MSL:
A model for W3C XML Schema

Allen Brown, Microsoft

Matthew Fuchs, Commerce One

Jonathan Robie, Software AG

Philip Wadler, Avaya Labs

W3C XML Schema Formalism
(W3C working draft)

(editors)

Allen Brown, Microsoft

Matthew Fuchs, Commerce One

Jonathan Robie, Software AG

Philip Wadler, Avaya Labs

“Where a mathematical reasoning can be had, it’s as great folly to make use of any other, as to grope for a thing in the dark, when you have a candle standing by you.”

— Arbuthnot

Part I

MSL by example

"Mathematicians are like Frenchmen: whatever you say to them they translate into their own language and forthwith it is entirely different."

— Goethe

Data in XML

```
<bib>
  <book year="1999">
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
  </book>
  <book year="2002">
    <title>XML Query</title>
    <author>Fernandez</author>
    <author>Suciu</author>
  </book>
</bib>
```

Data in MSL

```
bib [
  book [
    @year  [ 1999 ],
    title  [ "Data on the Web" ],
    author [ "Abiteboul" ],
    author [ "Buneman" ],
    author [ "Suciu" ]
  ],
  book [
    @year  [ 2002 ],
    title  [ "XML Query" ],
    author [ "Fernandez" ],
    author [ "Suciu" ]
  ]
]
```

Elements in Schema

```
<element name="bib">
  <complexType>
    <sequence>
      <element name="book"
        minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
<element name="book">
  <complexType>
    <sequence>
      <element name="title" type="xsi:string"/>
      <element name="year" type="xsi:integer"/>
      <element name="author" type="xsi:string"
        minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
```

Elements in MSL

```
component(
  sort = element,
  name = bib,
  content =
    bib [ book* ]
)
component(
  sort = element,
  name = book,
  content =
    book [
      title  [ xsi:string ],
      year   [ xsi:integer ],
      author [ xsi:string ]+
    ]
)
```

Elements and types in Schema

```
<element name="bib" type="bibContent"/>
<complexType name="bibContent">
  <sequence>
    <element name="book"
      minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
<element name="book" type="bookContent"/>
<complexType name="bookContent">
  <sequence>
    <element name="title" type="xsi:string"/>
    <element name="year" type="xsi:integer"/>
    <element name="author" type="xsi:string"
      minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

Elements and types in MSL

```
component(
  sort = element,
  name = bib,
  content =
    bib [ bibContent ]
)
component(
  sort = type,
  name = bibContent,
  content =
    book*
)
```

```
component(
  sort = element
  name = book,
  content =
    book [ bookContent ]
)
component(
  sort = type,
  name = bookContent,
  content =
    title  [ xsi:string ],
    year   [ xsi:integer ],
    author [ xsi:string ]+
)
```

Derivation and abstraction in Schema

```
<complexType name="u" final="extension" abstract="false">
  <restriction base="t">
    <choice>
      <element name="d"/>
      <element name="e"/>
    </choice>
  </restriction>
</complex>
```

Derivation and abstraction in MSL

```
component(
  sort = type,
  name = u,
  base = t,
  derivation = restriction,
  refinement = { restriction },
  abstract = false,
  content =
    d | e
)
```

Part II

Syntax

"I never come across one of Laplace's 'Thus it plainly appears' without feeling sure that I have hours of hard work in front of me."

— Bowditch

Syntax

Data	d	$::=$	`@`a`[`d`]`	attribute
		$\mid$	e`[`d`]`	element
		$\mid$	d_1 `,` d_2	sequence
		$\mid$	`()`	empty sequence
Groups	g	$::=$	`@`a`[`g`]`	attribute
		$\mid$	e`[`g`]`	element
		$\mid$	g_1 `,` g_2	sequence
		$\mid$	g_1 `|` g_2	choice
		$\mid$	g`*`	repetition
		$\mid$	`()`	empty
		$\mid$	$\emptyset$	none

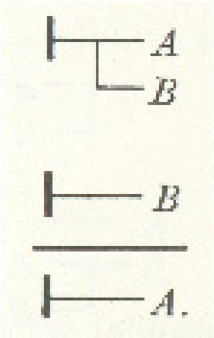
Abbreviations

g`?`	$=$	g `|` `()`	optional
g`+`	$=$	g `,` g`*`	one or more
g`{`m`+1,`n`+1}`	$=$	g `,` g`{`m`,`n`}`	counting
g`{0,`n`+1}`	$=$	`(`g `,` g`{0,`n`})?`	
g`{0,0}`	$=$	`()`	
g`{`m`+1,`∞`}`	$=$	g `,` g`{`m`,*}`	
g`{0,`∞`}`	$=$	g`*`	

Part III

Inference rules

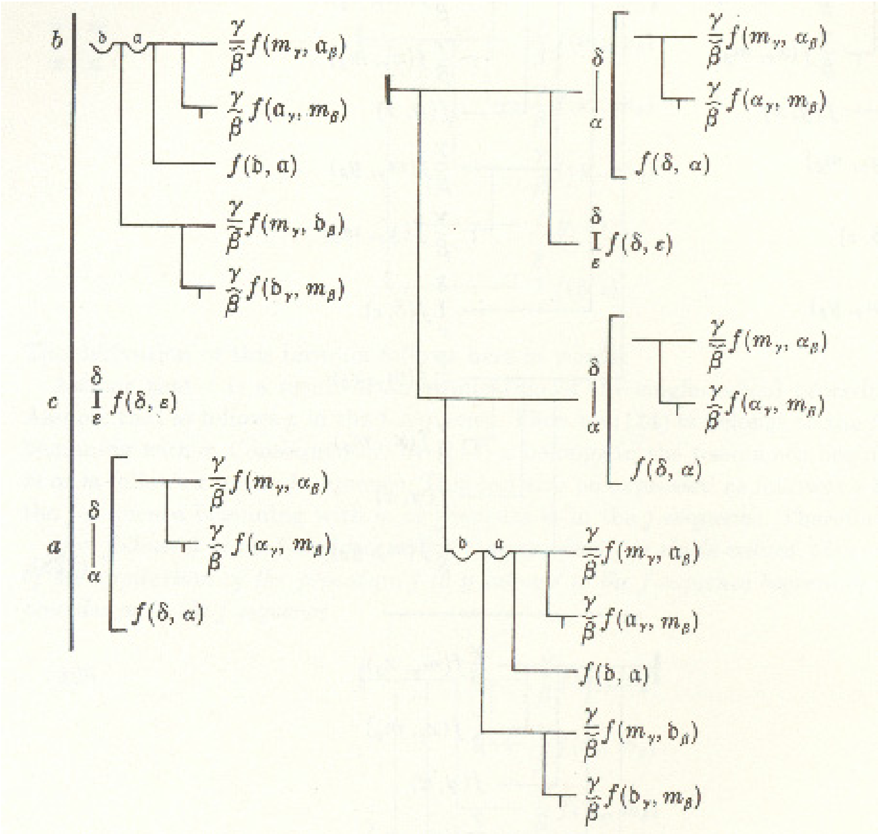
Modus ponens Frege, 1879



Gentzen, 1934

$$\frac{\vdash B \rightarrow A \qquad \vdash B}{\vdash A} \quad (\rightarrow\text{-I})$$

Frege's *Begriffschrift*, 1879



Inference rules

$$\frac{d \in g}{e\texttt{[}d\texttt{]} \in e\texttt{[}g\texttt{]}} \qquad \text{(element)}$$

$$\frac{d_1 \in g_1 \qquad d_2 \in g_2}{d_1 \texttt{,} d_2 \in g_1 \texttt{,} g_2} \qquad \text{(sequence)}$$

$$\frac{}{\texttt{()} \in \texttt{()}} \qquad \text{(empty)}$$

$$\frac{d \in g_1}{d \in g_1 \texttt{|} g_2} \qquad \text{(choice 1)}$$

$$\frac{d \in g_2}{d \in g_1 \texttt{|} g_2} \qquad \text{(choice 2)}$$

$$\frac{d_1 \in g \qquad d_2 \in g\texttt{*}}{d_1 \texttt{,} d_2 \in g\texttt{*}} \qquad \text{(repeat 1)}$$

$$\frac{}{\texttt{()} \in g\texttt{*}} \qquad \text{(repeat 2)}$$

How typing works: element and sequence

$$\dfrac{\dfrac{\dfrac{\texttt{"Data on the Web"} \in \texttt{String}}{\texttt{title["Data on the Web"]} \in \texttt{title[String]}}\;\textsf{elt} \qquad \dfrac{\texttt{1999} \in \texttt{Integer}}{\texttt{year[1999]} \in \texttt{year[Integer]}}\;\textsf{elt}}{\texttt{title["Data on the Web"],year[1999]} \in \texttt{title[String],year[Integer]}}\;\textsf{seq}}{\texttt{book[title["Data on the Web"],year[1999]]} \in \texttt{book[title[String],year[Integer]]}}\;\textsf{seq}$$

How typing works: repetition

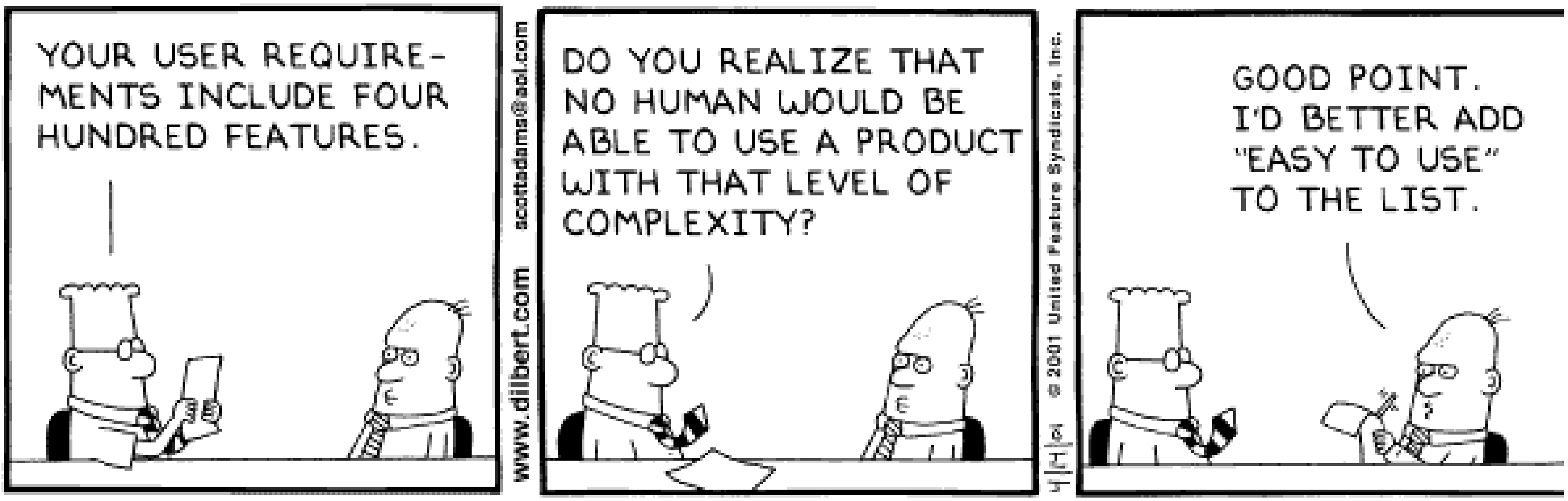
$$\frac{\dfrac{\texttt{"A"} \in \texttt{String}}{\texttt{auth["A"]} \in \texttt{auth[String]}}\,\text{elt} \qquad \dfrac{\dfrac{\texttt{"B"} \in \texttt{String}}{\texttt{auth["B"]} \in \texttt{auth[String]}}\,\text{elt} \quad \dfrac{}{\texttt{()} \in \texttt{auth[String]*}}\,\text{rep2}}{\texttt{(auth["B"],())} \in \texttt{auth[String]*}}\,\text{rep1}}{\texttt{auth["A"],(auth["B"],())} \in \texttt{auth[String]*}}\,\text{rep1}$$

$$\begin{aligned} & \texttt{auth["A"],(auth["B"],())} \\ = \; & \texttt{(auth["A"],auth["B"]),()} \\ = \; & \texttt{auth["A"],auth["B"]} \end{aligned}$$

Part IV

Derivation by restriction

Dilbert



YOUR USER REQUIREMENTS INCLUDE FOUR HUNDRED FEATURES.
DO YOU REALIZE THAT NO HUMAN WOULD BE ABLE TO USE A PRODUCT WITH THAT LEVEL OF COMPLEXITY?
GOOD POINT. I'D BETTER ADD "EASY TO USE" TO THE LIST.
scottadams@aol.com
www.dilbert.com
© 2001 United Feature Syndicate, Inc.

"Besides it is an error to believe that rigor in the proof is the enemy of simplicity. On the contrary we find it confirmed by numerous examples that the rigorous method is at the same time the simpler and the more easily comprehended. The very effort for rigor forces us to find out simpler methods of proof."

— Hilbert

Derivation by restriction

We write $g <:_{\texttt{res}} g'$ if the instances of group g are a subset of the instance of group g'. That is, $g <:_{\texttt{res}} g'$ if for every document d such that $d \in g$ it is also the case that $d \in g'$.

Derivation by restriction

We write $g <:_{\texttt{res}} g'$ if the instances of group g are a subset of the instance of group g'. That is, $g <:_{\texttt{res}} g'$ if for every document d such that $d \in g$ it is also the case that $d \in g'$.

$$\frac{\forall d.\, d \in g \Rightarrow d \in g'}{g <:_{\texttt{res}} g'} \qquad \text{(restriction)}$$

Part V

Conclusions

What's in MSL

- Model groups and validity.
- Derivation by extension and restriction.
- Interleaving (all groups).
- Attributes.
- Normalized names.

What's not in MSL

- Identity constraints.
- The mapping from XML Schema syntax into components.
- Skip and lax wildcard validation.
- The unambiguity restriction on content models.
- The sibling element constraint.
- The `xsi:nil` attribute.
- A check that abstract components are not instantiated.
- Support for form and form default.
- Support for final, block, use, and value.
- The Post Schema Validation Infoset.
- Atomic datatypes.

"Much intellectual mediocrity can be and actually is concealed by some technique sufficiently recondite to discourage outside criticism."

— George Sarton

“Never express yourself more clearly than you are able to think.”

— Niels Bohr