XML notions formalized

Maarten Fokkinga

Version of April 23, 2003, 14:09

Abstract. We provide a formal definition of several XML notions: schema, document, validity (of a document for a schema), query and answer, and integration. The definitions have the following properties:

- precise and intuitive at the same time,
- using common, standard mathematical notions wherever possible,
- at a high level of abstraction (= many aspects are not dealt with),
- suitable for use in proving properties.

(Admittedly, being *intuitive* is a rather subjective property, and the *suitability* for proving properties has to be established yet.) This work arose in attempt to understand some definitions in an early working version of Marko's paper "XML schema integration — the Xblock approach".

* * *

Remark (Joeri): "it is important —for some reason, but he forgot which— that element and attribute names are on the edges rather than the nodes."

Joeri: is respection similar to continuity ("respecting open sets")?

Joeri: can cardinality respection be eliminated in exchange for extra choice and auxiliary nodes in the schema? For example, $\mathit{elt}\,a \overset{1..3}{\longmapsfrom} \mathit{elt}\,b$ is represented as:

$$
\begin{array}{lllll}
\mathit{elt}\,a & \longmapsfrom \mathit{choice} & \longmapsfrom \mathit{aux} \longmapsfrom \mathit{elt}\,b & & \text{once } \mathit{elt}\,b \\
 & \longmapsfrom \mathit{choice} & \longmapsfrom \mathit{aux} \longmapsfrom \mathit{elt}\,b & & \text{twice } \mathit{elt}\,b \\
 & & \longmapsfrom \mathit{aux} \longmapsfrom \mathit{elt}\,b & & \\
 & \longmapsfrom \mathit{choice} & \longmapsfrom \mathit{aux} \longmapsfrom \mathit{elt}\,b & & \text{thrice } \mathit{elt}\,b \\
 & & \longmapsfrom \mathit{aux} \longmapsfrom \mathit{elt}\,b & & \\
 & & \longmapsfrom \mathit{aux} \longmapsfrom \mathit{elt}\,b & &
\end{array}
$$

MMF: From http://www.w3.org/TR/xquery/ paragraph 5.3: "It is worth noting ... that document order is defined in such a way that a node is considered to precede its descendants in document order." This is not (yet) done in this paper.

Common mathematical concepts

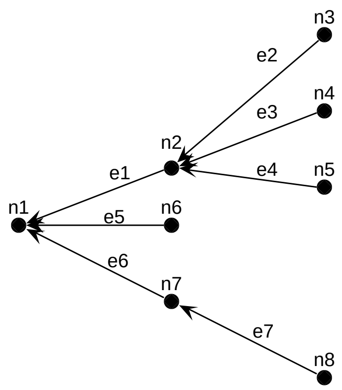
1 Tree and Forest. A *forest* consists of the following components:

- a set *Node* of so-called *nodes*,
- a relation $Edge : Node \times Node$, also called the *parent-child connection*,

such that:

- *Edge* is a partial function from right to left
  (that is, a child has at most one parent: $(m, n), (m', n) \in Edge \Rightarrow m = m'$).
- The transitive closure of *Edge* contains no cycles: $(n, n) \notin Edge^+$
  (that is, a node is not its own predecessor or descendant).

Figure 1: A *forest.*



$$Node = \{n_1, \quad \ldots \quad, n_8\}$$
$$Edges = \{\underbrace{(n_1, n_2)}_{=e_1}, \quad \ldots \quad, \underbrace{(n_1, n_7)}_{=e_6}, \underbrace{(n_7, n_8)}_{=e_7}\}$$

Node n_1 is a *root*. The forest is a *tree* since there is only one root. Node n_1 has three children: $children\, n_1 = \{n_2, n_6, n_7\}$; $children\, n_6 = \varnothing$; and $children\, n_7 = \{n_8\}$ so $theChild\, n_7 = n_8$. Pair (n_1, n_2) is an edge: $(n_1, n_2) \in Edge$; in the text we write this property as: $n_1 \hookleftarrow n_2$.

We let m, n vary over *Node*. If $(n, n') \in Edge$, we call n the *parent* of n' and n' a *child* of n, and we write $n \hookleftarrow n'$ (pointing from a child to its parent, thus reminding of the functional property of *Edge*). Some auxiliary obvious notions:

$children\, n$	$=$	$\{n' : Node \mid n \hookleftarrow n'\}$	the children of node n
$theChild\, n = n'$	$\Leftrightarrow$	$children\, n = \{n'\}$	the child of n, provided $\#(children\, n) = 1$
$root\, n$	$\Leftrightarrow$	$\neg\, \exists\, n' \bullet n' \hookleftarrow n$	node n is a root if it has no parent
$tree\, F$	$\Leftrightarrow$	$\exists_1\, n : Node_F \bullet root\, n$	forest F is a tree if it has exactly one root
$path\, n$	$=$	"the sequence of edges from the root (above n) up to n"	

Figure 1 gives a graphical example of a tree.

A *node labeled* forest is a forest that has these additional components:

- a set *NLabel* of *node labels*,
- a function $lab : Node \rightarrow NLabel$.

An *edge labeled* forest is a forest that has these additional components:

- a set *ELabel* of *edge labels*,

- a function $lab : Edge \rightarrow ELabel$.

A *partially ordered* forest is a forest that has this additional component:

- a partial order $(\preceq) : Node \times Node$

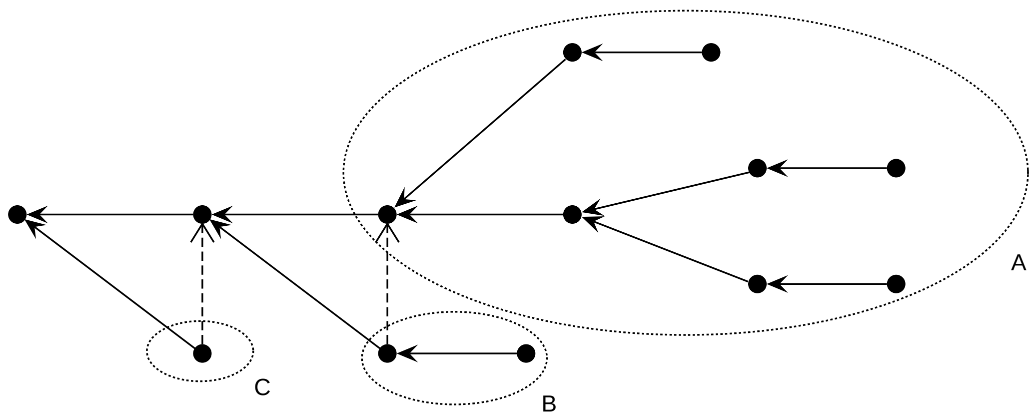
that is consistent with the tree structure:

- $m \overset{*}{\mapsfrom} n \wedge m' \overset{*}{\mapsfrom} n' \wedge \quad m \preceq m' \quad \Rightarrow \quad n \preceq n'.$ [consistency]

In pictures, the ordering between n and n' will often not be drawn if $n \preceq n'$ holds on account of the *consistency* rule. Figure 2 gives a graphical example of a partially ordered tree. A *finitely branching* forest is a forest with this property:

- $children\, n$ is finite (for each node n).

Figure 2: A partially ordered tree.



A dashed arrow denotes an ordered pair of nodes, the arrow head pointing to the smaller side. Due to the consistency requirement some more nodes are ordered: for all nodes n in area A, all n' in B, and all n'' in C we also have $n \preceq n' \preceq n''$. The drawing would become unreadable if all these were drawn. In this example, the nodes within area A are not ordered; the same holds for area B.

When discussing several forests or trees $F, T \ldots$ at the same time, we use subscripts for disambiguation, thus writing, say, $Node_F$, $Edge_T$, $m \mapsfrom_F m'$, $m \preceq_F n'$, and $root_T$.

2 Mappings between forests. Let F, G be two forests.

A function $f : Node_F \rightarrow Node_G$ is called *xyz-preserving* if, informally, properties of kind *xyz* in the source do hold for the image under f in the target; formally:

$$
\begin{array}{llll}
m \mapsfrom_F m' & \Rightarrow & f\, m \mapsfrom_G f\, m' & \text{edge preservation} \\
m \overset{*}{\mapsfrom}_F m' & \Rightarrow & f\, m \overset{*}{\mapsfrom}_G f\, m' & \text{path preservation} \\
lab_F\, m & = & lab_G\, (f\, m) & \text{node label preservation} \\
lab_F\, (m, m') & = & lab_G\, (f\, m, f\, m'), \quad \text{if } m \mapsfrom_F m' & \text{edge label preservation}
\end{array}
$$

$$\begin{array}{lcll} root_F\, m & \Rightarrow & root_G(f\ m) & \text{root preservation} \\ m \preceq_F m' & \Rightarrow & f\,m \preceq_G f\,m' & \text{order preservation} \end{array}$$

Edge label preservation only makes sense if the function preserves edges; we do not need this notion in the sequel. (Preservation is more or less the *same* as the algebaic notion of homomorphism.)

A function $f : Node_F \rightarrow Node_G$ might be called *xyz-respecting* if, informally, constraints or predicates of kind *xyz* expressed in the target do hold for the origin under f in the source. However, taken literally this notion is not very useful, since one node in G may have several origins under f. Therefore, we look at the sets $children_F\, m$ separately (where m is an origin under f of n). Moreover, we will use respection in a specific way, dedicated to XML particularities. Namely, we assume that '*seq*' and '*choice*' are specific node labels, and that cardinalities (= subsets of $\mathbb{N}$) are the edge labels (thus writing '*card*' instead of the edge labeling '*lab*'). This leads to the following definition:

$$\begin{array}{lcll} m', m'' \in children_F\, m & \Rightarrow & f\,m' \preceq_G f\,m'' \Rightarrow m' \preceq_F m'' & \text{order respection} \\ f\,m = n \wedge lab_G\, n = choice & \Rightarrow & \#\{m' : children_F\, m \bullet f\,m'\} = 1 & \text{choice respection} \\ f\,m = n \wedge n \hookleftarrow_G n' & \Rightarrow & \#\{m' : children_F\, m \mid f\,m' = n'\} \in card_G(n, n') & \\ & & & \text{cardinality respection} \end{array}$$

In order that cardinality respection and choice respection are well-defined, the forest needs to be finitely branching (for otherwise the '$\#\ldots$' expressions make no sense).

3 Retraction, congruence. Let F and G be forests, possibly node labeled, edge labeled and partially ordered. Forest F is a *retraction* of G, denoted $F \cong G$, if: there exists a *injective* function $f : Node_F \rightarrow Node_G$ that preserves edges and, when applicable, also node labels, edge labels and the partial order. We also say that F is a restract of G *via* f, and call f the *retract.*

Forest F is *congruent* to G, denoted $F \cong G$, if: F and G are retractions of each other via retracts that are each others inverse.

4 Subforest, subtree. Let F be a forest, and n a member of $Node_F$. The *subforest* at n in F, denoted $sub_F\, n$, is the forest G defined by:

$$\begin{array}{lcll} Node_G & = & \{n' : Node_F \mid n \overset{*}{\hookleftarrow} n'\} & \\ n' \hookleftarrow_G n'' & \Leftrightarrow & n' \hookleftarrow_F n'' & \text{(for all } n', n'' \in Node_G) \end{array}$$

When F is node labeled, then lab_G equals: lab_F restricted to the smaller set $Node_G$. Similarly when F is edge labeled or partially ordered. Note that each node of G *is* a node of F; indeed, $sub_F\, n$ is a retraction of F with the identity function as retract.

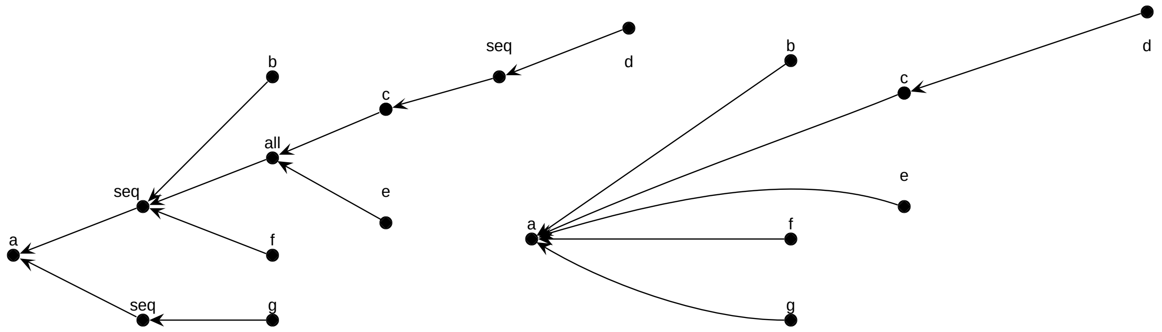
5 Edge contraction. Let F be a node labeled forest, and A be a subset of *NLabel*. We will construct a new forest out of F by contracting $m_0 \hookleftarrow m' \hookleftarrow \ldots \hookleftarrow m'' \hookleftarrow m_1$ to just $m_0 \hookleftarrow m_1$ in case the labels of the intermediate nodes $m', \ldots, m''$ are all in the designated subset A. (This operation is needed when comparing an XML schema to an XML document: in the former there may occur nodes with node labels *seq*, *choice*, and *all* but in the latter not.) Formally, the A-contraction of F is a forest G defined by:

$$
\begin{array}{lcl}
Node_G & = & Node_F \setminus \{n : Node_F \mid lab\, n \in A\} \\
n_0 \mathrel{\leftarrow\!\!\dashv}_G n_1 & \equiv & \exists\, n', \ldots, n'' \bullet n_0 \mathrel{\leftarrow\!\!\dashv}_F n' \mathrel{\leftarrow\!\!\dashv}_F \ldots \mathrel{\leftarrow\!\!\dashv}_F n'' \mathrel{\leftarrow\!\!\dashv}_F n_1 \wedge \{lab\, n', \ldots, lab\, n''\} \subseteq A
\end{array}
$$

$$\text{(for } n_0, n_1 \in Node_G\text{)}$$

When F is node labeled, then lab_G equals: lab_F restricted to the smaller set $Node_G$. Similarly when F is partially ordered. See Figure 3 for an illustration.

Figure 3: A *contraction*.



The right tree is the $\{seq, all, choice\}$-contraction of the left tree.

XML schemas and documents

Our notions of an XML schema and an XML document are a slight abstraction of some of the concepts defined by the W3C Consortium in May 2000: we do not elaborate the definitions of the *name* of an *element* and of the *value* of a *datatype*. Moreover, we abstract from a textual representation; our XML schemas and documents are *forests* and *trees*, respectively, of which a pretty-print comes quite close to the 'official' concepts.

TODO (suggestions by Marko). An attribute may have a list of children, (considered as one child?). Mixed content is dealt with wrongly. Some constraints deviate from the official definitions. In spite of these deviations, the approach below does show the suitability in this application area. And that's the only thing what I want to show, for the time being.

6 Preliminaries. We postulate the existence of several auxiliary sets:

Value	:	a set of values	the universe of all values
Datatype	=	$\mathbb{P}$ *Value*	each datatype is a subset of *Value*
Name	:	a set of names	for *elements* and *attributes*
Doctype	=	$\{$*element*, *attribute*, *content*$\}$	document node types
Comptype	=	$\{$*seq*, *all*, *choice*$\}$	compositor types

$$Type \quad = \quad Doctype \cup Comptype \qquad \text{node types}$$

An XML schema and an XML document will be defined below as node labeled forests with the following set of node labels:

$$\begin{aligned}
&schemaNLabel = \\
&\quad \{element\} \times Name \ \cup \ \{attribute\} \times Name \ \cup \ \{content\} \times Datatype \ \cup \ Comptype \\
&docNLabel = \\
&\quad \{element\} \times Name \ \cup \ \{attribute\} \times Name \ \cup \ \{content\} \times Value
\end{aligned}$$

Based on the node label function *lab* we define some auxiliary (overloaded) functions:

$$\begin{array}{lcll}
type\, n & = & element, & \text{if } lab\, n = (element, ...) \\
 & = & attribute, & \text{if } lab\, n = (attribute, ...) \\
 & = & content, & \text{if } lab\, n = (content, ...) \\
 & = & lab\, n, & \text{if } lab\, n \in Comptype \\
name\, n & = & x, & \text{if } lab\, n = (element, x) \\
 & = & x, & \text{if } lab\, n = (attribute, x) \\
value\, n & = & x, & \text{if } lab\, n = (content, x)
\end{array}$$

Finally, we define a weakened form of node label preservation (see §2 page 3). Given node labeled forests F and G, a function $f : Node_F \to Node_G$ is called *weakly node label preserving* if:

$$lab_F\, m \quad \approx \quad lab_G\, (f\, m) \qquad \text{weak node label preservation}$$

where $lhs \approx rhs$ holds true if:

$$\begin{array}{lcl}
(true & \Rightarrow & type\, lhs = type\, rhs) \wedge \\
(\{type\, lhs, type\, rhs\} \subseteq \{element, attribute\} & \Rightarrow & name\, lhs = name\, rhs) \wedge \\
(\{type\, lhs, type\, rhs\} \subseteq \{content\} & \Rightarrow & value\, lhs \in value\, rhs)
\end{array}$$

7 XML schema. (See Figure 4 for an example.) An *XML schema*, or just *schema*, is a forest with the following extras:
the forest is finitely branching; it is partially ordered (the order is intended to give the place of the content within an element in the pretty-print of the forest, and also the order between items in a sequence); the nodes are labeled with document types (*element*, *attribute*, *content*) or *compositor types* (*seq*, *choice*, *all*) together with their names and values, if applicable:

$$NLabel \quad = \quad schemaNLabel \qquad (schemaNLabel \text{ is defined in §6 page 6})$$

and the edges are labeled with so-called cardinalities (non-empty subsets of $\mathbb{N}$, conventionally denoted by $0\,..\,1$, or $1..*$ and the like):

$$ELabel \quad = \quad \mathbb{P}_1\, \mathbb{N}$$

For readability we shall write the edge label function as '*card*'. Moreover, the forest has the following series of properties. [*Consider these as examples; the properties for the* real *XML schemas might differ somewhat.*] First, with respect to the node labeling:

- A *content* has no children:

$$\mathit{type}\, n = \mathit{content} \quad \Rightarrow \quad \#(\mathit{children}\, n) = 0$$

- An *attribute* has exactly one child, being a *content*:

$$\mathit{type}\, n = \mathit{attribute} \quad \Rightarrow \quad \#(\mathit{children}\, n) = 1 \wedge \mathit{type}\,(\mathit{theChild}\, n) = \mathit{content}$$

- Only *element* and *attribute* occur multiple times in an *element*:

$$\begin{array}{l} \mathit{type}\, n = \mathit{element} \Rightarrow \\ \forall\, t : \mathit{Type} \setminus \{\mathit{element}, \mathit{attribute}\} \bullet \#\{n' : \mathit{children}\, n \mid \mathit{type}\, n' = t\} \leq 1 \end{array}$$

- Both *seq* and *choice* only have *element*, *seq*, *choice*, whereas *all* only has *element*s:

$$\begin{array}{lll} \mathit{type}\, n \in \{\mathit{seq}, \mathit{choice}\} & \Rightarrow & \#\{n' : \mathit{children}\, n \mid \mathit{type}\, n' \notin \{\mathit{element}, \mathit{seq}, \mathit{choice}\}\} = 0 \\ \mathit{type}\, n = \mathit{all} & \Rightarrow & \#\{n' : \mathit{children}\, n \mid \mathit{type}\, n' \neq \mathit{element}\} = 0 \end{array}$$

- Each root is an element:

$$\mathit{root}\, n \quad \Rightarrow \quad \mathit{type}\, n = \mathit{element}$$

Second, with respect to the edge labeling *card*:

- For the *content* of an attribute, the cardinality is "required";

$$\mathit{type}\, n = \mathit{attribute} \wedge n \leftarrow\!\!\dashv n' \wedge \mathit{type}\, n' = \mathit{content} \quad \Rightarrow \quad \mathit{card}(n, n') = \{1\}$$

- For the *content* of an *element*, and each *attribute* and *all*, the cardinality is "optional":

$$\begin{array}{lll} \mathit{type}\, n = \mathit{element} \wedge n \leftarrow\!\!\dashv n' \wedge \mathit{type}\, n' = \mathit{content} & \Rightarrow & \mathit{card}(n, n') = \{0, 1\} \\ n \leftarrow\!\!\dashv n' \wedge \mathit{type}\, n' \in \{\mathit{attribute}, \mathit{all}\} & \Rightarrow & \mathit{card}(n, n') = \{0, 1\} \end{array}$$

- [TODO: true??] For an *element* and *seq*, the min cardinality is at least 1:

$$n \leftarrow\!\!\dashv n' \wedge \mathit{type}\, n' \in \{\mathit{element}, \mathit{seq}\} \quad \Rightarrow \quad 0 \notin \mathit{card}(n, n')$$

Third, with respect to the partial order:

- The partial order is total on the children of a *seq*:

$$\mathit{type}\, n = \mathit{seq} \quad \Rightarrow \quad \forall\, n', n'' : \mathit{children}\, n \bullet n' \preceq n'' \vee n'' \preceq n'$$

Fourth, with respect to consistency in the use of *elements*:

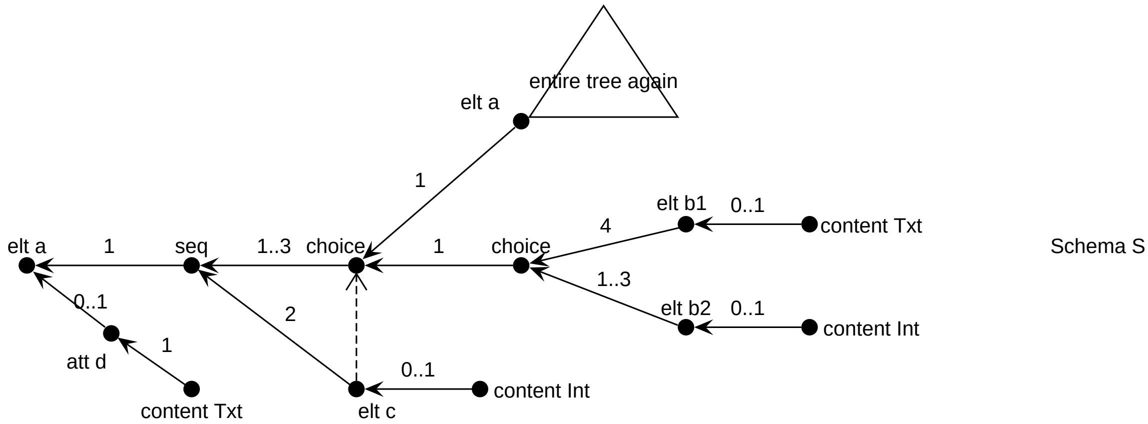
- Equally named *elements* have congruent subtrees:

$$\mathit{type}\, n = \mathit{element} = \mathit{type}\, n' \wedge \mathit{name}\, n = \mathit{name}\, n' \quad \Rightarrow \quad \mathit{sub}\, n \cong \mathit{sub}\, n'$$

Thus, in our notion of *schema*, the presence of recursive *element*s forces the schema to be infinite! (A pretty-print of the schema, however, might still be finite.) See Figure 4 for an example of a schema with a recursive *element*.
Fifth, dealing with unambiguity:

- Paragraph 11, below, gives one more requirement.

Figure 4: A *schema* and a pretty-print of it.



The partial order is, as before, drawn with a dashed arrow. Note that both the root and another node have label '*elt a*'. Therefore the entire tree and the subtree at that other node are congruent to each other, implying that the tree is infinite. The repetition is symbolically drawn by a triangle with text "entire tree again". Here is a pretty-print of the schema:

```
<element name="a">
  <complexType>
   <sequence>
      <choice maxOccurs="3">
         <element ref="a"/>
         <choice>
            <element name="b1" type="integer" minOccurs="4" maxOccurs="4"/>
            <element name="b2" type="integer" minOccurs="1" maxOccurs="3"/>
         </choice>
      </choice>
      <element name="c" type="integer" minOccurs="2" maxOccurs="2"/>
   </sequence>
   <attribute name="d" type="string"/>
  </complexType>
</element>
```

8 XML document. (See Figure 5 for an example.) An *XML document*, or just *document*, is a tree with the following extras:
the tree is finitely branching; it is partially ordered (the order is intended to give the place of the nodes in the pretty-print of the tree); the nodes are labeled with a document type (*element*, *attribute*, *content*) together with a name and a value, if appropriate:

$$NLabel \quad = \quad docNLabel \qquad\qquad (docNLabel \text{ is defined in §6 page 6})$$

Moreover the tree has the following series of properties. First, with respect to the node labeling:

- A *content* has no children:

$$type\, n = content \quad \Rightarrow \quad \#(children\, n) = 0$$

- An *attribute* has exactly one child, being a *content*:

$$type\, n = attribute \quad \Rightarrow \quad \#(children\, n) = 1 \wedge type\,(theChild\, n) = content$$

- An *element* has at most one *content*:

$$type\, n = element \quad \Rightarrow \quad \#\{n' : children\, n \mid type\, n' = content\} \leq 1$$

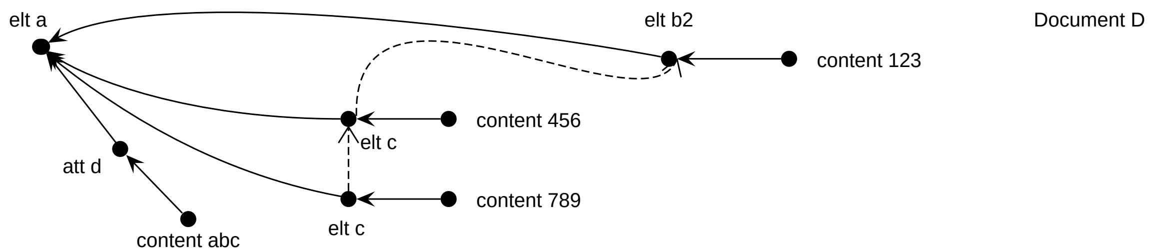
Second, with respect to the partial order:

- [TODO: true?] The partial order is total on the non-attribute children of an element.

$$type\, n = element \Rightarrow$$
$$\forall\, n', n'' : children\, n \mid type\, n' \neq attribute \neq type\, n'' \bullet n' \preceq n'' \vee n'' \preceq n'$$

See Figure 5 for an example document.

Figure 5: A *document* and a pretty-print of it.



Due to the consistency requirement, the partial order contains more pairs than drawn here. Here is a (the only possible) pretty-print of the document:

```
<a d="abc">
   <b2> 123 </b2>
   <c> 456 </c>
   <c> 789 </c>
</a>
```

9 Validity. Let D and S be a document and schema, respectively. Then '*D is valid for S*' means the following:

> There exists some node labeled and partially ordered tree $\hat{D}$ and a mapping M from $Node_{\hat{D}}$ to $Node_S$, such that:

– Document D is the $\{seq, choice, all\}$-contraction of $\hat{D}$.

– Mapping M preserves the edges and root of $\hat{D}$ and weakly preserves the node labels of $\hat{D}$ and respects the order, cardinalities, and choice of S.

We call M the *materialization* mapping. Below, we will elaborate the notion of materialization.

Isn't this a beautiful, elegant, precise-and-intuitive definition? Yes, it is!
See Figure 6 for an illustration.
As service to the reader, we spell out the properties of M, and explain the role of $\hat{D}$:

- The insertion of *seq*, *choice*, and *all* nodes in D to form $\hat{D}$ is free to some degree (thus giving rise to several ways in which a document may be valid for a schema); speaking informally and imprecisely, these nodes give a *trace* of how the schema could have been instantiated to the document.

- The extension of the partial order of D to the new nodes is free but constrained by the already present order on their children (see the consistency requirement for the partial order on a forest).

- M preserves the edges and the root and weakly preserves the node labels of $\hat{D}$:

$$\begin{array}{lcll} m \leftarrow\!\!\dashv_{\hat{D}} m' & \Rightarrow & M\,m \leftarrow\!\!\dashv_S M\,m' & \\ root_{\hat{D}}\,m & \Rightarrow & root_S\,(M\,m) & \\ type_{\hat{D}}\,m & = & type_S\,(M\,m) & \\ name_{\hat{D}}\,m & = & name_S\,(M\,m) & \text{where appropriate} \\ value_{\hat{D}}\,m & \in & values_S\,(M\,m) & \text{where appropriate} \end{array}$$

- M respects the order, choice, and cardinality of S:

$$\begin{array}{lcl} m', m'' \in children_{\hat{D}}\,m & \Rightarrow & M\,m' \preceq_S M\,m'' \Rightarrow m' \preceq_{\hat{D}} m'' \\ M\,m = n \wedge type_S\,n = choice & \Rightarrow & \#\{m' : children_{\hat{D}}\,m \bullet M\,m'\} = 1 \\ M\,m = n \wedge n \leftarrow\!\!\dashv_S n' & \Rightarrow & \#\{m' : children_{\hat{D}}\,m \mid M\,m' = n'\} \in cards_S(n, n') \end{array}$$

10 Materialization. Let schema S and document D be given, and suppose that D is valid for S via materialization mapping M and intermediate $\hat{D}$. Note that M is a function from D to S, not the other way around.

For a node n in S with a document type and a node m in $\hat{D}$ with $M\,m = n$ (so m has a document type too, and hence is in D as well), we call m a *materialization* of n.

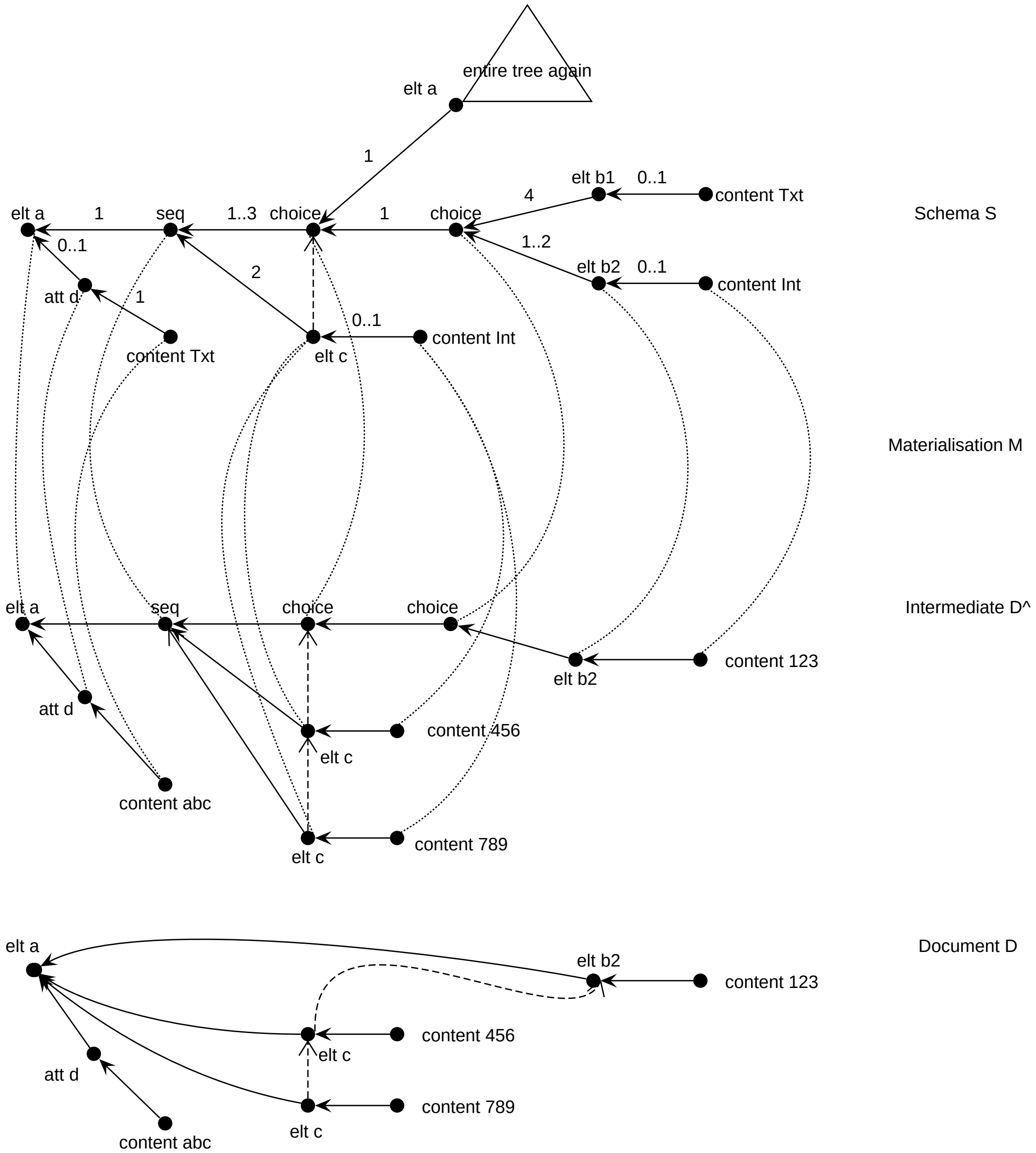
Note. For a node n in S with a compositor type (that is, its type is *seq*, *all*, or *choice*), it is hard to speak of materialization of n. We choose the following definition [but do not know yet whether this notion is of any use; so, **dear reader, please skip the remainder of this note**].

Let m_0 be a node in $\hat{D}$ with $M\,m_0 = n$. Then, the entire set

$$\{m : Node_D \quad | \quad type\,m \in Doctype \wedge (\exists\, m_1, \ldots, m_k : Node_{\hat{D}} \setminus Node_D \bullet m_0 \leftarrow\!\!\dashv \ldots \leftarrow\!\!\dashv m_k \leftarrow\!\!\dashv m) \qquad \}$$

is called *a materialization of* node n.

Figure 6: Validity of D for S via M and $\hat{D}$.



Graphically corresponding blobs and arrows of D and $\hat{D}$ are different drawings of the *same* nodes and edges. The ordered pairs drawn in $\hat{D}$ imply, by the convention explained in Figure 2, the order drawn in D. For both *choice*-nodes in schema S the lower alternative is instantiated. For the *seq*-node in S, the first branch is instantiated once (consistent with cardinality 1..3) and the second branch is instantiated twice (forced by the cardinality 2). Element $b2$ is instantiated once (consistent with 1..2).

Conjecture (soundness): This set is the same as:

$\{n' : Nodes_S \quad | \quad type\, n' \in Doctype \wedge$

$$(\exists\, n_1, \ldots, n_k : \mathit{Nodes}_S \mid \mathit{type}\,(\!|\ \{n_1, \ldots, n_k\}\ |\!) \cap \mathit{Doctype} = \varnothing \bullet n \hookleftarrow n_1 \hookleftarrow \ldots \hookleftarrow n_k \hookleftarrow n')$$
$$\bullet \quad \mu(m : \mathit{Node}_D \bullet M\, m = n) \qquad \}$$

11 XML schema again. In order that our definition of *schema* better reflects the definition recommended by the W3C Consortium, it is presumably the case that the following requirement has to be added:

- For any document valid for the schema, the intermediate $\hat{D}$ and the materialization M is unique.

It remains to be seen whether this property can be formulated in the style of all the other properties. Whether the W3C Recommendation does so or not, in the sequel we assume this restriction to be part of the definition of *schema*. This is a consequence:

$$\overset{seq}{\bullet} \xleftarrow{\{1,2\}} \overset{choice}{\bullet} \xleftarrow{1..2} \overset{elt\,a}{\bullet} \leftarrow \cdots \qquad \text{K.O.: parsing 2 instances of } \mathit{elt}\,a \text{ is ambiguous}$$

$$\overset{seq}{\bullet} \xleftarrow{\{1,3\}} \overset{choice}{\bullet} \xleftarrow{1..2} \overset{elt\,a}{\bullet} \leftarrow \cdots \qquad \text{O.K.; parsing instances of } \mathit{elt}\,a \text{ is unambiguous}$$

12 Research question. Is there an effective procedure for deciding whether a document is valid for a schema?

What about the pre-order $\sqsubseteq$ defined by:

$$S \sqsubseteq S' \quad \equiv \quad \forall\, D \bullet D \text{ is valid for } S \Rightarrow D \text{ is valid for } S'$$

What can we say about S and S' when $S \sqsubseteq S' \sqsubseteq S$? Is there a minimal and maximal (or least and greatest) element in the induced partial order? Is there an effective procedure for deciding whether $S \sqsubseteq S'$?

Querying

13 Query. Rather than trying to give a *syntax* to express queries, we wish to define the *meaning* of a query.

> In XQuery (and related languages), a query consists of a *pattern* clause (called *path* expression, IF I'M NOT MISTAKEN), a *filter* clause, and a *constructor* clause. We focus on the pattern clause only; that's what we call a query. We take the position that arbitrary language (for instance mathematics) may be used for the filter and constructor clause.

Here is our definition.

Let a schema S be given. A *query over* S is: a pair (n, P) of a node n in Node_S and a predicate $P(\_,\_)$ on pairs of a document and a node in that document. Let furthermore a document D be given, valid for S via materialization M. (In view of the assumption/constraint explained in paragraph 11, function M is uniquely determined.) The *answer* in D to query (n, P) is the set of the materializations in D of n that satisfy P:

$$\mathit{answer}_D\,(n, P) \quad = \quad \{m : \mathit{Node}_D \mid M\, m = n \wedge P(D, m) \bullet \mathit{sub}_D\, m\}$$

We have defined the answers to be subtrees rather than their root nodes, because this seems more general — as confirmed by our treatment of nested queries below. For each document in $answer_D\,(n, P)$ we know a schema for which it is valid:

$$answerschema\,(n, P) \quad = \quad sub_S\, n$$

Crucial for the success or failure of our notion of query is the intention that "the user" can only "extract information out of a document" by means of querying. In other words, to "the user" documents are presented as an abstract datatype for which query answering is the only information retrieval operation. This is in contrast with the notion of schema: the predicate P in a query (n, P) over schema S may use all knowlwedge of S (see the example below). UNFORNTUNATELY, I DON'T KNOW WHETHER AND HOW THE PRECEDING SENTENCE CAN BE FORMALIZED.

> *Remark 1.* It might be wise to require that for a query (n, P) node n has a document type; asking for the materializations of a *seq* node n, say, may be interpreted as asking for the materializations of the highest nodes under n that do have a document type. We shall not elaborate this option here.
>
> *Remark 2.* One XQuery expression may express *several* paths or nodes; in that case the XQuery expression is, in our formalization, a *set* of queries.
>
> *Remark 3.* In order to bring structure into the answer, the query may also specify a schema S' that says how to present the query answers. An answer schema might even present the answers to several questions at once. For the time being we shall not elaborate this option here. We feel that, because of its simplicity, our notion of query is more useful for further use (in, for example, the notion of integration) than the more complex notion of query that delivers a document. It might be the case that our notion of view is what other people would call a 'query with a presentation of the answwers in a document'.

14 Example: nested queries. Let S be a schema, D a document valid for S, and q a query over S. Let furthermore q' and q'' be queries over $answerschema\,q$. Then the following set comprehension expresses a "nested query":

$$\begin{array}{l} \{D', A', A'' : DOCUMENT \\ \qquad\qquad \mid D' \in answer_D\, q \wedge A' \in answer_{D'}\, q' \wedge A'' \in answer_{D'}\, q'' \\ \qquad\qquad \bullet\ (A', A'')\} \end{array}$$

Note that in this example, the documents are dealt with as an abstract datatype: query answering is the only means to get information out of a document, but these information retrieval operations may be composed in whatever way you want (here: using set comprehension). I THINK THAT The XQuery formulation would look like:

```
<results>
for $D' in D/q return
   for $A in $D'/q', $A'' in $D'/q'' return
      <result> {$A'} {$A''} </result>
</results>
```

15 Example: nontrivial predicate. For schema S in Figure 6, we shall formulate the following query:

> What is the content of the second *element* c after the first occurrence of either *element* a, *element* $b1$, of *element* $b2$ (whichever is present) immediately below the root of the document?

For document D of Figure 6, the answer is a set with just one member, namely the (subtree of the) node labeled *content* 789. Note that all ingredients of the query makes sense in view of schema S, except for one: the notion of "the *second* element c" is well-defined only because we have stipulated that in a document (and not necessarily in a schema) the partial order on the non-attribute children of an *element* is total.

Let us now formalize the query as a pair (n, P). The node n of the query is:

n = the child of the only node labeled *elt* c in schema S.

This node has two materializations in document D. The predicate of the query is:

$P(D', m')$ $\Leftrightarrow$ node m' in D' is a child of a node that is the second node after the first child of the root labeled *element* a, *element* $b1$, or *element* $b2$.

Here, the 'second' is well-defined in view of the total order on the *element* children of the root of a document; the 'first' is already well-defined in view of schema S (which specifies that the root of the document shall have a sequence of children of either an a, $b1$, or $b2$ element alternately with a c element).

If the predicate would have expressed the condition that the m' node must be the third one after the first a, $b1$, or $b2$ element, then the answer in D is still well defined, but empty.

Note that the predicate is mathematically perfectly defined; but it is not evident what a suitable (powerful, easy to use, convenient, concise) syntax is to express such predicates — XQuery might be a practical solution.

16 Reseach question. Suppose we have a schema S and a query q over S. Let D be a document valid for S. It may be the case that there exists another schema S' such that q is also a query over S' and D also is valid for S'. Do we want, and does it follow, that $answer_D\, q = answer_{D'}\, q$?

Views and Integration

17 Added in proof. Requirements for the notion of view over several schemas.

- A view over a schema S_1 is a schema S together with a way to construct a document D valid for S out of a document D_1 valid for S_1.
- For the construction of D, the only way to get information out of D_1 is via queries over S_1 (answerd in D_1) (plus further use of -mathematical- operations).
- Queries over S answered in D cannot reveal more information than queries over S_1 answered in D_1. (This requirement is met if the previous one is met.)

- Supposing that an answer (in D_1) to a query (over S_1) is a set or list (of subtrees or nodes), is it allowed to use the *size* of that set or list in the construction of D (rather than, or next to, using the *individual* items separately)?
  Assuming the previous claim, it depends on the notion of query.

- We already have defined our notion of query. Can a user compute the size of an answer set, or can a user only map a function to all items in the answer set of a query. (Huh, probably an unsound question. What do I mean, what do I wish to say?)

- It might be the case that our notion of view (in which our notion of query plays a role) is the same as what other people (and XQuery) would call a query.

* * *

Whenever S is an integration of S_1 and S_2, it is the case that part of S is a view of S_1, and also that part of S is a view of S_2. Therefore we first investigate the notion of view.

18 Goal. Let S_1 be a schema. A *view* of S_1 is a schema S in which each *content* node "stands for" a query over S_1: the query over S_1 associated to node n in S is given by a function $query_1$. Formally, a *view* over S_1 is a tuple $(S, query_1, mkval)$ in which:

S	is a true schema
$query_1$	is a function mapping *content* nodes of S to queries over S_1
$mkval$	see below

[SURELY, ASSOCIATING A QUERY TO EACH *content* NODE OF S, AND NONE TO EACH *element* NODE, IS OKAY, BUT WHAT ABOUT THE *seq* NODES? AS IT STANDS NOW, A *SEQUENCE* IN THE DOCUMENT WILL BE AN INFINITE REPETITION OF THE SAME SUBTREES!] Since S is a true schema, the notions of query and answer over S are well defined, but a document is needed in order to produce an answer. Since S is somehow related to S_1, we wish to be able to construct a document D valid for S out of a document D_1 valid for S_1. In the construction of D function *mkval* plays a role: the value of *content* nodes in D corresponding to node n in S, are given by $mkval\, v$ where v varies over $answer_{D_1}(query_1\, n)$. (Maybe, function *mkval* also needs S_1, D_1, and n as arguments. For simplicity we shall not do so until the proofs force us). Since we want D to be valid for S, it turns out that the cardinalities in S cannot be arbitrary, unrelated to those of S_1: there would be an inconsistency if S says that a node has cardinality $2\,..\,3$, say, whereas document D has 4 materializations of that node (the possibility of which depends on the cardinalities in S_1 and the construction of D by means of *mkval*).

We shall, in addition to the well-defined notions of query and answer over schema S (relative to a document D), also define new notions of query and answer over a view $(S, query_1, mkval)$ (relative to documents D_1 valid for S_1). We wish these different ways of querying and answering to yield the same result:

$$\begin{tikzcd}
S_1 \arrow[r, dash, "\text{reln involving } query_1"] & S \\
D_1 \arrow[u, "materialization\ M_1"] \arrow[r, dash, "\text{reln involving } mkval"] & D \arrow[u, swap, "materialization\ M"]
\end{tikzcd}$$

The answer to a query *over the view* $(S, query_1, mkval)$ is in D_1, constructed via the top-and-left path; the answer to a query *over the schema* S is in (D, which in turn is constructed out of) D_1, constructed via the right-and-bottom path.

19 Integration. An integration is nothing more than a view of several schemas simultaneously. The generalization of a view of *one* schema to a view of *several* schemas is obvious:

$$\begin{tikzcd}
S_1 & \dots & S_k \arrow[rr, dash, "\text{reln involving } query_{1..k}"] & & S \\
D_1 \arrow[u, "materialization\ M_1"] & \dots & D_k \arrow[u, "M_k"] \arrow[rr, dash, "\text{reln involving } mkval"] & & D \arrow[u, "materialization\ M"']
\end{tikzcd}$$

Now $query_i$ yields a query over S_i, and *mkval* takes as argument a k-tuple of answers.

20 Overview of the construction of D. Let $(S, query_1, mkval)$ be a view over S_1. Let D_1 be valid for S_1. We're going to construct a document D valid for S.

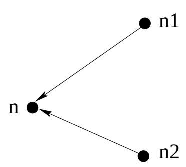
We shall first form an infinite $\hat{D}$ (out of D_1) which we then prune to a (finite) $\hat{D}'$, that in turn is edge-contracted to the requested document D:

$$\begin{tikzcd}
S \arrow[r, mapsto] & \hat{D} \arrow[r, mapsto, "\text{prune}"] & \hat{D}' \arrow[r, mapsto, "\text{edge-contract}"] & D \\
S_1 \arrow[r, dash, "\text{validity}"] & D_1 \arrow[u]
\end{tikzcd}$$

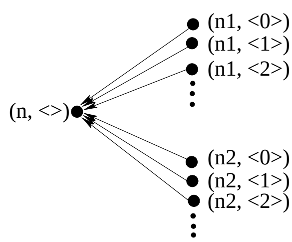
Artefact $\hat{D}'$ will be the intermediate for the validity to D to S. Artefact $\hat{D}$ is an auxiliary one that makes it easy to define $\hat{D}'$: the auxiliary $\hat{D}$ has nodes that are *very regularly* formed out of those of S, but with the consequence that it is infinitely branching (containing a lot of unwanted leaf nodes, and therefore also unwanted internal nodes and edges).

The structural part of document D that we're going to construct (hence also the structural part of $\hat{D}'$) is entirely determined by schema S; there is no choice for us in the construction of D if D has to be valid for S [assuming that for each document valid for a schema there exists exactly one intermediate artefact and one materialisation that establish the validity — see paragraph 11]. However, the content values —and only these— come from D_1 (via function $query_1$) and not from S. Of course, if the view is "erroneous", then our construction will end up with a document D that is *not* valid for S. We hope to find the right requirements for a view by our construction below, in order to rule out erroneous views.

21 Step 1: defining $\hat{D}$. Essentially, $\hat{D}$ is a infinitely unfolded copy of S:



in S becomes in $\hat{D}$:



Slightly more generally, consider a node n in S at a distance of, say, two edges from the root, and let n' be a child of n. Then these nodes are copied as nodes in $\hat{D}$ in the form

$(n, \langle p, q\rangle)$ and $(n', \langle p, q, r\rangle)$, for arbitrary numbers p, q, r; node $(n, \langle p, q\rangle)$ is the parent of node $(n', \langle p, q, r\rangle)$, for all r. Formally:

$$\begin{aligned}
\mathit{Node}_{\hat{D}} &= \{n : \mathit{Node}_S;\ \mathit{xs} : \mathrm{seq}\,\mathbb{N} \mid \#\mathit{xs} = \#(\mathit{paths}_S\, n) \bullet (n, \mathit{xs})\}\} \\
\mathit{Edge}_{\hat{D}} &= \{n, n' : \mathit{Node}_S;\ \mathit{xs} : \mathrm{seq}\,\mathbb{N};\ x : \mathbb{N} \\
&\qquad\qquad \mid n \hookleftarrow_S n' \wedge \#\mathit{xs} = \#(\mathit{paths}_S\, n) \quad \bullet \quad ((n, \mathit{xs}), (n', \mathit{xs} \frown \langle x\rangle)) \quad \}
\end{aligned}$$

The partial order in $\hat{D}$ is taken over from S in the obvious way; it is the least relation $\preceq$ on $\mathit{Node}_{\hat{D}}$ such that:

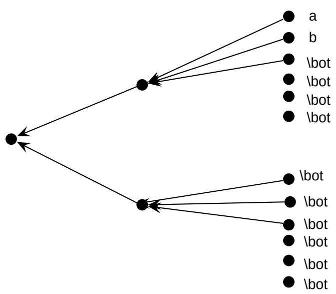
$$\begin{aligned}
&n \preceq_S n' &&\Rightarrow \quad (n, \mathit{xs}) \preceq (n', \mathit{xs}') \\
&\mathit{lab}_S\, n = \mathit{seq} \wedge n \hookleftarrow_S n' \wedge i \leq j &&\Rightarrow \quad (n', \mathit{xs} \frown \langle i\rangle) \preceq (n', \mathit{xs} \frown \langle j\rangle)
\end{aligned}$$

The node labels are inherited from S, except for those that have type *content*; the content in the view is given by queries over the underlying schema(s) and the answers in the corresponding document(s):

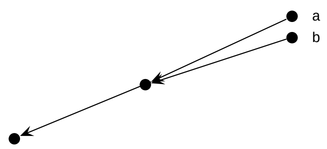
$$\begin{aligned}
\mathit{lab}_{\hat{D}}\,(n, \mathit{xs}) &= \mathit{lab}_S\, n, && \text{if } \mathit{types}_S\, n \neq \mathit{content} \\
\mathit{lab}_{\hat{D}}\,(n, \mathit{xs} \frown \langle i\rangle) &= (\mathit{content}, \mathit{mkval}\, v_i), && \text{if } \mathit{types}_S\, n = \mathit{content} \wedge i{<}j \\
&= (\mathit{content}, \perp), && \text{if } \mathit{types}_S\, n = \mathit{content} \wedge i{\geq}j \\
&\quad \text{where} \\
&\quad \langle v_0, \ldots, v_{j-1}\rangle = \text{ the set } \mathit{answer}_{D_1}(\mathit{query}_1\, n) \text{ ordered in a sequence according to the total(!) order } \preceq_{D_1}
\end{aligned}$$

Below we will prune away the nodes that contain the fictitious value '$\perp$', together with the appropriate edges.

22 Step 2: pruning $\perp$. Thanks to the use of an *infinite* number of copies of nodes of S to form nodes of $\hat{D}$, rather than a *finite* number consistent with the cardinalities in S, artefact $\hat{D}$ was easily defined. The price to be paid for this ease is the task of pruning away the superfluous nodes and edges. Fortunately, that's easy too. A node in $\hat{D}$ is deleted (to construct $\hat{D}'$) iff all its leaf nodes have $\perp$ ("`\bot`") in the label:



in $\hat{D}$ becomes in $\hat{D}'$:



Formally, intermediate $\hat{D}'$ has the following set of nodes and edges:

$$\begin{aligned}
\mathit{Node}_{\hat{D}'} &= \{n : \mathit{Node}_{\hat{D}} \mid \mathit{subtree}_{\hat{D}}\, n \text{ has a leaf node whose label doesnot contain } \perp\} \\
\mathit{Edge}_{\hat{D}'} &= \mathit{Edge}_{\hat{D}} \ \cap\ \mathit{Node}_{\hat{D}'} \times \mathit{Node}_{\hat{D}'}
\end{aligned}$$

The node labels and partial order in $\hat{D}'$ are the appropriate restrictions of the node labels and partial order in $\hat{D}$. (So, $\hat{D}'$ is a retract of $\hat{D}$ via the identity function.)

23 Step 3: defining D. We define document D to be the *Comptype*-contraction of $\hat{D}'$.

24 Theorem: D is valid for S. Define mapping M from $Node_{\hat{D}}$ to $Node_S$ as follows:

$$M\,(n, xs) \quad = \quad n$$

We claim that $\hat{D}'$ together with M together establish that D is valid for S. By construction, D is the *Comptype*-contraction of $\hat{D}'$. It is also quite obvious (and true, I hope) that function M preserves the edges, the root, the partial order, the types, and the names (where appropriate). Depending on *mkval*, $query_1$ (and D_1?), it also "weakly preserves *value*" (where appropriate; in the sense that $M\,(value_{\hat{D}}\,m) \in values_S\,(M\,m)$). In order to make sure that it does so, a constraint must be placed on *mkval*, $query_1$, S and S_1. THE FOLLOWING SEEMS SUFFICIENT:

$$mkval(\!|\ values_{S_1}\,(\text{the node of } query_1\,n)\ |\!) \quad \subseteq \quad values_S\,n \qquad \text{for all } content \text{ nodes } n \text{ in } S$$

In other words:

$$\forall\, v : values_{S_1}\,(\text{the node of } query_1\,n) \bullet mkval\,v \in values_S\,n \qquad \text{for all } content \text{ nodes } n \text{ in } S$$

It is not yet clear that M respects the cardinalities, nor that M respects choice. HERE AGAIN WE NEED TO USE REQUIREMENTS TO BE PLACED ON A VIEW, SO THAT —UNDER THOSE REQUIREMENTS— IT IS TRUE THAT M RESPECTS CARDINALITIES AND CHOICE. STILL TO DO. SOME INGENUITY REQUIRED.........

25 Answering queries on S directly in terms of queries over S_1. TO BE DONE.

Let S_1 and a document D_1 confroming to S_1 be given. Let $(S, query_1, mkval)$ be a view over S_1. Let q be a query over S; so q is of the form (n, P) for some node n of S and some predicate $P(\_,\_)$ (taking as argument a document valid for S and a node in that document).

26 Theorem: Commutativety of the diagram. Answering a query over S in D gives the same result as via queries over S_1 on D_1. See the commutativety diagram in paragraph 18. TO BE DONE.