XML
Integrity Constraints

Mirian Halfeld Ferrari

European Master’s Program - Information Technologies for Business Intelligence
Bibliographic notes

- Béatrice Bouchou, Mírian Halfeld Ferrari Alves, Maria Adriana Vidigal de Lima. **Attribute Grammar for XML Integrity Constraint Validation.** DEXA (1) 2011 : 94-109


- Béatrice Bouchou, Mírian Halfeld Ferrari, Maria Adriana Lima. **Contraintes d’intégrité pour XML. Visite guidée par une syntaxe homogène.** Technique et Science Informatiques 28(3) : 331-364 (2009)

- Béatrice Bouchou, Ahmed Cheriat, Mírian Halfeld Ferrari, Dominique Laurent, Maria Adriana Lima, Martin A. Musicante. **Efficient Constraint Validation for Updated XML Database.** Informatica (Slovenia) 31(3) : 285-309 (2007)
Importance of integrity constraints

- Integrity constraints are traditionally part of a schema specification.
- Integrity constraints are important in order to define some semantics and to assure consistence.
- Several constraint languages for XML have been proposed.
- Different kinds of integrity constraints: keys (XKeys), foreign keys (XFK), functional dependencies (XFD), inclusion dependencies (XID)
Paths

A path for an XML tree $t$ is defined by a sequence of tags or labels. Our path languages used to define integrity constraints over XML trees:

- **Path language $PL_s$** (defined by $\rho ::= l | \rho/\rho | \_.$).

- **Path language $PL$** (defined by $\nu ::= [ ] | \rho | \nu//\rho$).

The language $PL_s$ describes a path in $t$, while $PL$ is a generalization of $PL_s$ including "//".

A path $P$ is **valid** if it conforms to the syntax of $PL_s$ or $PL$ and for all tag $l \in P$, if $l = data$ or $l \in \Sigma_{att}$, then $l$ is the last symbol in $P$.

Examples: `/project/supplier/component` or `fac//student/courseTaken/courseCode`
Path Instances

Let \( I = v_1 / \ldots / v_n \) be a sequence of positions such that each \( v_i \) is a direct descendant of \( v_{i-1} \) in \( t \).

\( I \) is an instance of \( P \) over \( t \) if and only if the sequence \( t(v_1)/\ldots/t(v_n) \in L(A_P) \). \( A_P \) the finite-state automaton defined according to \( P \).
A **pattern** is a finite set of *prefix-closed* paths in a tree \( t \).

An **instance of a pattern** is defined by considering the longest common prefix and an unique instance for it.

A set of paths: \( \{ \text{fac} / / \text{student} / \text{courseTaken} / \text{courseCode}, \text{fac} / / \text{student} / \text{courseTaken} / \text{deptName} \} \)

Common prefix: \( \text{fac} / / \text{student} / \text{courseTaken} \)

\[ \tau_{C/P,Q} = [C-BD-07S2, \text{Computing}] \]

\[ \tau_{C/P,Q} = [M-1, \text{Math}] \]
Two types of equality

- **Value equality**: two nodes are *value equal* when they are roots of isomorphic sub-trees.
- **Node equality**: two nodes are *node equal* when they are in the same position.

*Nodes 0.1 and 1.1 are value equal.*
The relational case

In a relational database a functional dependency is defined as follows:

- Let $U$ be the set of attributes of a relation schema $R$. We usually write $R[U]$.
- Let $X \subseteq U$ and $A \in U$. An instance $I$ of $R$ satisfies the functional dependency

\[ X \rightarrow A \]

when for all two tuples $u$ and $v$ if $u[X] = v[X]$ then $u[A] = v[A]$

- Given $X \rightarrow A$, we say that $X$ functionally determines $A$.

Consider the functional dependency $Name, Year \rightarrow Activity$

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mario</td>
<td>2010</td>
<td>theatre</td>
</tr>
<tr>
<td>Diana</td>
<td>2010</td>
<td>theatre</td>
</tr>
<tr>
<td>Peter</td>
<td>2010</td>
<td>volleyball</td>
</tr>
<tr>
<td>Barbara</td>
<td>2011</td>
<td>volleyball</td>
</tr>
<tr>
<td>Mario</td>
<td>2011</td>
<td>football</td>
</tr>
</tbody>
</table>
XFD Syntax

An XML functional dependency (XFD) is an expression of the form:

\[
\gamma = (C, \{P_1 [E_1], \ldots, P_k [E_k]\} \rightarrow Q [E])
\]

- \(C, P_1 \ldots P_k\) and \(Q\) are path expressions.
- Path \(C\) represents the context for the dependency verification.
- \(\{P_1, \ldots, P_k\}\) are the determinant paths of the XFD.
- \(Q\) is the dependent path.
- Symbols \(E_1, \ldots, E_k\) represent the associated equality type.
XFD Semantics

Let

- XML document $\mathcal{T}$
- XFD $\gamma = (C, \{P_1[E_1], \ldots, P_k[E_k]\} \rightarrow Q[E])$
- Pattern $\mathcal{P} : \{C/P_1, \ldots, C/P_k, C/Q\}$

XFD satisfaction

$\mathcal{T} \models \gamma$ if and only if for all two instances $(I_1, I_2)$ of pattern $\mathcal{P}$ in $\mathcal{T}$ that coincide at least on their prefix $C$, we have:

$$\tau_1[C/P_1, \ldots, C/P_k] =_{E_i, i \in \{1 \ldots k\}} \tau_2[C/P_1, \ldots, C/P_k] \Rightarrow \tau_1[C/Q] =_E \tau_2[C/Q]$$

where $\tau_1$ (resp. $\tau_2$) is the tuple obtained from $I_1$ (resp. $I_2$).
**XFD Example**

\[ \gamma_1 : (db, ( \{ /project/supplier/@sname[V],
                    /project/supplier/component/@cname[V] \} \\
                \rightarrow /project/supplier/component/price[V])) ) \]

```
[ MSI, 955X Neo, 185.00 ]
[ MSI, K8N, 182.90 ]
```
A general integrity constraint validation method

- **Our goal**: integrity constraint validation on XML documents.

- **Our validation method**: a *grammarware* (based on a grammar) describing an XML document to which we associate attributes and semantic rules.

- **Attribute grammar**: the grammar is augmented by semantic rules that define, for each integrity constraint, the verification process.
Building an attribute grammar

- **Attribute grammar**: attach a set of semantic rules to each production of a context-free grammar.
- **General CFG (context free grammar)** to describe an XML tree.
  - Rule for the root element: $\text{ROOT} \rightarrow \alpha_1 \ldots \alpha_m$.
  - Rule for an internal element node: $A \rightarrow \alpha_1 \ldots \alpha_m$.
  - Rule for an element containing data and for an attribute: $A \rightarrow \text{data}$.
To model the paths of an XFD, we use finite-state automata (FSA) or transducers (FST).

\[ \gamma : (\text{db}/\text{project}, ( \{ /\text{supplier}/@\text{sname}[\text{V}], /\text{supplier}/\text{component}@\text{cname}[\text{V}] \} \rightarrow /\text{supplier}/\text{component}/\text{quantity}[\text{V}]) )) \]

Diagram:

- **M**: The initial state is \( e_0 \), followed by \( db \) to \( e_1 \), then \( \text{project} \) to \( e_2 \).
- **T'**: From \( e_3 \) to \( e_4 \) with \( \text{supplier} \), \( @\text{sname} \) to \( e_6 \) with value \( (\text{V},1) \).
- **T''**: From \( e_8 \) to \( e_9 \) with \( \text{supplier} \) and \( \text{component} \), then to \( e_{10} \) with \( \text{quantity} \) to \( e_{11} \) with value \( \text{V} \).
Attribute Grammar for XFD Validation

Descending Direction: Inherited Attribute $conf$

- $conf$ is used at each node to indicate its role concerning an XFD
- its value is a set of FSA configurations
- all nodes are bound to a $conf$ attribute, except data nodes
- $conf$ is an empty set when the node is not in any XFD path
Example: Descending direction
Example: Descending direction
Example: Descending direction

M: \( e_0 \xrightarrow{db} e_1 \xrightarrow{project} e_2 \)

\( T': e_3 \xrightarrow{supplier} e_4 \xrightarrow{\text{@sname} \{V.1\}} e_5 \xrightarrow{\text{component}} e_6 \xrightarrow{\text{@cname}} e_7 \xrightarrow{\{V.2\}} \)

\( T^*: e_8 \xrightarrow{\text{supplier}} e_9 \xrightarrow{\text{component}} e_{10} \xrightarrow{\text{quantity}} e_{11} \)

\( M: \) conf = \{M.e_1\}

\( T': \) conf = \{T'.e_4, T''.e_9\}

\( T^*: \) conf = \{T'.e_6, T''.e_{10}\}

\( \text{project} \)

\( \text{conf} = \{} \)

\( \text{conf} = \{T'.e_5\} \)

\( \text{conf} = \{T'.e_6, T''.e_10\} \)

\( \text{conf} = \{T'.e_7\} \)

\( \text{conf} = \{T'.e_7\} \)

\( \text{conf} = \{T''.e_{11}\} \)

\( \text{conf} = \{T''.e_{11}\} \)

\( \text{conf} = \{T'.e_6, T''.e_{10}\} \)

Mirian Halfeld Ferrari (IT4BI)
Example: Descending direction
Example: Descending direction

- **M**: $\text{db} \rightarrow \text{project} \rightarrow \text{e}_2$
- **T**: $\text{supplier} \rightarrow \text{component} \rightarrow \text{e}_7 | (V.1)$
- **$T'$**: $\text{supplier} \rightarrow \text{component} \rightarrow \text{quantity} \rightarrow \text{e}_{11} | V$

Confidence:

- $\text{conf} = \{M.e_1\}$
- $\text{conf} = \{M.e_2\}$
- $\text{conf} = \{T'.e_4, T''.e_9\}$
- $\text{conf} = \{T'.e_5\}$
- $\text{conf} = \{T'.e_4, T''.e_9\}$
- $\text{conf} = \{T'.e_6\}$
- $\text{conf} = \{T'.e_6, T''.e_{10}\}$
- $\text{conf} = \{T'.e_7\}$
- $\text{conf} = \{T'.e_6, T''.e_{10}\}$
- $\text{conf} = \{T'.e_7\}$
- $\text{conf} = \{T'.e_6, T''.e_{10}\}$
- $\text{conf} = \{T'.e_7\}$
- $\text{conf} = \{T'.e_6, T''.e_{10}\}$
- $\text{conf} = \{T'.e_7\}$
- $\text{conf} = \{T'.e_6, T''.e_{10}\}$
- $\text{conf} = \{T'.e_7\}$
Example: Descending direction

M: $\theta_0 \xrightarrow{db} \theta_1 \xrightarrow{\text{project}} \theta_2$

$T$: $\theta_3 \xrightarrow{\text{supplier}} \theta_4$
$\theta_5 \xrightarrow{@\text{iname}} (V,1)$
$\theta_6 \xrightarrow{\text{component}} \theta_7 \xrightarrow{(V,2)}$

$T^*$: $\theta_8 \xrightarrow{\text{supplier}} \theta_9 \xrightarrow{\text{component}} \theta_{10} \xrightarrow{\text{quantity}} \theta_{11} \xrightarrow{V}$

$db$ $conf = \{ M.e_1 \}$

$0$ $conf = \{ M.e_2 \}$

$pname$ $conf = \{ \}$

$Conf = \{ T'.e_4, T''.e_9 \}$ $supplier$

$Conf = \{ T'.e_6, T''.e_10 \}$ $component$

$Conf = \{ T'.e_7 \}$ $price$

$Conf = \{ T''.e_11 \}$ $quantity$

$Conf = \{ T'.e_5 \}$

$Conf = \{ \}$

$Conf = \{ T'.e_6, T''.e_10 \}$

$Conf = \{ \}$

$Conf = \{ \}$

$Conf = \{ T'.e_7 \}$

$Conf = \{ \}$

$Conf = \{ \}$

$Conf = \{ \}$

$Conf = \{ T''.e_11 \}$
Example: Descending direction
Just an example of a rule in the attribute grammar

<table>
<thead>
<tr>
<th>Production</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{ROOT} \to \alpha_1 \ldots \alpha_m )</td>
<td>( \text{ROOT}.\text{conf} := { M.q_1 \mid \delta_M(q_0, \text{ROOT}) = q_1 } )</td>
</tr>
</tbody>
</table>

\(/* \text{Inherited Attributes} */\)

for each \( \alpha_i \ (1 \leq i \leq m) \) do

\[ \alpha_i.\text{conf} := \{ M.q' \mid \delta_M(q_1, \alpha_i) = q' \} \]

if \( (q_1 \in F_M) \) then

\[ \alpha_i.\text{conf} := \alpha_i.\text{conf} \cup \{ T'.q'_1 \mid \delta_{T'}(q'_0, \alpha_i) = q'_1 \} \]
\[ \cup \{ T''.q''_1 \mid \delta_{T''}(q''_0, \alpha_i) = q''_1 \} \]
Attribute Grammar for XFD Validation

**Ascending Direction**: Synthesized Attributes $c, \textit{inters}, dc, ds_j$.

- $c$ carries the dependency validity (true or false) from context level to the root.
- $\textit{inters}$ gathers the values from the nodes that are in determinant and dependent path intersections.
- $ds_j$ and $dc$ store the values needed to verify the dependency.
Example: Ascending direction

Computing synthesized attributes:

```
c = <true>
db
  c = <true>
  project
    0
      c = <true>
      Inter = {<<MSI, 955XNeo>, 5>, <<MSI, K8N>, 7>}
      ds1 = <MSI>
      ds2 = <955XNeo>
        dc = <5>
        inters = {<<epsilon, 955XNeo>, 5>}
      inters = {<<epsilon, K8N>, 7>}
      ds2 = <K8N>
        dc = <7>
```

Mirian Halfeld Ferrari (IT4BI)
Example: Ascending direction

Computing synthesized attributes:

```
dc = < 5 >
ds = 955X Neo
inters = { << MSI, 955XNeo >, 5 >, << MSI, K8N >, 7 > }
ds = K8N
dc = 7
inters = { << K8N, 7 > }
ds = MSI
dc = 5
inters = { << MSI, 955XNeo >, 5 >, << MSI, K8N >, 7 > }
```

```
dc = < 7 >
ds = 955X Neo
inters = { << 955XNeo, 5 >, << K8N, 7 > }
ds = K8N
inters = { << 955XNeo, 5 >, << K8N, 7 > }
dc = 5
inters = { << MSI, 955XNeo >, 5 >, << MSI, K8N >, 7 > }
```

```
dc = < 7 >
ds = 955X Neo
inters = { << 955XNeo, 5 >, << K8N, 7 > }
dc = 5
inters = { << 955XNeo, 5 >, << K8N, 7 > }
dc = < 7 >
ds = 955X Neo
inters = { << 955XNeo, 5 >, << K8N, 7 > }
```

```
dc = < 7 >
ds = 955X Neo
inters = { << 955XNeo, 5 >, << K8N, 7 > }
dc = 5
inters = { << 955XNeo, 5 >, << K8N, 7 > }
```

```
data 185,00
price 5
quantity 9
```

```
data 182,90
price 7
```

```
data 185,00
price 5
quantity 9
```

```
data 182,90
price 7
```
Example: Ascending direction

Computing synthesized attributes:
Example: Ascending direction

Computing synthesized attributes:

```
db = (true)

project = (true)

pname = Proj1

@fname = MSI

supplier =
  component =
    component =
      component =
        price =
          data = 185,00
        quantity = 5
      price =
        data = 182,90
      quantity =
        data = 7
```
### Just an example of a rule in the attribute grammar

<table>
<thead>
<tr>
<th>Production</th>
<th>Attributes</th>
</tr>
</thead>
</table>
| $A \rightarrow data$ | /* Synthesized Attributes */
| | for each configuration $M.q$ in $A.conf$ do
| |   if $(M = T') \land (q \in F_{T'})$
| |     $y := \lambda'_{T'}(q)$
| |     $j := y.rank$
| |     if ($y.equality = V$)
| |       then $A.ds_{j} := <value(t, data)>$
| |     else $A.ds_{j} := <value(t, A)>$
| |   if $(M = T'') \land (q \in F_{T''})$
| |     if $(\lambda''_{T''}(q) = V)$
| |       then $A.dc := <value(t, data)>$
| |     else $A.dc := <pos(t, A)>$ |
XFD Validation Overview

XML Document

XFDs

Validation using Attribute Grammar

TRUE / FALSE (for each constraint)

Auxiliary Data Structures

InhAtt_Stack

SyntAtt_Stack

Validation_HTables

Mirian Halfeld Ferrari (IT4BI)
XFD Validation Overview: Auxiliary Structures

**InhAtt_Stack**

- `xdf_1`, `xdf_2`, ..., `xdf_n`
- `{ M_r.e_j1, M_r.e_k1, ... }`
- Set of configurations for `xdf_n` at a node

**SyntAtt_Stack**

- `xdf_1`, `xdf_2`, ..., `xdf_n`
- `xdf_n` is linked to a set of configurations

**Validation_HTables**

- `l_1`, `l_2`, `l_1'`, `l_2'`, `...`
- The values in tuples `<l_1,l_2>` are maintained in a hashtable, which is used to verify the XFD.

- `c`: list of boolean values
- `ds`: list of strings
- `dc`: string
- `inters`: list of tuples `<l_1,l_2>`
  - `l_1` for `ds` and `l_2` for `dc`
<table>
<thead>
<tr>
<th>Constr.</th>
<th>Path expression</th>
<th>FSA</th>
<th>Attributes</th>
</tr>
</thead>
</table>
| XFD     | $(C, \{\{P_1 [E_1], \ldots, P_k [E_k]\}\} \rightarrow Q [E]))$ | $M, T$ and $T'$ | Inherit. : : $conf$
|         |                 |     | Synth. : $c, inters, ds_j, ds_c$ |
| XID     | $(C,\{P_1, \ldots, P_k\} \subseteq \{Q_1 \ldots Q_k\}))$ | $M, T$ and $T'$ | Inherit. : : $conf$
|         |                 |     | Synth. : $c, inters, ds_j, dc_j$ |
| XKeys   | $(C, (Tg, \{P_1, \ldots, P_k\}))$ | $A_C, A_{Tg}$ et $A_P$ | Inherit. : $conf$
|         |                 |     | Synth. : $c, tg$ et $f$ |
| XFK     | $(C, (Tg^R, \{P_1^R, \ldots, P_k^R\}) \subseteq (Tg, \{P_1, \ldots, P_k\}))$ | $A_C, A_{Tg}^R, A_P^R$ | Inherit : $conf$
|         |                 |     | Synth. : $c, tg$ et $f$ |