals
 - S is a set of start symbols, where $S \subseteq N$
 - P is a finite set of production rules of the form $X \rightarrow a[r]$ where $X \in N$, $a \in T$, and r is a regular expression over N
- The **language** of G , denoted $\mathcal{L}(G)$, is the set of trees over T that can be derived from a start symbol (with the obvious definition of “derived”...)

N : types

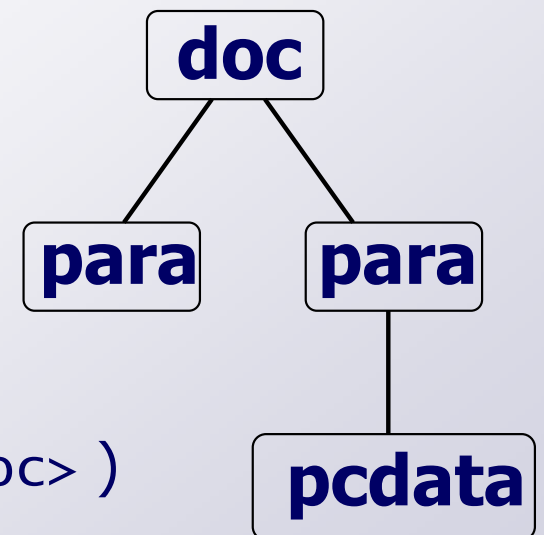
T : element names

Example

Let $G = (N, T, S, P)$ be defined by

- $N = \{Doc, Para1, Para2, PCDATA\}$
- $T = \{\mathbf{doc}, \mathbf{para}, \mathbf{pcdata}\}$
- $S = \{Doc\}$
- $P = \{ Doc \rightarrow \mathbf{doc}[Para1, Para2^*],$
 $Para1 \rightarrow \mathbf{para}[\epsilon],$
 $Para2 \rightarrow \mathbf{para}[PCDATA],$
 $PCDATA \rightarrow \mathbf{pcdata}[\epsilon] \}$

N : types
 T : element names



- A tree that can be derived from G :
(The corresponding XML document:
`<doc><para/><para>text chunk</para></doc>`)
- What is $\mathcal{L}(G)$?

Regular tree grammars and finite-state automata

- Everything you know about regular languages and finite automata on strings **generalizes elegantly to trees!**
 - grammars vs. automata
 - automata operations
 - closure properties
 - decision procedures
 - ...
- and (hopefully) also your intuition about expressibility and limitations!

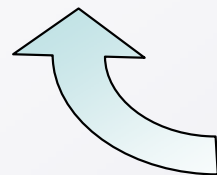
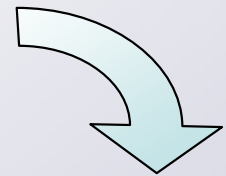
Regular tree grammars vs. RELAX NG

- From (simplified) RELAX NG schema to regular tree grammar*:
 - named pattern \Rightarrow nonterminal
 - element name \Rightarrow terminal

* Ignoring name classes, interleave, attributes, and chardata...

Example

```
<grammar ...>
  <start><ref name="Start"/></start>
  <define name="Start">
    <element name="addressBook">
      <zeroOrMore>
        <ref name="Card"/>
      </zeroOrMore>
    </element>
  </define>
  <define name="Card">...</define>
  ...
</grammar>
```



Start → addressBook[Card*]
Card → ...
...

Local tree grammars

- A ***local tree grammar*** is a regular tree grammar $G = (N, T, S, P)$ where P contains **no** two productions on the form

$$X \rightarrow \mathbf{a}[r]$$

$$Y \rightarrow \mathbf{a}[r']$$

where $X \neq Y$, $\mathbf{a} \in T$, and r and r' are regular expressions over N

- in other words, the **terminal (=element name)** *uniquely determines* the **regular expression (=content model)**

Example

Let $G = (N, T, S, P)$ be defined by

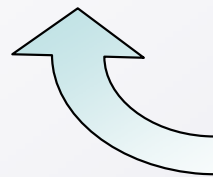
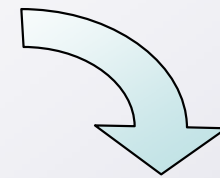
- $N = \{Doc, Para1, Para2, PCDATA\}$
- $T = \{\mathbf{doc}, \mathbf{para}, \mathbf{pcdata}\}$
- $S = \{Doc\}$
- $P = \{ Doc \rightarrow \mathbf{doc}[Para1, Para2^*],$
 $Para1 \rightarrow \mathbf{para}[\epsilon],$
 $Para2 \rightarrow \mathbf{para}[PCDATA],$
 $PCDATA \rightarrow \mathbf{pcdata}[\epsilon] \}$

– is G a *local* tree grammar?

Local tree grammars vs. DTD

- In DTD, the element name uniquely determines the content model !
- From DTD schema to local tree grammar^{*}:

```
<!ELEMENT foo (bar|baz+)>  
<!ELEMENT bar ...>  
<!ELEMENT baz ...>
```



```
Foo → foo[Bar|Baz+]  
Bar → bar[...]  
Baz → baz[...]
```

^{*} Ignoring attributes and chardata...

Regular tree grammars vs. XML Schema

- [Murata et al., TOIT 2005] explains how **XML Schema** can be modeled with **regular tree grammars**
 - complex type \Rightarrow nonterminal
 - element declaration \Rightarrow terminal
- We omit the (many) details.....

Single-type tree grammars

- Let $G = (N, T, S, P)$ be a regular tree grammar
- Two different nonterminals $X, Y \in N$ **compete** if P contains two productions on the form
$$\begin{array}{l} X \rightarrow \mathbf{a}[\dots] \\ Y \rightarrow \mathbf{a}[\dots] \end{array} \quad \text{for some } \mathbf{a} \in T$$
- G is a **single-type tree grammar** if
 - for each production rule in P , no two nonterminals on the right-hand-side compete, and
 - start symbols in S do not compete
- G local $\Rightarrow G$ single-type

Example

Let $G = (N, T, S, P)$ be defined by

- $N = \{Doc, Para1, Para2, PCDATA\}$
- $T = \{\mathbf{doc}, \mathbf{para}, \mathbf{pcdata}\}$
- $S = \{Doc\}$
- $P = \{ Doc \rightarrow \mathbf{doc}[Para1, Para2^*],$
 $Para1 \rightarrow \mathbf{para}[\epsilon],$
 $Para2 \rightarrow \mathbf{para}[PCDATA],$
 $PCDATA \rightarrow \mathbf{pcdata}[\epsilon] \}$

– is G a *single-type* tree grammar?

Single-type tree grammars vs. XML Schema

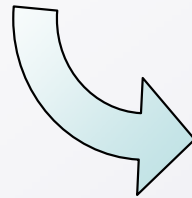
Why the restriction of
nonterminal **competition**?

- it implies that types can be uniquely determined from element names when processing a child sequence!
- it exactly corresponds to the infamous “**Element Declarations Consistent**” restriction in XML Schema!

Example of violation

```
<element name="E">  
  <complexType>  
    <choice>  
      <element name="F" type="X"/>  
      <element name="F" type="Y"/>  
    </choice>  
  </complexType>  
</element>
```

← *violates
EDC!*



```
... → E[F1|F2]  
F1 → F[X]  
F2 → F[Y]
```

← *F₁ and F₂
compete!*

A design bug... ☹

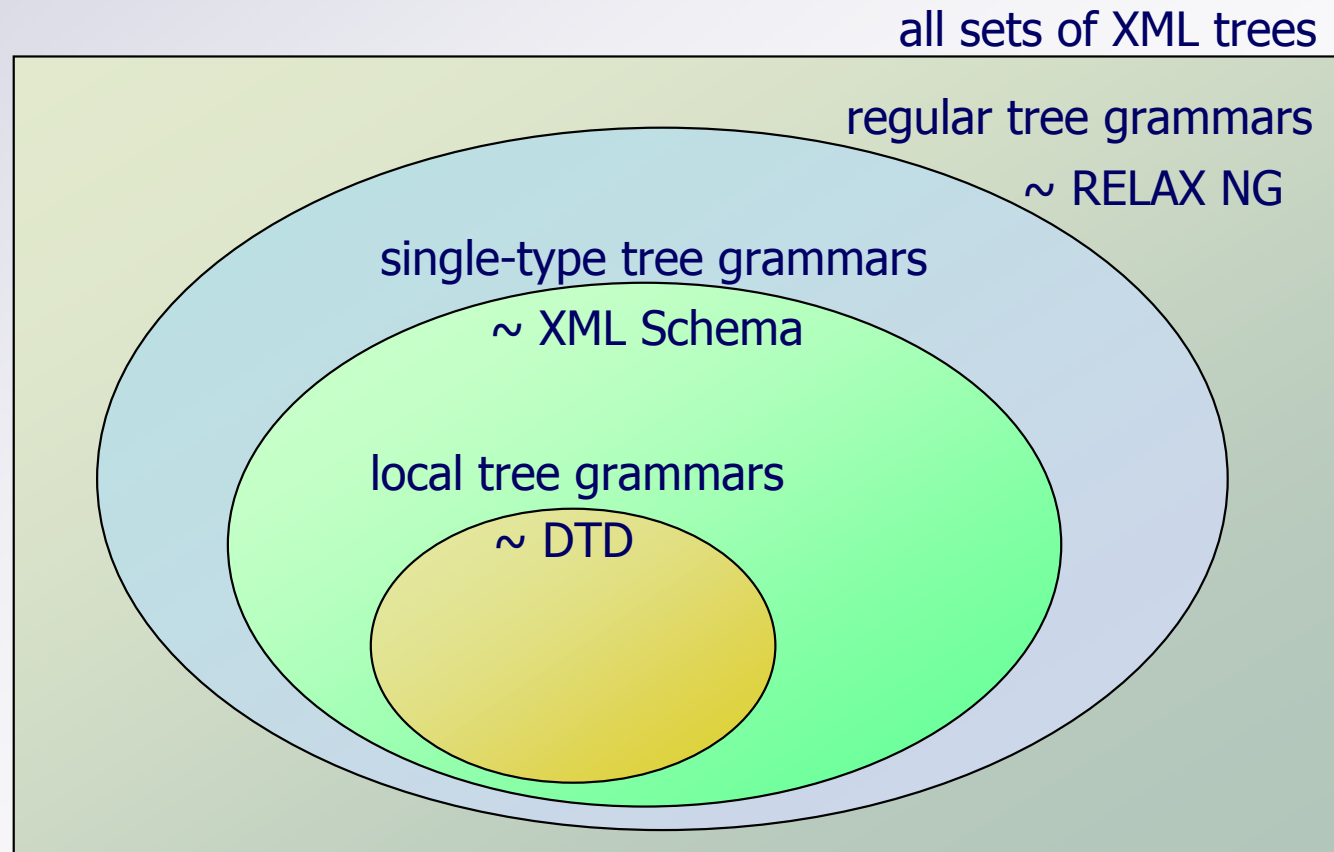
- XML Schema can essentially be modeled as single-type tree grammars, but...
- the any constructs in XML Schema makes it possible to express certain ***non***-single-type tree grammars!
- See example in [Murata et al., TOIT 2005]

Closure properties

	\cap	\cup	\setminus
regular (RELAX NG)	✓	✓	✓
single-type (XML Schema)	✓	÷	÷
local (DTD)	✓	÷	÷

[Murata et al., TOIT 2005]

Summary



- foundation for treating **XML schemas as types** in XML programming languages