1. The purpose of using schemas

2. **Regular Expressions** — a commonly used formalism in schema languages

3. The schema language **DTD** and its expressiveness
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?
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...
An **XML Language** is a set of XML documents that belong to the same “application domain”

**Examples:**
- XHTML
- XML Recipes
- ...
**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

```xml
<!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 reductionCDATA #IMPLIED>
```
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:
A schema language is a notation by which schemas can be defined.
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 Definitions (next lesson)
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
Definition

- Let $\Sigma$ be an alphabet (typically the set of all Unicode characters or the set of all element names)
Definition

- Let $\Sigma$ 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
Definition

- Let $\Sigma$ 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
Definition

- Let $\Sigma$ 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 $\alpha$ and $\beta$ are regular expressions, then the following are also regular expressions:

\[
\alpha? \quad \alpha^* \quad \alpha + \quad \alpha \beta \quad \alpha \mid \beta \quad (\alpha)
\]
Let $\Sigma$ 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 $\alpha$ and $\beta$ are regular expressions, then the following are also regular expressions:
   $$\alpha? \quad \alpha* \quad \alpha+ \quad \alpha\beta \quad \alpha|\beta$$

A regular expression over the alphabet \{0, 1, ..., 9, −\}
$$0 \mid (\sim? (1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*)$$

A regular expression over the alphabet \{caption, col, colgroup, thead, tfoot, tbody, tr\}
$$caption? (col*|colgroup*) thead? tfoot? (tbody+|tr+)$$
Intuitively, a regular expression matches a sequence over $\Sigma$:

- $\sigma \in \Sigma$ matches only $\sigma$
- $\alpha?$ matches zero or one $\alpha$
- $\alpha*$ matches zero or more $\alpha$’s
- $\alpha+$ matches one or more $\alpha$’s
- $\alpha \beta$ matches any concatenation of an $\alpha$ and a $\beta$
- $\alpha \mid \beta$ matches the union of $\alpha$ and $\beta$
Intuitively, a regular expression matches a sequence over $\Sigma$:

- $\sigma \in \Sigma$ matches only $\sigma$
- $\alpha?$ matches zero or one $\alpha$
- $\alpha^*$ matches zero or more $\alpha$’s
- $\alpha^+$ matches one or more $\alpha$’s
- $\alpha \beta$ matches any concatenation of an $\alpha$ and a $\beta$
- $\alpha | \beta$ matches the union of $\alpha$ and $\beta$

Consider

$$0 | ( \neg ? (1|2|3|4|5|6|7|8|9) (0|1|2|3|4|5|6|7|8|9)^* )$$
Intuitively, a regular expression matches a sequence over $\Sigma$:

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

Consider

$$0 \mid (\neg (?) \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*)$$

- It matches 0; it also matches 123
Intuitively, a regular expression matches a sequence over $\Sigma$:

- $\sigma \in \Sigma$ matches only $\sigma$
- $\alpha?$ matches zero or one $\alpha$
- $\alpha*$ matches zero or more $\alpha$’s
- $\alpha+$ matches one or more $\alpha$’s
- $\alpha \beta$ matches any concatenation of an $\alpha$ and a $\beta$
- $\alpha | \beta$ matches the union of $\alpha$ and $\beta$

Consider

0 | (−? (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
Intuitively, a regular expression matches a sequence over $\Sigma$:

- $\sigma \in \Sigma$ matches only $\sigma$
- $\alpha?$ matches zero or one $\alpha$
- $\alpha*$ matches zero or more $\alpha$’s
- $\alpha+$ matches one or more $\alpha$’s
- $\alpha \beta$ matches any concatenation of an $\alpha$ and a $\beta$
- $\alpha | \beta$ matches the union of $\alpha$ and $\beta$

Consider

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

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

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

- \( L(\sigma) = \{ \sigma \} \)
- \( L(\alpha?) = \{ \varepsilon \} \cup L(\alpha) \)
- \( L(\alpha*) = \{ s_1 \ldots s_n \mid n \geq 0, \text{every } s_i \in L(\alpha) \} \)
- \( L(\alpha+) = \{ s_1 \ldots s_n \mid n \geq 1, \text{every } s_i \in L(\alpha) \} \)
- \( L(\alpha \beta) = \{ s_1s_2 \mid s_1 \in L(\alpha), s_2 \in L(\beta) \} \)
- \( L(\alpha | \beta) = L(\alpha) \cup L(\beta) \)

Here, \( \varepsilon \) denotes the empty sequence. We call \( L(\alpha) \) also the language of \( \alpha \).
• 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
<!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

```xml
<!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
<!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
<!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
Definition

Syntactically, a DTD is a collection of:

- element declarations
- attribute-list declarations
- entity declarations
An element declaration is of the form

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

  A content model can be:
  - EMPTY
  - ANY
  - #PCDATA
  - a regular expression over element names (concatenation with “,”)
  - ( #PCDATA | e₁ | e₂ | ... | eₙ )* (this is called mixed content)

Example:

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

- An **attribute declaration** is of the form
  ```xml
  <!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:**

- ```xml
  <!ATTLIST price reduction CDATA #IMPLIED>
  ```
- ```xml
  <!ATTLIST input maxlength CDATA #IMPLIED tabindex CDATA #REQUIRED >
  ```
ATTRIBUTES DECLARATIONS: ATTRIBUTE TYPES

**Definition**

An attribute type is either

- **CDATA**: any value

**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: ATTRIBUTE TYPES

**Definition**

An **attribute type** is either

- **CDATA**: any value
- \((s_1 | s_2 | \cdots | s_n)\): an enumeration of possible values

**Example:**

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

**Definition**
An attribute type is either
- **CDATA**: any value
- \((s_1 | s_2 | \cdots | s_n)\): an enumeration of possible values
- **ID**: must have a unique value across the document

**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
Attribute declarations: default declarations

**Definition**

An attribute default declaration is either

- **#REQUIRED**: must always be present
- **#IMPLIED**: optional

Example:

```xml
<!ATTLIST form
  action CDATA #REQUIRED
  onsubmit CDATA #IMPLIED
  method (get|post) "get">
```

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

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

```xml
<!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">
```
Example:

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

- Input:

```html
<form action="http://code.ulb.ac.be/hello.jsp">
...
</form>
```
ATTRIBUTES: DEFAULT DE clarify

Example:

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

- Input:

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

- Output after normalization:

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

  ```xml
  <!ENTITY copyrightnotice "Copyright © 2005 Widgets 'R' Us.”>
  ```
Internal Entity Declarations apply to the instance document

Example:

- In the DTD:

  ```xml
  <!ENTITY copyrightnotice "Copyright © 2005 Widgets 'R' Us."
  ```

- Input in Input Document:

  ```xml
  A gadget has a medium size head and a big gizmo subwidget.
  &copyrightnotice;
  ```
Internal Entity Declarations apply to the instance document

Example:
• In the DTD:
  ```xml
  <!ENTITY copyrightnotice "Copyright &© 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 &© 2005 Widgets ’R’ Us.
Internal Parameter Entity Declarations apply to the DTD, not the instance document.

Example:

- In the DTD:

  ```xml
  <!ENTITY % Shape "(rect|circle|poly|default)">
  ```
Internal Parameter Entity Declarations apply to the DTD, not the instance document.

Example:

- In the DTD:
  ```xml
  <!ENTITY % Shape "(rect|circle|poly|default)">
  ```
- Then:
  ```xml
  <!ATTLIST area shape %Shape; "rect">
  ```
  Corresponds to
  ```xml
  <!ATTLIST area shape (rect|circle|poly|default) "rect">
  ```
External Parsed Entity Declarations References XML Data in other files

Example:

- In the DTD:

  ```xml
  <!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:

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

- Input in Input Document:

  ```xml
  <items> &widgets; </items>
  ```
Example:

- In the DTD:

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

- Input in Input Document:

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

- Output after normalization:

  ```xml
  <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:

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

- Internal:
  ```xml
  <?xml version="1.1"?>
  <!DOCTYPE collection [
    <!ENTITY collection (description,recipy*)>
    ...
  ]>
  <collection> ... </collection>
  ```
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 $D$. 
Observation: There is only one declaration for every element in a DTD $D$.

Hence if $a \in D$
**Observation:** There is only one declaration for every element in a DTD $D$.

Hence if $a \in D$ and $a \in D$, we can use this to show that a tree language is not expressible as a DTD $D$. 
Observation: There is only one declaration for every element in a DTD $D$. Hence if $\in D$ and $\in D$ then $\in D$. 
**Observation:** There is only one declaration for every element in a DTD $D$.

Hence if $a \in D$ and $a \in D$ then $a \in D$.

We can use this to show that a tree language is not expressible as a DTD.
Example: there is no DTD recognizing only XML documents of the form

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

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

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

Obviously incorrect:
WHAT XML LANGUAGES CAN WE EXPRESS WITH A DTD SCHEMA?

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

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

Obviously incorrect:

```
XML Schema will remedy this!
```

NewCars

```
SoldCars

ad

model year
```

UsedCars

```
SoldCars

ad

model year
```

Dealer