Optimized Cartesian Product: A hybrid approach to derivation-chain checking in XSD 1.1

Maurizio Casimirri, Paolo Marinelli, Fabio Vitali

University of Bologna

Outline

- Co-constraints in XSD 1.1
  - CTA
- Derivation by restriction in XSD 1.1
  - General approaches to the problem
  - XSD 1.1 solution
- Optimized Cartesian Product
- Comparison between OCP and XSD 1.1 solution

CO-CONSTRAINTS IN XSD 1.1

Co-constraints

- W3C ESW wiki
  - “rules which govern what kinds of markup (elements, attributes) can occur together (co-occur) in an XML document.”

Co-constraints – A use case (1/2)

- <entry>
  - @kind: proceedings, journal, book
  - <authors>, author list
  - <title>, work title
  - <journal>, journal where work is published
  - <conference>, conference name
  - <pages @from @to>, pages
  - <series>, book series

Co-constraints – A use case (2/2)

```
<entry kind="proceedings">
  <authors>
    <author>…</author>
    <author>…</author>
  </authors>
  <title>….</title>
  <conference>…</conference>
  <pages from="10" to="20" />
</entry>
```

- <conference> iff "proceedings"
- <journal> iff "journal"
- <pages> iff "proceedings" or "journal"
- <series> iff "book"
- pages/@from less than pages/@to

```
<entry kind="journal">
  <authors>
    <author>…</author>
  </authors>
  <title>….</title>
  <journal>…</journal>
  <pages from="10" to="20" />
</entry>
```

```
<entry kind="book">
  <authors>
    <author>…</author>
</authors>
  <title>….</title>
  <series>…</series>
</entry>
```

Co-constraints in XSD 1.1

- Assertions
  - Types definitions
  - <xs:assert test=*“XPath 2.0”*>
- Conditional Type Assignment
  - Element declarations
  - Type depending on predicates
  - <xs:alternative test=*“XPath 2.0”* type=*“Type”*>

Co-constraints in XSD 1.1 – use case solution

```
<xs:element name="entry" type="entryType">
  <xs:alternative test="@kind = 'proceedings'“     type="proceedingEntryType" />
  <xs:alternative test="@kind = 'journal'"         type="journalEntryType" />
  <xs:alternative test="@kind = 'book'"            type="bookEntryType" />
</xs:element>
```

```
<xs:complexType name="pagesType">
   <xs:attribute name="from"          type="xs:positiveInteger"    use="required" />
   <xs:attribute name="to"            type="xs:positiveInteger"    use="required" />
   <xs:assert test="@to gt @from" />
</xs:complexType>
```

CTA – Semantics and Terminology

```
<xs:element name="entry" type="entryType">
  <xs:alternative test="@kind = 'proceedings'"        type="proceedingEntryType" />
  <xs:alternative test="@kind = 'journal'"            type="journalEntryType" />
  <xs:alternative test="@kind = 'book'"               type="bookEntryType" />
</xs:element>
```

- Declared type (entryType)
- Type Table
  - Type alternatives
  - Document order implies priority
  - Type alternatives validly substitutable for the declared type
  - Default type definition
- *xs:error*, unsatisfiable type

DERIVATION BY RESTRICTION IN XSD 1.1

General principle

- The derived type accepts a subset of what the base type accepts
- **B** and **R** types; **E** element
  - **T** type assigned in the context of R
  - **S** type assigned in the context of B
  - If **E** validates against **T** then **E** validates against **S**
  - **T** restricts **S**

Absence of CTA

```
<xs:complexType name="bibliographyType">
 <xs:sequence>
  <xs:element name="entry" type="entryType" maxOccurs="unbounded" />
 </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="myBibliographyType">
 <xs:complexContent>
  <xs:restriction base="bibliographyType">
   <xs:sequence>
    <xs:element name="entry" type="proceedingEntryType" maxOccurs="unbounded" />
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

- proceedingsEntryType **extends** entryType
- myBibliographyType **illegal restriction** of bibliographyType
- **Static verification is possible**
  - Henry S. Thompson and Richard Tobin, 2003
  - Matthew Fuchs and Allen Brown, 2003
  - C. M. Sperberg-McQueen, 2005

Presence of CTA

```
<xs:complexType name="bibliographyType">
 <xs:sequence>
  <xs:element name="entry" type="entryType" maxOccurs="unbounded“ />
 </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="myBibliographyType">
 <xs:complexContent>
  <xs:restriction base="bibliographyType">
   <xs:sequence>
    <xs:element name="entry" type="entryType" maxOccurs="unbounded">
     <xs:alternative test="@kind = 'proceedings' and @kind != 'proceedings'"
                     type="proceedingEntryType" />
    </xs:element>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

General approaches

- Static check through CTA limitation
- Full Dynamic approach
  - Performed during the validation phase
  - Adopted by the specs
- Hybrid approach
  - Some checks performed statically
  - Other checks performed dynamically

Run-time Check (RTC)

- Static phase
  - Ignore type alternatives
- Validation phase
  - *Conditional Type Substitutable in Restriction* (CTSR)
  - **E** element, **R** type
    - **B** base type of R
    - **T** type assigned in the context of R
    - **S** type assigned in the context of B
    - **T** restricts **S**
    - **E** and **B** recursively satisfy CTSR

RTC – example (1/3)

```
<xs:complexType name="bibliographyType">
 <xs:sequence>
  <xs:element name="entry" type="entryType" maxOccurs="unbounded">
   <xs:alternative test="@kind = 'proceedings'"        type="proceedingEntryType" />
   <xs:alternative test="@kind = 'journal'"           type="journalEntryType" />
   <xs:alternative test="@kind = 'book'"              type="bookEntryType" />
  </xs:element>
 </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="myBibliographyType">
 <xs:complexContent>
  <xs:restriction base="bibliographyType">
   <xs:sequence>
    <xs:element name="entry" type="entryType" maxOccurs="unbounded">
     <xs:alternative test="@kind = 'proceedings'"      type="proceedingEntryType" />
     <xs:alternative                                   type="xs:error" />
    </xs:element>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

RTC – example (2/3)

- Static phase
  - entryType restricts itself
  - The schema is valid

```
<xs:complexType name="bibliographyType">
 <xs:sequence>
  <xs:element name="entry" type="entryType" maxOccurs="unbounded“ />
</xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="myBibliographyType">
 <xs:complexContent>
  <xs:restriction base="bibliographyType">
   <xs:sequence>
    <xs:element name="entry" type="entryType" maxOccurs="unbounded“ />
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

RTC – example (3/3)

- Dynamic phase

 <entry kind="proceedings"> ... </entry> 	myBibliographyType proceedingEntryType bibliographyType proceedingEntryType proceedingEntryType restricts itself No validation error
 <entry> ... </entry> 	myBibliographyType xs:error bibliographyType entryType xs:error does not restrict entryType Validation error

OPTIMIZED CARTESIAN PRODUCT

General idea (1/2)

- Static phase
  - Find the XPath predicates affecting the CTSR checking result
- Dynamic phase
  - Check CTSR
  - Use the XPath predicates found in the static phase

General Idea (2/2)

```
<xs:complexType name="bibliographyType">
 <xs:sequence>
  <xs:element name="entry" type="entryType" maxOccurs="unbounded">
   <xs:alternative test="@kind = 'proceedings'"        type="proceedingEntryType" />
   <xs:alternative test="@kind = 'journal'"           type="journalEntryType" />
   <xs:alternative test="@kind = 'book'"              type="bookEntryType" />
  </xs:element>
 </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="myBibliographyType">
 <xs:complexContent>
  <xs:restriction base="bibliographyType">
   <xs:sequence>
    <xs:element name="entry" type="entryType" maxOccurs="unbounded">
     <xs:alternative test="@kind = 'proceedings'"      type="proceedingEntryType" />
     <xs:alternative                                   type="xs:error" />
    </xs:element>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

Static phase

- Visit the type hierarchy
- For each type **T**
  - For each name **e** matching some declaration
    - Get the Type Table associated to **e**
    - Annotate each type alternative with an **error condition**

Dynamic phase

- **T** type, **E** element
- $\mathbf{T_1}$, …, $\mathbf{T_k}$ derivation chain
  - $\mathbf{T_1}$ = anyType
  - $\mathbf{T_k}$ = T
  - $\mathbf{T_{i-1}}$ = T_i's base type
- $\mathbf{TT_i}$ = Type Table for E in T_i
- Evaluate $\mathbf{TT_i}$
  - Evaluate the error condition
    - If true then E and T violate CTSR
    - Otherwise repeat with TT_{i-1}

Error condition construction (1/2)

- **B** base type, **R** restricted type, **e** element name
- $\mathbf{TT_B}$ Type Table for **e** in **B**
- $\mathbf{TT_R}$ Type Table for **e** in **R**
- Boolean expression over $\mathbf{TT_B}$ predicates
- Rewriting phase
  - Lazy boolean evaluation rules

Error condition construction (2/2)

```
<xs:complexType name="myBibliographyType">
 <xs:complexContent>
  <xs:restriction base="bibliographyType">
   <xs:sequence>
    <xs:element name="entry" type="entryType" maxOccurs="unbounded">
     <xs:alternative test="@kind = 'proceedings'"      type="proceedingEntryType" />
     <xs:alternative                                   type="xs:error" />
    </xs:element>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

- <xs:alternative test="@kind = 'proceedings'" type="proceedingEntryType" />
  - not(proceedings) and (journal or book or true())
  - **not(proceedings)**
- <xs:alternative type="xs:error" />
  - proceedings or journal or book or true()
  - **true()**

COMPARISON BETWEEN OCP AND RTC

Single step of the derivation chain

- **R**, restricted type
- **B**, base type
- **E**, element
- **TT_R**, Type Table in R
- **TT_B**, Type Table in B
- **n**, TT_R size
- **m**, TT_B size

- RTC
  - evaluates TT_R, ($\leq$ n)
  - evaluates TT_B, ($\leq$ m)
- OCP
  - evaluates TT_R, ($\leq$ n)
  - evaluates the error condition ($\leq$ m)

Example

```
<p xsi:type="myBibliographyType">
  <entry>
    ...
  </entry>
</p>
```

RTC	OCP
@kind = 'proceedings' true() @kind = 'proceedings' @kind = 'journal' @kind = 'book' true()	@kind = 'proceedings' true() true()

Whole derivation chain

- OCP might evaluate predicates twice
- Mark evaluated predicates

Conclusions

- An optimization of RTC
- Acceptable cost for the static phase
- Clarification of CTA and derivation by restriction