Types in the XML Algebra

Philip Wadler, Avaya

Part I

Types by example

Data in XML

```
<bib>
  <book>
    <title>Data on the Web</title>
    <year>1999</year>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
  </book>
  <book>
    <title>XML Query</title>
    <year>2001</year>
    <author>Fernandez</author>
    <author>Suciu</author>
  </book>
</bib>
```

Data in XML and in Query

```
<bib>
  <book>
    <title>Data on the Web</title>
    <year>1999</year>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
  </book>
  <book>
    <title>XML Query</title>
    <year>2001</year>
    <author>Fernandez</author>
    <author>Suciu</author>
  </book>
</bib>
```

```
bib [
  book [
    title  [ "Data on the Web" ],
    year   [ 1999 ],
    author [ "Abiteboul" ],
    author [ "Buneman" ],
    author [ "Suciu" ]
  ],
  book [
    title  [ "XML Query" ],
    year   [ 2001 ],
    author [ "Fernandez" ],
    author [ "Suciu" ]
  ]
]
```

Types 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>
```

Types in Schema and in Query

```
<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>
```

```
type Bib =
  bib [ Book* ]
type Book =
  book [
    title  [ String ],
    year   [ Integer ],
    author [ String ]+
  ]
```

Content 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>
```

Content types in Schema and in Query

```
<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>
```

```
type Bib =
  bib [ BibContent ]
type BibContent =
  Book*
type Book =
  book [ BookContent ]
type BookContent =
  title  [ String ],
  year   [ Integer ],
  author [ String ]+
```

More element types in Schema

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

More element types in Schema and in Query

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

```
type Book =
  book[
   Title,
   Year,
   Author+
  ]
type Title =
  title[ String ]
type Year =
  year[ Integer ]
type Author =
  author[ String ]
```

Part II

A little type theory

Syntax

Data	d	::=	$a\,[\,d\,]$	element
		|	$d_1\ ,\ d_2$	sequence
		|	$()$	empty sequence
Types	t	::=	$a\,[\,t\,]$	element
		|	$t_1\ ,\ t_2$	sequence
		|	$t_1\ \|\ t_2$	choice
		|	$t*$	repetition
		|	$()$	empty
		|	$\emptyset$	none

Abbreviations

t?	=	t | ()	optional
t+	=	t , t*	one or more
t{m+1,n+1}	=	t , t{m,n}	counting
t{0,n+1}	=	(t , t{0,n})?	
t{0,0}	=	()	
t{m+1,*}	=	t , t{m,*}	
t{0,*}	=	t*	

Type rules

$$\frac{d : t}{a[d] : a[t]} \quad \text{(element)}$$

$$\frac{d_1 : t_1 \quad d_2 : t_2}{d_1 \, , \, d_2 : t_1 \, , \, t_2} \quad \text{(sequence)}$$

$$\frac{}{() : ()} \quad \text{(empty)}$$

$$\frac{d_1 : t_1}{d_1 : t_1 \mid t_2} \quad \text{(choice 1)}$$

$$\frac{d_1 : t_1}{d_1 : t_1 \mid t_2} \quad \text{(choice 2)}$$

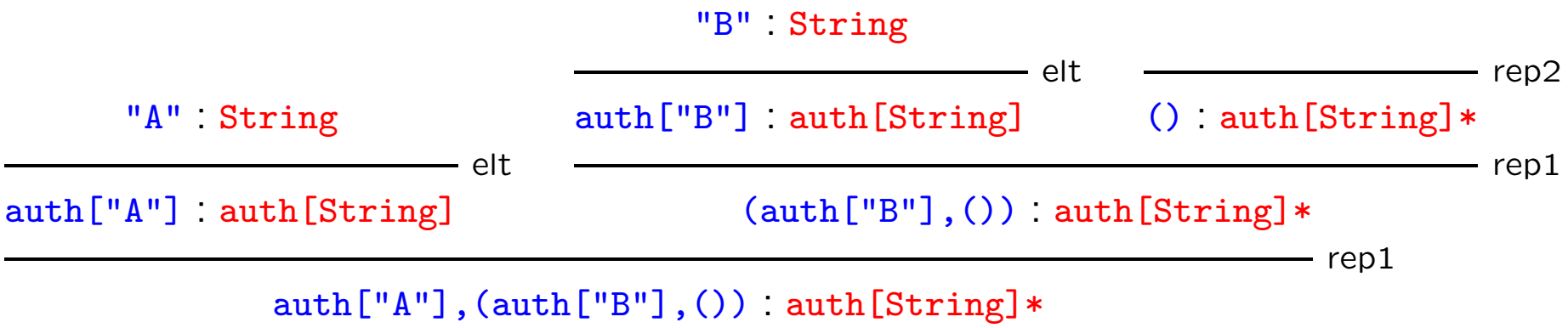
$$\frac{d_1 : t \quad d_2 : t*}{d_1 \, , \, d_2 : t*} \quad \text{(repeat 1)}$$

$$\frac{}{() : t*} \quad \text{(repeat 2)}$$

How typing works: element and sequence

$$\dfrac{\dfrac{\dfrac{\texttt{"Data on the Web"} : \texttt{String}}{\texttt{title["Data on the Web"]} : \texttt{title[String]}}\,\text{elt} \quad \dfrac{\texttt{1999} : \texttt{Integer}}{\texttt{year[1999]} : \texttt{year[Integer]}}\,\text{elt}}{\texttt{title["Data on the Web"],year[1999]} : \texttt{title[String],year[Integer]}}\,\text{seq}}{\texttt{book[title["Data on the Web"],year[1999]]} : \texttt{book[title[String],year[Integer]]}}\,\text{seq}$$

How typing works: repetition




```
  auth["A"],(auth["B"],())
= (auth["A"],auth["B"]),()
= auth["A"],auth["B"]
```

Part III

Better types for ‘for’

Basic type rules

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash a[e] : a[t]} \quad \text{(element)}$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma \vdash e_2 : t_2}{\Gamma \vdash e_1 \texttt{ , } e_2 : t_1 \texttt{ , } t_2} \quad \text{(sequence)}$$

$$\frac{}{\Gamma \vdash \texttt{()} : \texttt{()}} \quad \text{(empty)}$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma, v : t_1 \vdash e_2 : t_2}{\Gamma \vdash \texttt{let } v \texttt{ = } e_1 \texttt{ in } e_2 : t_2} \quad \text{(let)}$$

Old type rules

$$\frac{\Gamma \vdash e : t \qquad p\{m,n\} = \texttt{factor}\ a\ t}{\Gamma \vdash \texttt{project}\ a\ e : p\{m,n\}} \quad \text{(projection)}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : t_1 \\ p_1\{m_1,n_1\} = \texttt{factor}\ t_1 \\ \Gamma, v : p \vdash e_2 : t_2 \\ p_2\{m_2,n_2\} = \texttt{factor}\ t_2\end{array}}{\Gamma \vdash \texttt{for}\ v\ \texttt{<-}\ e_1\ \texttt{in}\ e_2 : p_2\{m_1 \cdot m_2, n_1 \cdot n_2\}} \quad \text{(iteration)}$$

Factor

$$
\begin{array}{lcl}
\texttt{factor}\ a\ (t_1, t_2) & = & (p_1 \mid p_2)\{m_1 + m_2, n_1 + n_2\} \\
 & & \quad \text{where} \quad p_1\{m_1, n_1\} = \texttt{factor}\ a\ t_1 \\
 & & \qquad\qquad\ \ p_2\{m_2, n_2\} = \texttt{factor}\ a\ t_2 \\
\texttt{factor}\ a\ (t_1 \mid t_2) & = & (p_1 \mid p_2)\{\mathsf{min}\ \ m_1\ \ m_2, \mathsf{max}\ \ n_1\ \ n_2\} \\
 & & \quad \text{where} \quad p_1\{m_1, n_1\} = \texttt{factor}\ a\ t_1 \\
 & & \qquad\qquad\ \ p_2\{m_2, n_2\} = \texttt{factor}\ a\ t_2 \\
\texttt{factor}\ a\ (t\{m, n, \}) & = & p_1\{m_1 \cdot m, n_1 \cdot n\} \\
 & & \quad \text{where} \quad p_1\{m_1, n_1\} = \texttt{factor}\ a\ t_1 \\
\texttt{factor}\ a\ (a[x]) & = & a[x]\{1, 1\} \\
\texttt{factor}\ a\ (b[x]) & = & \emptyset\{0, 0\}, \quad \text{if } a \neq b \\
\texttt{factor}\ a\ (\tilde{}[x]) & = & a[x]\{0, 1\} \\
\texttt{factor}\ a\ [] & = & \emptyset\{0, 0\} \\
\texttt{factor}\ a\ \emptyset & = & \emptyset\{*, 0\}
\end{array}
$$

Factor

$$
\begin{array}{lcl}
\texttt{factor}\ (t_1, t_2) & = & (p_1 \mid p_2)\{m_1 + m_2, n_1 + n_2\} \\
 & & \quad \text{where} \quad p_1\{m_1, n_1\} = \texttt{factor}\ t_1 \\
 & & \qquad\qquad\ \ p_2\{m_2, n_2\} = \texttt{factor}\ t_2 \\
\texttt{factor}\ (t_1 \mid t_2) & = & (p_1 \mid p_2)\{\min\ m_1\ m_2, \max\ n_1\ n_2\} \\
 & & \quad \text{where} \quad p_1\{m_1, n_1\} = \texttt{factor}\ t_1 \\
 & & \qquad\qquad\ \ p_2\{m_2, n_2\} = \texttt{factor}\ t_2 \\
\texttt{factor}\ (t\{m, n, \}) & = & p_1\{m_1 \cdot m, n_1 \cdot n\} \\
 & & \quad \text{where} \quad p_1\{m_1, n_1\} = \texttt{factor}\ t_1 \\
\texttt{factor}\ (a[x]) & = & a[x]\{1, 1\} \\
\texttt{factor}\ (\tilde{}[x]) & = & \_[x]\{1, 1\} \\
\texttt{factor}\ [] & = & \emptyset\{0, 0\} \\
\texttt{factor}\ \emptyset & = & \emptyset\{*, 0\}
\end{array}
$$

New type rules

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma \; ; \; \texttt{for}\, v : t_1 \vdash e_2 : t_2}{\Gamma \vdash \texttt{for}\ v\ \texttt{<-}\ e_1\ \texttt{in}\ e_2 : t_2} \qquad \text{(for)}$$

$$\frac{\Gamma, v : a[t] \vdash e : t'}{\Gamma \; ; \; \texttt{for}\, v : a[t] \vdash e : t'} \qquad \text{(for element)}$$

$$\frac{\Gamma \; ; \; \texttt{for}\, v : t_1 \vdash e : t_1' \quad \Gamma \; ; \; \texttt{for}\, v : t_2 \vdash e : t_2'}{\Gamma \; ; \; \texttt{for}\, v : t_1 \texttt{ , } t_2 \vdash e : t_1' \texttt{ , } t_2'} \qquad \text{(for sequence)}$$

$$\frac{\Gamma \; ; \; \texttt{for}\, v : t_1 \vdash e : t_1' \quad \Gamma \; ; \; \texttt{for}\, v : t_2 \vdash e : t_2'}{\Gamma \; ; \; \texttt{for}\, v : t_1 \texttt{ | } t_2 \vdash e : t_1' \texttt{ | } t_2'} \qquad \text{(for choice)}$$

$$\frac{\Gamma \; ; \; \texttt{for}\, v : t \vdash e : t'}{\Gamma \; ; \; \texttt{for}\, v : t\texttt{*} \vdash e : t'\texttt{*}} \qquad \text{(for repetition)}$$

$$\frac{}{\Gamma \; ; \; \texttt{for}\, v : \texttt{()} \vdash e : \texttt{()}} \qquad \text{(for empty)}$$

$$\frac{}{\Gamma \; ; \; \texttt{for}\, v : \emptyset \vdash e : \emptyset} \qquad \text{(for none)}$$

Consistent element restriction

The *consistent element restriction* of Schema requires that all elements in a regular expression must have the same content. For example,

$a[t]\ ,\ a[t]$ — legal
$a[t]\ ,\ a[u]$ — illegal

One equivalence between types is

$$a[t \mid u] = a[t] \mid a[u]$$

The new (for) rule assigns different types to the left and right sides. (Horror!)

But the consistent element restriction makes the right hand side illegal. (Whew!)

Projection simplified!

Previously, we translated

```
book0/author
```

as

```
project author (children(book0))
```

but now we translate it as

```
for x <- children(book0) in
  case x of
    y : author => y
  | z => ()
```