INFO-H-509 XML Technologies
XML Schema Languages Part I

Stijn Vansummeren

March 6, 2013

Objectives

1. The purpose of using schemas
2. **Regular Expressions** — a commonly used formalism in schema languages
3. The schema language **DTD** and its expressiveness

Our story so far ...

XML

- Is a standard, flexible notation for text with markup
- Does **not** constrain the form of XML documents

However:

- Applications typically expect XML documents of a **specific form**: XHTML, XML Recipes, ...
- How can this form be described?
- How can applications check that an input document has the requested form?

Our story so far ...

XML

- Is a standard, flexible notation for text with markup
- Does **not** constrain the form of XML documents

However:

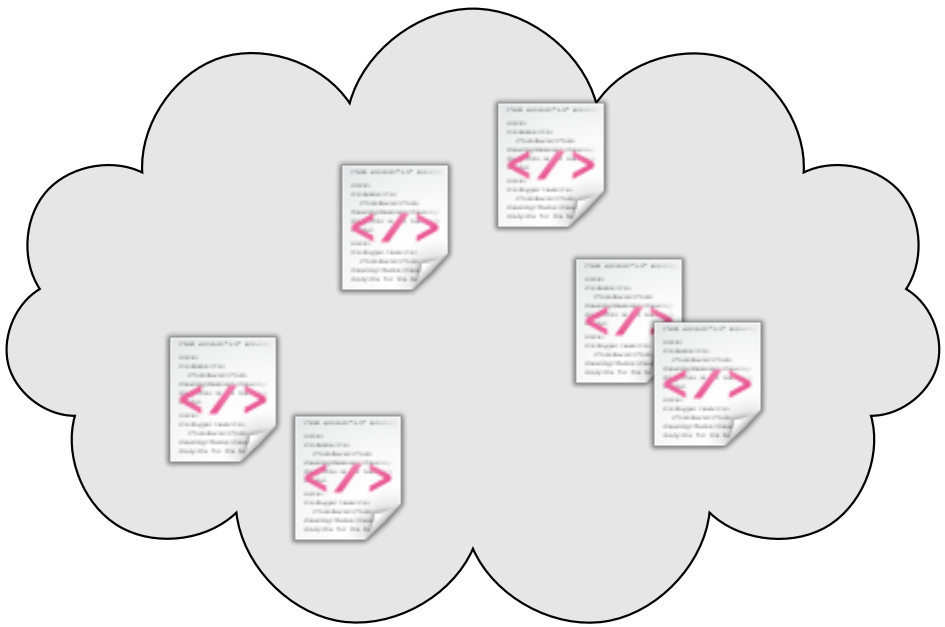
- Applications typically expect XML documents of a **specific form**: XHTML, XML Recipes, ...
- How can this form be described?
- How can applications check that an input document has the requested form?

Implementing a specialized validation tool for each XML Language separately is not the solution...

XML Languages, Schemas, and validation

Definition

- An **XML Language is a set of XML documents** that belong to the same "application domain"



Examples:

- XHTML
- XML Recipes
- ...

XML Languages, Schemas, and validation

Definition

- A **schema** is a formal definition of the syntax of an XML language
- A document is either **valid** w.r.t. a schema, or not

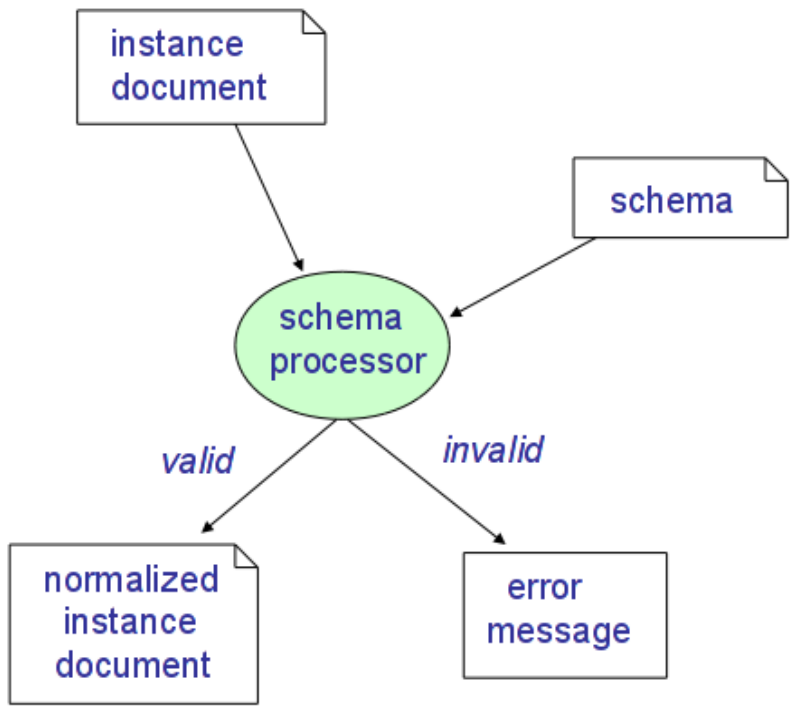
```
<!ELEMENT store (order*, stock)>
<!ELEMENT order (customer, item+)>
<!ELEMENT customer (name, email*)>
<!ELEMENT item (id, price
        +(qty, (supplier + item+)))>
<!ELEMENT stock (item*)>
<!ELEMENT supplier (first, last, email*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ATTLIST price reductionCDTATA#IMPLIED>
```

defines

The benefit of having schemas

- Formal but human-readable descriptions
- Documents can be **validated** automatically with existing schema processors (no need to "roll your own")

The idea of validation:



XML Languages, Schemas, and validation

Definition

- A **schema language** is a notation by which schemas can be defined.

XML Languages, Schemas, and validation

Definition

- A **schema language** is a notation by which schemas can be defined.

Here we will study two schema languages

- **DTDs**: Document Type Definitions
- **XSDs**: XML Schema Defintions (next lesson)

General Schema Language Requirements

- Expressiveness
- Efficiency
- Comprehensibility

General Schema Language Requirements

- Expressiveness
- Efficiency
- Comprehensibility

Here we will study two schema languages

- **DTDs**: limited expressiveness, very efficient, straightforward to use
- **XSDs**: greater expressiveness, efficient, more difficult to use

Intermezzo: Regular Expressions (Syntax)

Definition

- Let Σ be an **alphabet** (typically the set of all Unicode characters or the set of all element names)
- A **regular expression** is an expression built from the following rules
  1. each $\sigma \in \Sigma$ is by itself a regular expression
  2. if α and β are regular expressions, then the following are also regular expressions:

$$\alpha? \qquad \alpha* \qquad \alpha+ \qquad \alpha\,\beta \qquad \alpha \,|\, \beta \qquad (\alpha)$$

Intermezzo: Regular Expressions (Syntax)

Definition

- Let Σ be an **alphabet** (typically the set of all Unicode characters or the set of all element names)
- A **regular expression** is an expression built from the following rules
  1. each $\sigma \in \Sigma$ is by itself a regular expression
  2. if α and β are regular expressions, then the following are also regular expressions:

$$\alpha? \qquad \alpha* \qquad \alpha+ \qquad \alpha\,\beta \qquad \alpha\,|\,\beta \qquad (\alpha)$$

Intermezzo: Regular Expressions (Syntax)

Definition

- Let Σ be an **alphabet** (typically the set of all Unicode characters or the set of all element names)
- A **regular expression** is an expression built from the following rules
  1. each $\sigma \in \Sigma$ is by itself a regular expression
  2. if α and β are regular expressions, then the following are also regular expressions:

$$\alpha? \qquad \alpha* \qquad \alpha+ \qquad \alpha\,\beta \qquad \alpha\,|\,\beta \qquad (\alpha)$$

Intermezzo: Regular Expressions (Syntax)

Definition

- Let Σ be an **alphabet** (typically the set of all Unicode characters or the set of all element names)
- A **regular expression** is an expression built from the following rules
  1. each $\sigma \in \Sigma$ is by itself a regular expression
  2. if α and β are regular expressions, then the following are also regular expressions:

$$\alpha? \qquad \alpha* \qquad \alpha+ \qquad \alpha\,\beta \qquad \alpha \,|\, \beta \qquad (\alpha)$$

Intermezzo: Regular Expressions (Syntax)

Definition

- Let Σ be an **alphabet** (typically the set of all Unicode characters or the set of all element names)
- A **regular expression** is an expression built from the following rules
  1. each $\sigma \in \Sigma$ is by itself a regular expression
  2. if α and β are regular expressions, then the following are also regular expressions:

$$\alpha? \qquad \alpha* \qquad \alpha+ \qquad \alpha\,\beta \qquad \alpha\,|\,\beta \qquad (\alpha)$$

- A regular expression over the alphabet $\{0, 1, \ldots, 9, -\}$

$$0\,|\,(-?\,(1|2|3|4|5|6|7|8|9)\,(0|1|2|3|4|5|6|7|8|9)*)$$

- A regular expression over the alphabet {caption, col, colgroup, thead, tfoot, tbody, tr}

$$\text{caption?}\ (\text{col}*\,|\,\text{colgroup}*)\ \text{thead?}\ \text{tfoot?}\ (\text{tbody}+\,|\,\text{tr}+)$$

Intermezzo: Regular Expressions (Semantics)

Intuitively, a regular expression matches a sequence over Σ:

- $\sigma \in \Sigma$ matches only σ
- $\alpha?$ matches zero or one α
- $\alpha*$ matches zero or more α's
- $\alpha+$ matches one or more α's
- $\alpha\,\beta$ matches any concatenation of an α and a β
- $\alpha \mid \beta$ matches the union of α and β

Intermezzo: Regular Expressions (Semantics)

Intuitively, a regular expression matches a sequence over Σ:

- $\sigma \in \Sigma$ matches only σ
- $\alpha?$ matches zero or one α
- $\alpha*$ matches zero or more α's
- $\alpha+$ matches one or more α's
- $\alpha\,\beta$ matches any concatenation of an α and a β
- $\alpha \mid \beta$ matches the union of α and β

Consider

$$0 \mid (-? \, (1|2|3|4|5|6|7|8|9) \, (0|1|2|3|4|5|6|7|8|9)* \,)$$

Intermezzo: Regular Expressions (Semantics)

Intuitively, a regular expression matches a sequence over Σ:

- $\sigma \in \Sigma$ matches only σ
- $\alpha?$ matches zero or one α
- $\alpha*$ matches zero or more α's
- $\alpha+$ matches one or more α's
- $\alpha\,\beta$ matches any concatenation of an α and a β
- $\alpha \mid \beta$ matches the union of α and β

Consider

$$0 \mid (-? \, (1|2|3|4|5|6|7|8|9) \, (0|1|2|3|4|5|6|7|8|9)* \,)$$

- It matches 0; it also matches 123

Intermezzo: Regular Expressions (Semantics)

Intuitively, a regular expression matches a sequence over Σ:

- $\sigma \in \Sigma$ matches only σ
- $\alpha?$ matches zero or one α
- $\alpha*$ matches zero or more α's
- $\alpha+$ matches one or more α's
- $\alpha\,\beta$ matches any concatenation of an α and a β
- $\alpha \mid \beta$ matches the union of α and β

Consider

$$0 \mid (-? \, (1|2|3|4|5|6|7|8|9) \, (0|1|2|3|4|5|6|7|8|9)* \,)$$

- It matches 0; it also matches 123
- It does not match 01; neither does it match 3.14

Intermezzo: Regular Expressions (Semantics)

Intuitively, a regular expression matches a sequence over Σ:

- $\sigma \in \Sigma$ matches only σ
- $\alpha?$ matches zero or one α
- $\alpha*$ matches zero or more α's
- $\alpha+$ matches one or more α's
- $\alpha\,\beta$ matches any concatenation of an α and a β
- $\alpha \mid \beta$ matches the union of α and β

Consider

$$0 \mid (-? \, (1|2|3|4|5|6|7|8|9) \, (0|1|2|3|4|5|6|7|8|9)* \,)$$

- It matches 0; it also matches 123
- It does not match 01; neither does it match 3.14
- Does it match -19320?

Intermezzo: Regular Expressions (Semantics)

Definition

Formally, define the set $\mathcal{L}(\alpha)$ of all sequences matched by a regular expression α by induction on α:

- $\mathcal{L}(\sigma) = \{\sigma\}$
- $\mathcal{L}(\alpha?) = \{\varepsilon\} \cup \mathcal{L}(\alpha)$
- $\mathcal{L}(\alpha*) = \{s_1 \dots s_n \mid n \geq 0, \text{every } s_i \in \mathcal{L}(\alpha)\}$
- $\mathcal{L}(\alpha+) = \{s_1 \dots s_n \mid n \geq 1, \text{every } s_i \in \mathcal{L}(\alpha)\}$
- $\mathcal{L}(\alpha\,\beta) = \{s_1 s_2 \mid s_1 \in \mathcal{L}(\alpha), s_2 \in \mathcal{L}(\beta)\}$
- $\mathcal{L}(\alpha \mid \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$

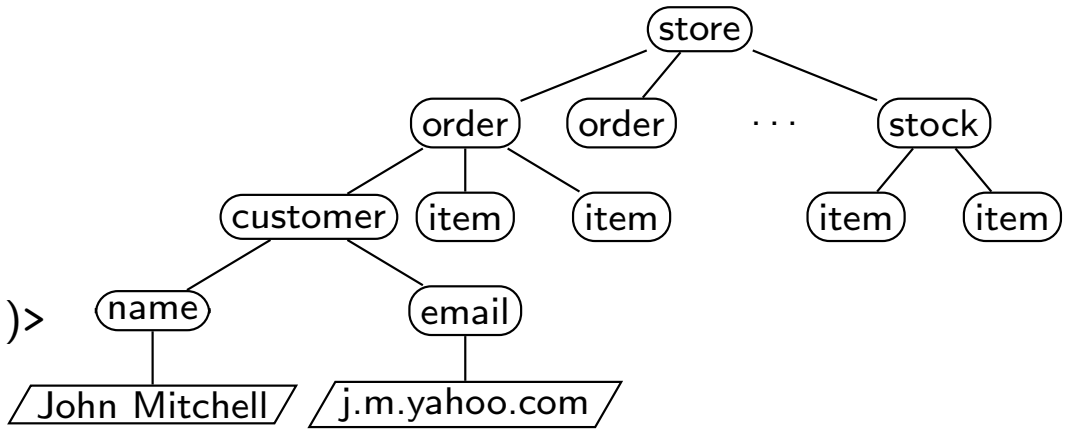
Here, ε denotes the **empty sequence**. We call $\mathcal{L}(\alpha)$ also the **language** of α.

DTDs: Document Type Definitions

- Defined as a subset of the DTD formalism from SGML
- Specified as an integral part of XML 1.0
- A starting point for development of more expressive schema languages
- Considers elements, attributes, and character data – processing instructions and comments are mostly ignored; no support for namespaces

Example DTD

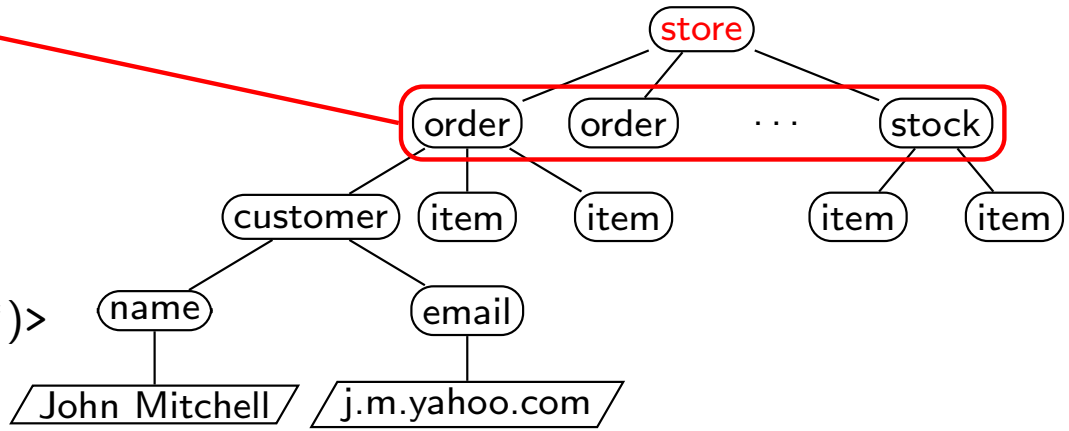
```
<!ELEMENT store (order*, stock)>
<!ELEMENT order (customer, item+)>
<!ELEMENT customer (name, email*)>
<!ELEMENT item (id, price
        | (qty, (supplier | item+)))>
<!ELEMENT stock (item*)>
<!ELEMENT supplier (first, last, email*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ATTLIST price reduction CDATA #IMPLIED>
```



- DTD provides **content models** for elements
- Specifies attribute names for elements, plus their legal values

Example DTD

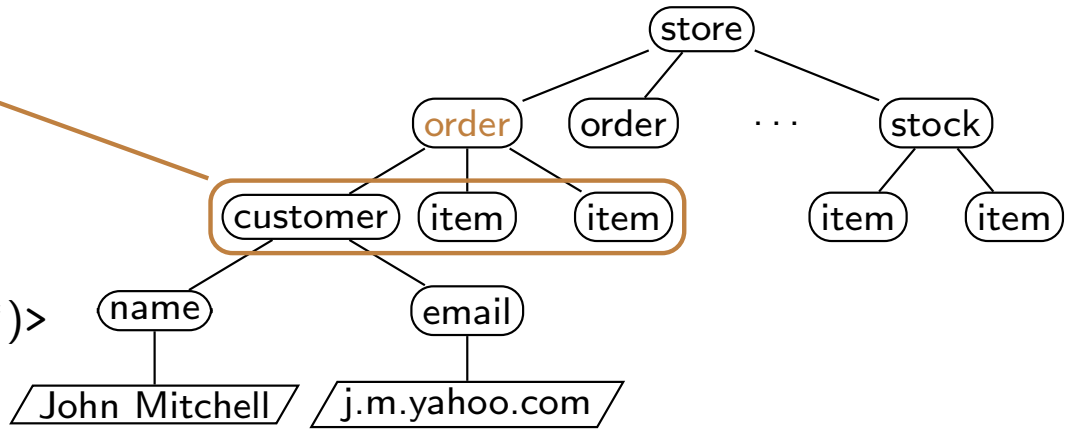
```
<!ELEMENT store (order*, stock)>
<!ELEMENT order (customer, item+)>
<!ELEMENT customer (name, email*)>
<!ELEMENT item (id, price
        | (qty, (supplier | item+)))>
<!ELEMENT stock (item*)>
<!ELEMENT supplier (first, last, email*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ATTLIST price reduction CDATA #IMPLIED>
```



- DTD provides **content models** for elements
- Specifies attribute names for elements, plus their legal values

Example DTD

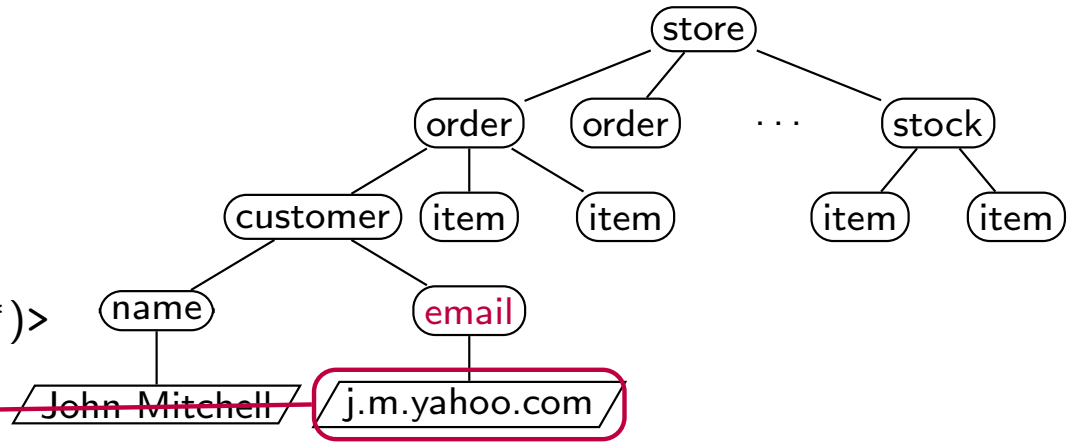
```
<!ELEMENT store (order*, stock)>
<!ELEMENT order (customer, item+)>
<!ELEMENT customer (name, email*)>
<!ELEMENT item (id, price
        | (qty, (supplier | item+)))>
<!ELEMENT stock (item*)>
<!ELEMENT supplier (first, last, email*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ATTLIST price reduction CDATA #IMPLIED>
```



- DTD provides **content models** for elements
- Specifies attribute names for elements, plus their legal values

Example DTD

```
<!ELEMENT store (order*, stock)>
<!ELEMENT order (customer, item+)>
<!ELEMENT customer (name, email*)>
<!ELEMENT item (id, price
        | (qty, (supplier | item+)))>
<!ELEMENT stock (item*)>
<!ELEMENT supplier (first, last, email*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ATTLIST price reduction CDATA #IMPLIED>
```



- DTD provides **content models** for elements
- Specifies attribute names for elements, plus their legal values

DTDs: Document Type Definitions: Syntax

Definition

Syntactically, a **DTD** is a collection of:

- **element declarations**
- **attribute-list declarations**
- **entity declarations**

Element declarations

Definition

- An **element declaration** is of the form

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

  A **content model** can be:

  - `EMPTY`
  - `ANY`
  - `#PCDATA`
  - a regular expression over element names (concatenation with ",")
  - (`#PCDATA` | e_1 | e_2 | $\cdots$ | e_n)$*$ (this is called **mixed content**)

Element declarations

Definition

- An **element declaration** is of the form

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

  A **content model** can be:

  - `EMPTY`
  - `ANY`
  - `#PCDATA`
  - a regular expression over element names (concatenation with ",")
  - (`#PCDATA` | e_1 | e_2 | $\cdots$ | e_n)$*$ (this is called **mixed content**)

Example:

- `<!ELEMENT store (order*, stock)>`
- `<!ELEMENT description ((#PCDATA | name | price)*) >`
- `<!ELEMENT is_on_offer EMPTY >`

Attribute declarations

Definition

- An **attribute declaration** is of the form

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

- *attribute-definitions* is a sequence of attribute definitions, separated by whitespace
- An **attribute definition** consists of 3 components
  - an **attribute name**
  - an **attribute type**
  - a **default declaration**

Attribute declarations

Definition

- An **attribute declaration** is of the form

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

- *attribute-definitions* is a sequence of attribute definitions, separated by whitespace
- An **attribute definition** consists of 3 components
  - an **attribute name**
  - an **attribute type**
  - a **default declaration**

Example:

- `<!ATTLIST price reduction CDTATA #IMPLIED>`
- `<!ATTLIST input maxlength CDTATA #IMPLIED tabindex CDTATA #REQUIRED >`

Attribute declarations: attribute types

Definition

An **attribute type** is either

- `CDATA`: any value
- $(s_1 \mid s_2 \mid \cdots \mid s_n)$: an enumeration of possible values
- `ID`: must have a unique value across the document
- `IDREF`: value must occur in an `ID` attribute somewhere in the document
- `IDREFS`: list of `IDREF`, separated by whitespace

Example:

- `<!ATTLIST price reduction CDATA #IMPLIED>`

Attribute declarations: attribute types

Definition

An **attribute type** is either

- `CDATA`: any value
- $(s_1 \mid s_2 \mid \cdots \mid s_n)$: an enumeration of possible values
- `ID`: must have a unique value across the document
- `IDREF`: value must occur in an `ID` attribute somewhere in the document
- `IDREFS`: list of `IDREF`, separated by whitespace

Example:

- `<!ATTLIST price reduction CDATA #IMPLIED>`
- `<!ATTLIST p align (left | center | right | justify) #IMPLIED>`

Attribute declarations: attribute types

Definition

An **attribute type** is either

- `CDATA`: any value
- $(s_1 \mid s_2 \mid \cdots \mid s_n)$: an enumeration of possible values
- `ID`: must have a unique value across the document
- `IDREF`: value must occur in an `ID` attribute somewhere in the document
- `IDREFS`: list of `IDREF`, separated by whitespace

Example:

- `<!ATTLIST price reduction CDATA #IMPLIED>`
- `<!ATTLIST p align (left | center | right | justify) #IMPLIED>`
- `<!ATTLIST recipe id ID #IMPLIED>`

Attribute declarations: attribute types

Definition

An **attribute type** is either

- `CDATA`: any value
- $(s_1 \mid s_2 \mid \cdots \mid s_n)$: an enumeration of possible values
- `ID`: must have a unique value across the document
- `IDREF`: value must occur in an `ID` attribute somewhere in the document
- `IDREFS`: list of `IDREF`, separated by whitespace

Example:

- `<!ATTLIST price reduction CDATA #IMPLIED>`
- `<!ATTLIST p align (left | center | right | justify) #IMPLIED>`
- `<!ATTLIST recipe id ID #IMPLIED>`
- `<!ATTLIST related ref IDREF #IMPLIED>`

Attribute declarations: default declarations

Definition

An **attribute default declaration** is either

- `#REQUIRED`: must always be present
- `#IMPLIED`: optional
- `"value"`: optional, if attribute is not present, *value* will be used as default
- `#FIXED "value"`: required, must have this value

Attribute declarations: default declarations

Definition

An **attribute default declaration** is either

- `#REQUIRED`: must always be present
- `#IMPLIED`: optional
- `"`*`value`*`"`: optional, if attribute is not present, *`value`* will be used as default
- `#FIXED "`*`value`*`"`: required, must have this value

Attribute declarations: default declarations

Definition

An **attribute default declaration** is either

- `#REQUIRED`: must always be present
- `#IMPLIED`: optional
- `"`*`value`*`"`: optional, if attribute is not present, *`value`* will be used as default
- `#FIXED "`*`value`*`"`: required, must have this value

Attribute declarations: default declarations

Definition

An **attribute default declaration** is either

- `#REQUIRED`: must always be present
- `#IMPLIED`: optional
- `"`*`value`*`"`: optional, if attribute is not present, *`value`* will be used as default
- `#FIXED "`*`value`*`"`: required, must have this value

Attribute declarations: default declarations

Definition

An **attribute default declaration** is either

- `#REQUIRED`: must always be present
- `#IMPLIED`: optional
- `"`*value*`"`: optional, if attribute is not present, *value* will be used as default
- `#FIXED "`*value*`"`: required, must have this value

Example:

```
<!ATTLIST   form
            action CDATA #REQUIRED
            onsubmit CDATA #IMPLIED
            method (get | post) "get"
            enctype CDATA "application/x-www-form-urlencoded">
```

Attribute declarations: default declarations

Definition

An **attribute default declaration** is either

- `#REQUIRED`: must always be present
- `#IMPLIED`: optional
- `"`*value*`"`: optional, if attribute is not present, *value* will be used as default
- `#FIXED "`*value*`"`: required, must have this value

Example:

```
<!ATTLIST   form
            action CDATA #REQUIRED
            onsubmit CDATA #IMPLIED
            method (get | post) "get"
            enctype CDATA "application/x-www-form-urlencoded">

<!ATTLIST   html
            xmlns CDATA #FIXED "http : //www.w3.org/1999/xhtml">
```

Attribute declarations: default declarations

Example:

```
<!ATTLIST   form
            action CDATA #REQUIRED
            onsubmit CDATA #IMPLIED
            method (get | post) "get"
            enctype CDATA "application/x-www-form-urlencoded">
```

- Input:

```
<form action="http://code.ulb.ac.be/hello.jsp">
...
</form>
```

Attribute declarations: default declarations

Example:

```
<!ATTLIST   form
            action CDATA #REQUIRED
            onsubmit CDATA #IMPLIED
            method (get | post) "get"
            enctype CDATA "application/x-www-form-urlencoded">
```

- Input:

```
<form action="http://code.ulb.ac.be/hello.jsp">
...
</form>
```

- Output after **normalization**:

```
<form action="http://code.ulb.ac.be/hello.jsp"
      method="get"
      enctype="application/x-www-form-urlencoded">
      ...
</form>
```

Entity Declarations

Entity Declarations are a simple macro mechanism

There are 4 kinds of entity declarations

- Internal entity declarations
- Internal parameter entity declarations
- External parsed entity declarations
- External unparsed entity declarations

Internal Entity Declarations

Internal Entity Declarations apply to the instance document

Example:

- In the DTD:

```
<!ENTITY copyrightnotice "Copyright &#169; 2005 Widgets 'R' Us.">
```

Internal Entity Declarations

Internal Entity Declarations apply to the instance document

Example:

- In the DTD:

```
<!ENTITY copyrightnotice "Copyright &#169; 2005 Widgets 'R' Us.">
```

- Input in Input Document:

```
A gadget has a medium size head and a big gizmo subwidget.
&copyrightnotice;
```

Internal Entity Declarations

Internal Entity Declarations apply to the instance document

Example:

- In the DTD:

```
<!ENTITY copyrightnotice "Copyright &#169; 2005 Widgets 'R' Us.">
```

- Input in Input Document:

```
A gadget has a medium size head and a big gizmo subwidget.
&copyrightnotice;
```

- Output after **normalization**:

```
A gadget has a medium size head and a big gizmo subwidget.
Copyright &#169; 2005 Widgets 'R' Us.
```

Internal Parameter Entity Declarations

Internal Parameter Entity Declarations **apply to the DTD**, not the instance document

Example:

- In the DTD:

```
<!ENTITY % Shape "(rect|circle|poly|default)">
```

Internal Parameter Entity Declarations

Internal Parameter Entity Declarations **apply to the DTD**, not the instance document

Example:

- In the DTD:

```
<!ENTITY % Shape "(rect|circle|poly|default)">
```

- Then:

```
<!ATTLIST area shape %Shape; "rect">
```

Corresponds to

```
<!ATTLIST area shape (rect|circle|poly|default) "rect">
```

External Parsed Entity Declarations

External Parsed Entity Declarations References XML Data in other files

Example:

- In the DTD:

```
<!ENTITY widgets SYSTEM "http://www.brics.dk/ixwt/widgets.xml">
```

External Parsed Entity Declarations

External Parsed Entity Declarations References XML Data in other files

Example:

- In the DTD:

```
<!ENTITY widgets SYSTEM "http://www.brics.dk/ixwt/widgets.xml">
```

- Input in Input Document:

```
<items> &widgets; </items>
```

External Parsed Entity Declarations

External Parsed Entity Declarations References XML Data in other files

Example:

- In the DTD:

```
<!ENTITY widgets SYSTEM "http://www.brics.dk/ixwt/widgets.xml">
```

- Input in Input Document:

```
<items> &widgets; </items>
```

- Output after **normalization**:

```
<items> contents of widgets.xml goes here </items>
```

External Unparsed Entity Declarations

External Unparsed Entity Declarations References non-XML Data in other files

Example:

- In the DTD:

```
<!ENTITY widget-image SYSTEM
"http://www.brics.dk/ixwt/widget.gif" NDATA gif >
<!NOTATION gif SYSTEM
"http://www.iana.org/assignments/media-types/image/gif">
```

Specifying a DTD for a document

By means of a **Document Type Declaration**

- External DTD: <!DOCTYPE *rootelem* SYSTEM *url*>
- Internal DTD: <!DOCTYPE *rootelem* [*declarations*] >

Specifying a DTD for a document

By means of a **Document Type Declaration**

- External DTD: <!DOCTYPE *rootelem* SYSTEM *url*>
- Internal DTD: <!DOCTYPE *rootelem* [*declarations*] >

Example:

- External:

```
<?xml version="1.1"?>
<!DOCTYPE collection SYSTEM "http://www.brics.dk/ixwt/recipes.dtd">
<collection> ... </collection>
```

Specifying a DTD for a document

By means of a **Document Type Declaration**

- External DTD: <!DOCTYPE *rootelem* SYSTEM *url*>
- Internal DTD: <!DOCTYPE *rootelem* [*declarations*] >

Example:

- External:

```
<?xml version="1.1"?>
<!DOCTYPE collection SYSTEM "http://www.brics.dk/ixwt/recipes.dtd">
<collection> ... </collection>
```

- Internal:

```
<?xml version="1.1"?>
<!DOCTYPE collection [
  <!ENTITY collection (description,recipy*)>
  ...
]>
<collection> ... </collection>
```

Checking Validity With DTDs

A DTD processor (also called a validating XML parser):

- parses the input document (includes checking well-formedness);
- checks the root element name;
- for each element, checks its contents and attributes;
- inserts default values for attributes, if necessary;
- checks uniqueness and referential constraints (ID/IDREF(S) attributes);
- expands references to internal and external entities.

Specifying RecipyML with DTDs

By means of
Online
demonstration

Problems with the DTD description

- `calories` should contain a non-negative number;
- `protein` should contain a value on the form $N\%$ where N is between 0 and 100;
- `comment` should be allowed to appear anywhere in the contents of `recipe`;
- `unit` should only be allowed in an elements where `amount` is also present;
- nested `ingredient` elements should only be allowed when amount is absent;

Problems with the DTD description

- `calories` should contain a non-negative number;
- `protein` should contain a value on the form $N\%$ where N is between 0 and 100;
- `comment` should be allowed to appear anywhere in the contents of `recipe`;
- `unit` should only be allowed in an elements where `amount` is also present;
- nested `ingredient` elements should only be allowed when amount is absent;

Conclusion: **Our DTD schema permits in some cases too much and in other cases too little!**

Limitations of DTDs

1. Cannot constrain character data
2. Specification of attribute values is too limited
3. Element and attribute declarations are context insensitive
4. Character data cannot be combined with the regular expression content model
5. The content models lack an "interleaving" operator
6. The support for modularity, reuse, and evolution is too primitive
7. The normalization features lack content defaults and proper whitespace control
8. Structured embedded self-documentation is not possible
9. The ID/IDREF mechanism is too simple
10. It does not itself use an XML syntax
11. No support for namespaces

Limitations of DTDs

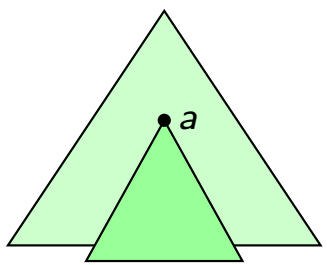
1. Cannot constrain character data
2. Specification of attribute values is too limited
3. **Element and attribute declarations are context insensitive**
4. Character data cannot be combined with the regular expression content model
5. The content models lack an "interleaving" operator
6. The support for modularity, reuse, and evolution is too primitive
7. The normalization features lack content defaults and proper whitespace control
8. Structured embedded self-documentation is not possible
9. The ID/IDREF mechanism is too simple
10. It does not itself use an XML syntax
11. No support for namespaces

What XML languages can we express with a DTD schema?

Observation: There is only one declaration for every element in a DTD D

What XML languages can we express with a DTD schema?

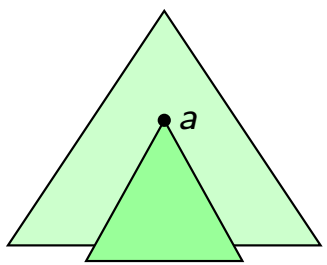
Observation: There is only one declaration for every element in a DTD D

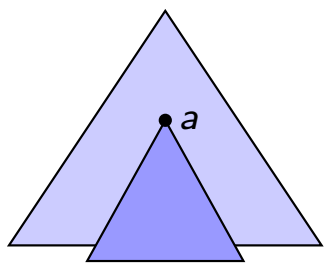
Hence if 

$\in D$

What XML languages can we express with a DTD schema?

Observation: There is only one declaration for every element in a DTD D

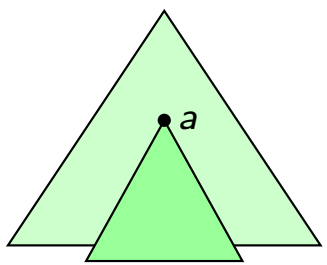
Hence if 

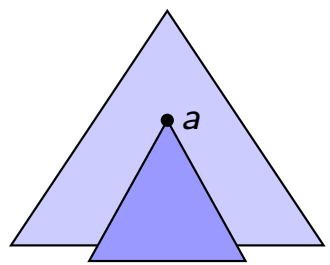
$\in D$ and 

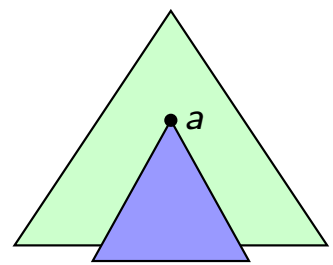
$\in D$

What XML languages can we express with a DTD schema?

Observation: There is only one declaration for every element in a DTD D

Hence if 

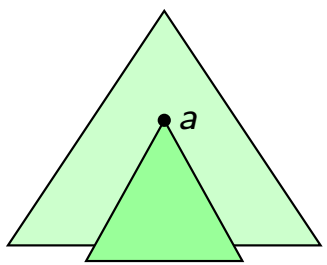
$\in D$ and 

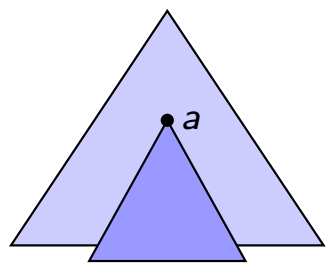
$\in D$ then 

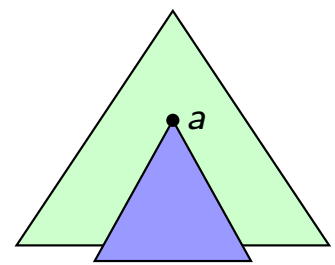
$\in D$

What XML languages can we express with a DTD schema?

Observation: There is only one declaration for every element in a DTD D

Hence if 

$\in D$ and 

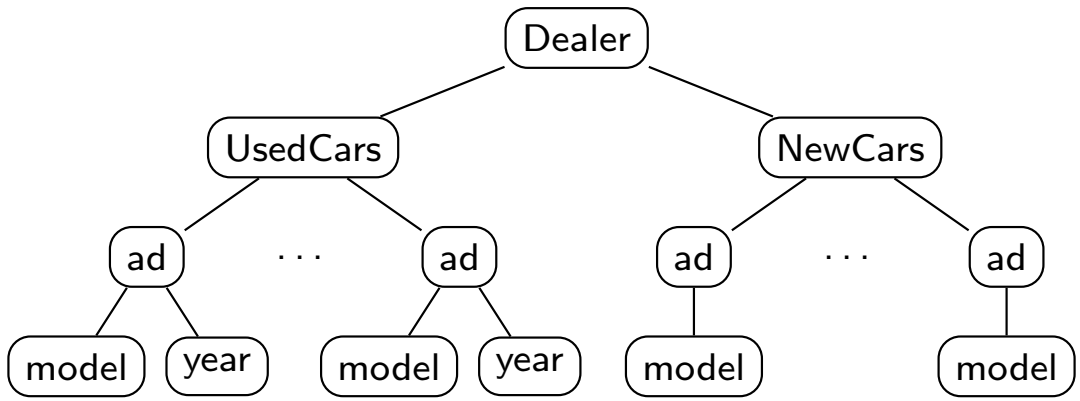
$\in D$ then 

$\in D$

We can use this to show that a tree language is not expressible as a DTD

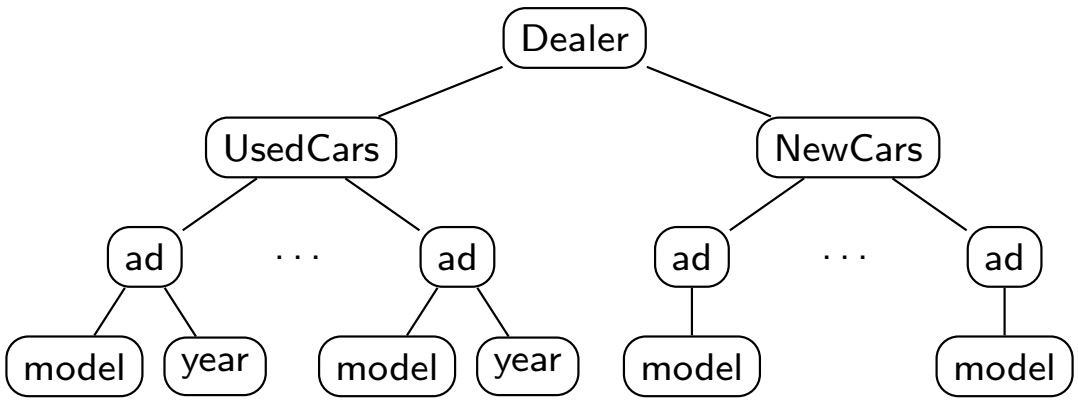
What XML languages can we express with a DTD schema?

Example: there is no DTD recognizing only XML documents of the form



What XML languages can we express with a DTD schema?

Example: there is no DTD recognizing only XML documents of the form

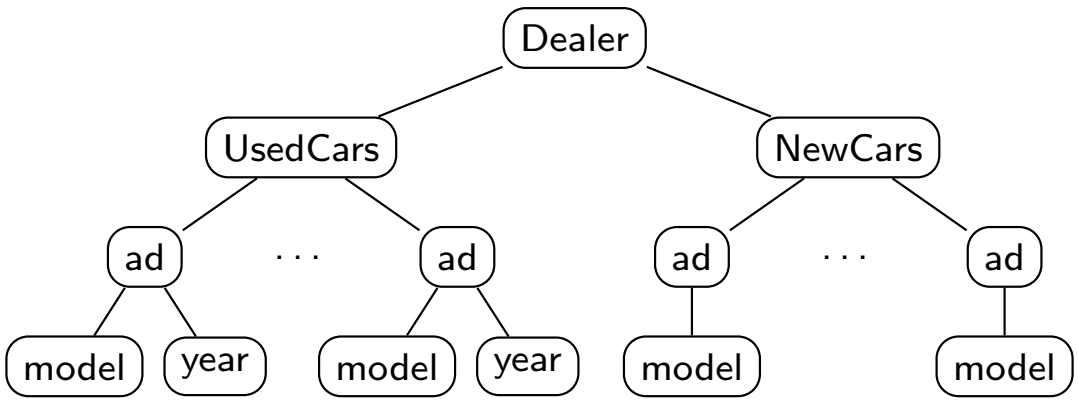


Obviously incorrect:

```
<!DOCTYPE Dealer [
    <!ELEMENT Dealer (UsedCars, NewCars)>
    <!ELEMENT UsedCars (ad*)>
    <!ELEMENT NewCars (ad*)>
    <!ELEMENT ad ((model, year) | model)>
]>
```

What XML languages can we express with a DTD schema?

Example: there is no DTD recognizing only XML documents of the form



Obviously incorrect:

XML Schema will remedy this!

```
<!DOCTYPE Dealer [
    <!ELEMENT Dealer (UsedCars, NewCars)>
    <!ELEMENT UsedCars (ad*)>
    <!ELEMENT NewCars (ad*)>
    <!ELEMENT ad ((model, year) | model)>
]>
```