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
- Ullman and Widom, **A first course in database systems**, Prentice-Hall International, 1997. *For some notions on relational databases.*
- 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

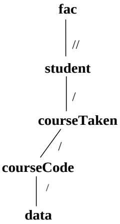
A path for an XML tree t is defined by a sequence of tags or labels.
Our path languages : used to define integrity constraints over XML trees :

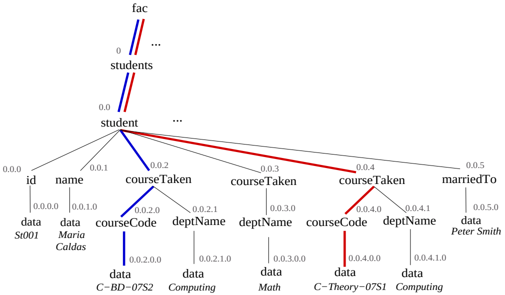
- Path language PL_s (defined by $\rho ::= \ l \mid \rho/\rho \mid \_$).
- Path language PL (defined by $\upsilon ::= [\,] \mid \rho \mid \upsilon//\rho$).
- The language PL_s describes a path in t, while PL is a generalization of PL_s including "$//$".
- A path P is **valid** if it conforms to the syntax 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 $l = v_1/\ldots/v_n$ be a sequence of positions such that each v_i is a direct descendant of v_{i-1} in t.
- **l 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.



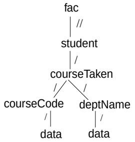


Patterns and pattern instances

- 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.

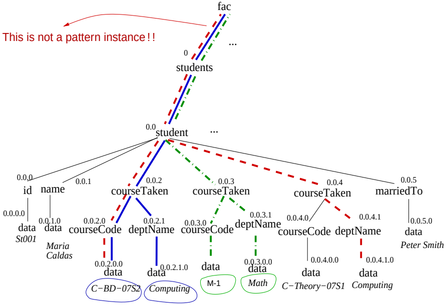
A set of paths : $\{fac//student/courseTaken/courseCode,$
$fac//student/courseTaken/deptName\}))$

Common prefix : $fac//student/courseTaken$



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

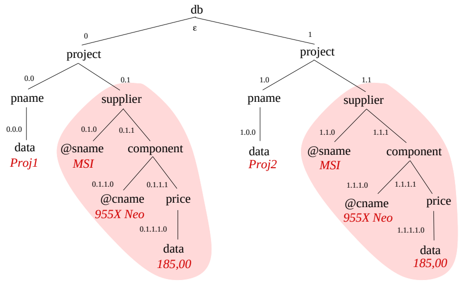
$\tau_{C/P,C/Q} = [M-1, Math]$



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 the same position.

*Nodes 0.1 and 1.1 are **value equal**.*



Functional Dependencies in Relational Databases

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

$$X \to A$$

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

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

Consider the functional dependency *Name*, *Year* $\to$ *Activity*

Name	Year	Activity
Mario	2010	theatre
Diana	2010	theatre
Peter	2010	volleyball
Barbara	2011	volleyball
Mario	2011	football

XFD Syntax

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

Notation

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

- 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

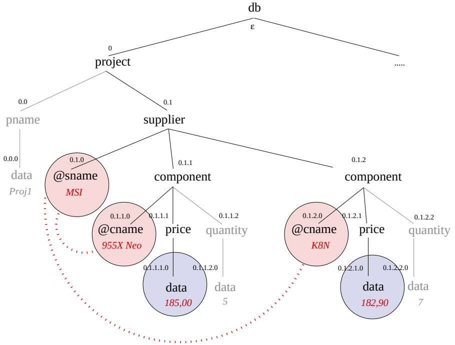
$\mathcal{T} \models \gamma$ if and only if for all two instances (I^1, I^2) of pattern $\mathcal{P}$ in $\mathcal{T}$ that coincide at least on their prefix C, we have :

$$\tau^1[C/P_1, \ldots, C/P_k] =_{E_i, i \in [1 \ldots k]} \tau^2[C/P_1, \ldots, C/P_k] \Rightarrow \tau^1[C/Q] =_E \tau^2[C/Q]$$

where τ^1 (resp. τ^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.

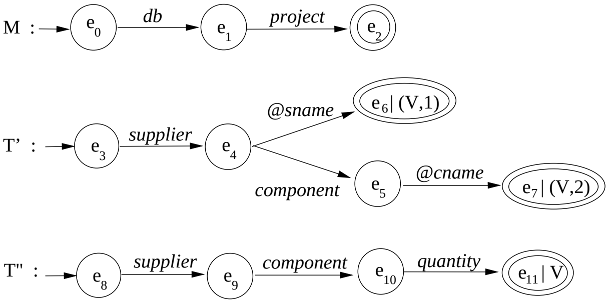
Building an attribute grammar

- 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 : $ROOT \rightarrow \alpha_1 \dots \alpha_m$.
  - Rule for an internal element node : $A \rightarrow \alpha_1 \dots \alpha_m$.
  - Rule for an element containing data and for an attribute : $A \rightarrow data$.

FSA and TSAs for XFD

To model the paths of an XFD, we use finite-state automata (FSA) or transducers (FST).

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

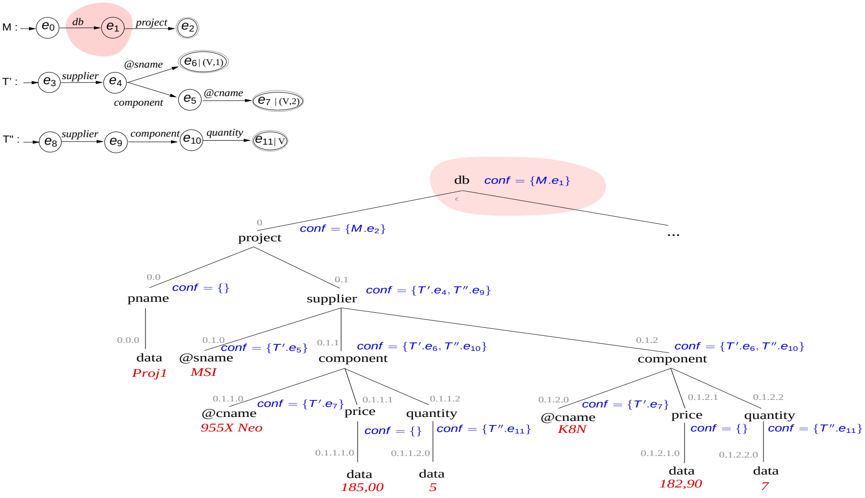


Attribute Grammar for XFD Validation

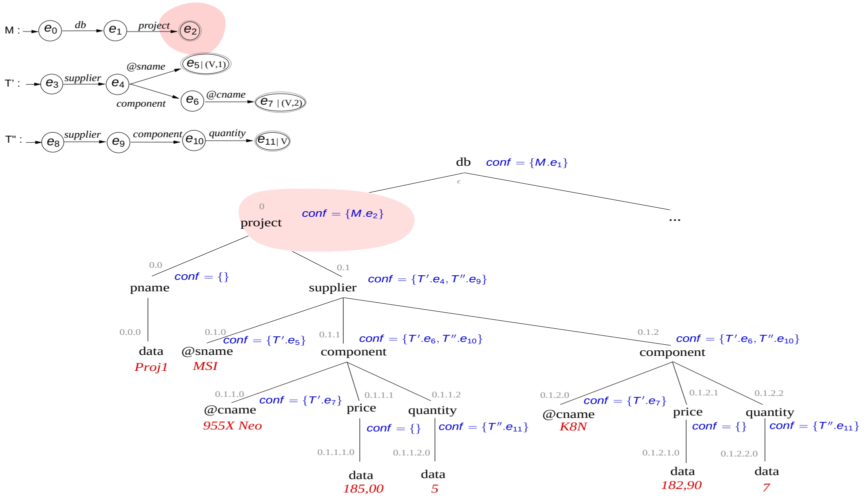
Descending Direction : Inherited Attribute ***conf***

- ***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

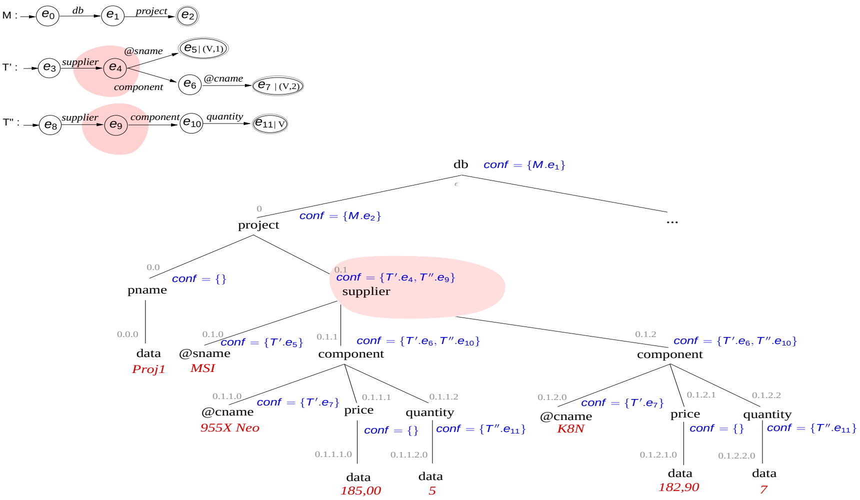
Example : Descending direction



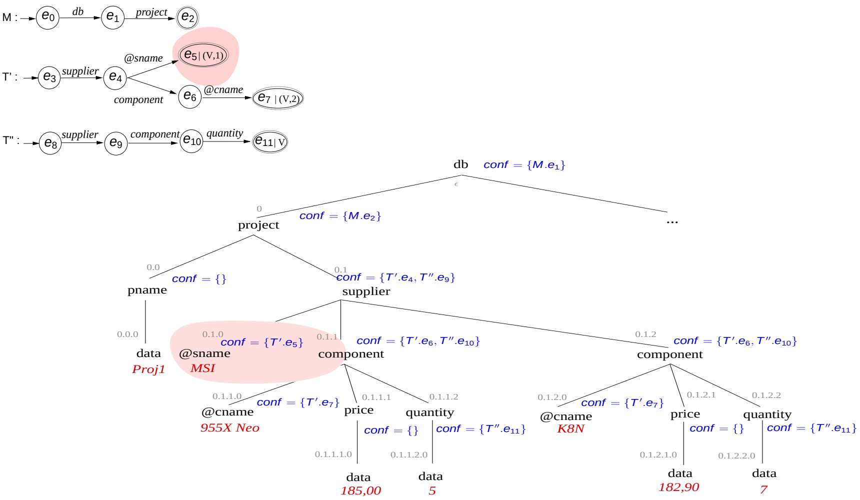
Example : Descending direction



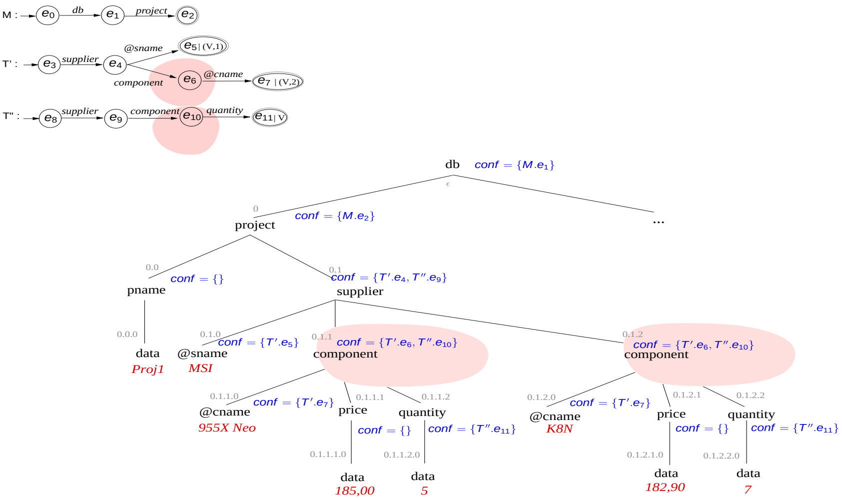
Example : Descending direction



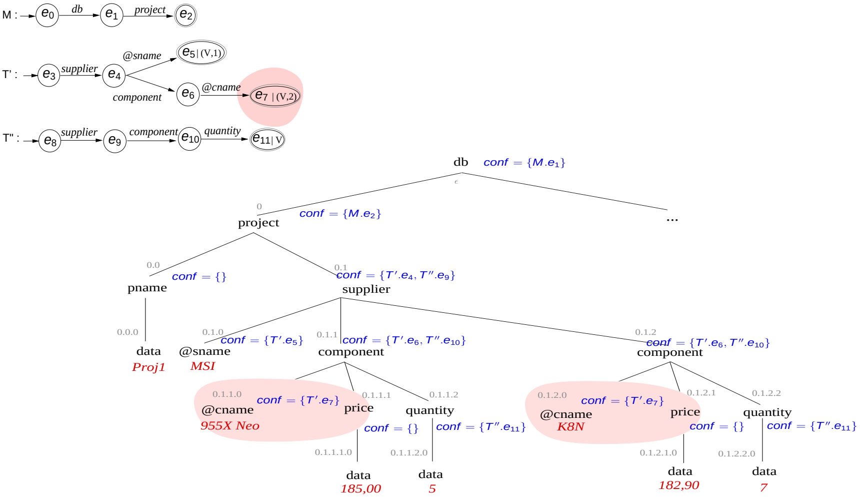
Example : Descending direction



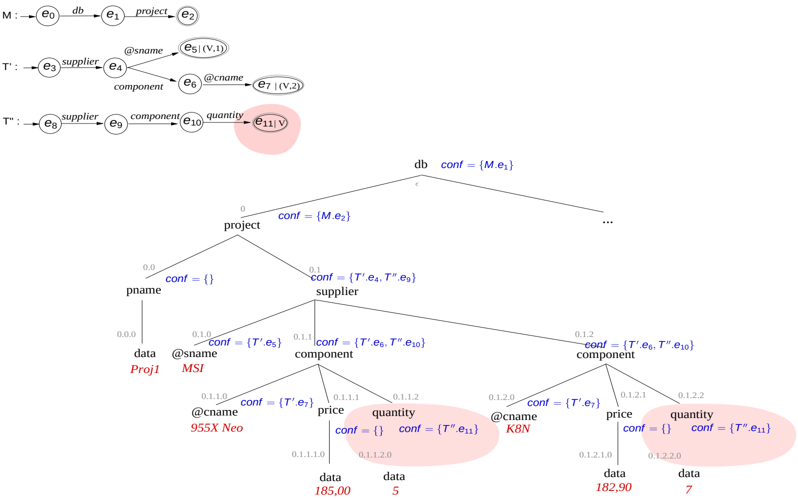
Example : Descending direction



Example : Descending direction



Example : Descending direction



Just an example of a rule in the attribute grammar

Production	**Attributes**

$ROOT \rightarrow \alpha_1 \ldots \alpha_m$	$ROOT.conf := \{ M.q_1 \mid \delta_M(q_0, ROOT) = q_1 \}$
	/ Inherited Attributes */*
	for each α_i $(1 \leq i \leq m)$ do
	$\alpha_i.conf := \{ M.q' \mid \delta_M(q_1, \alpha_i) = q' \}$
	if $(q_1 \in F_M)$ then
	$\alpha_i.conf := \alpha_i.conf \cup \{ T'.q'_1 \mid \delta_{T'}(q'_0, \alpha_i) = q'_1 \}$
	$\cup \{ T''.q''_1 \mid \delta_{T''}(q''_0, \alpha_i) = q''_1 \}$

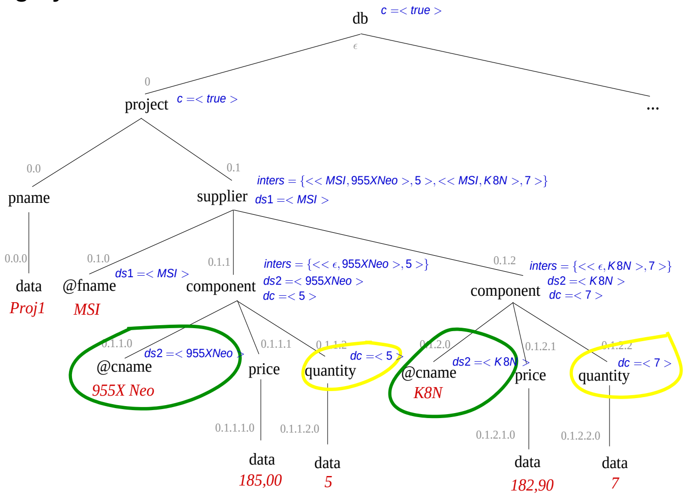
Attribute Grammar for XFD Validation

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

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

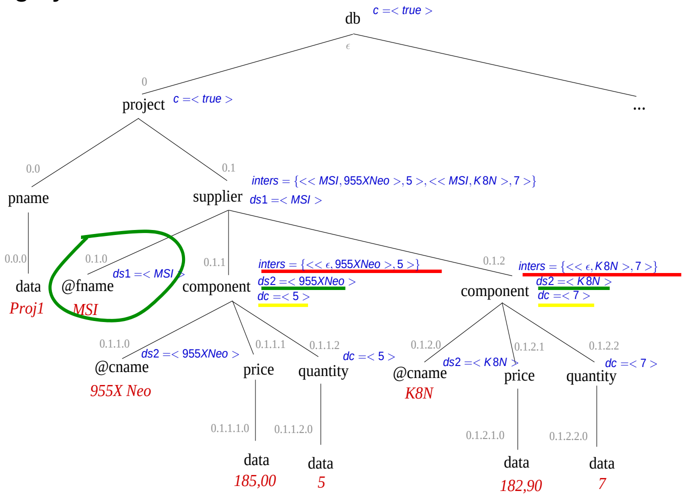
Example : Ascending direction

Computing synthesized attributes :



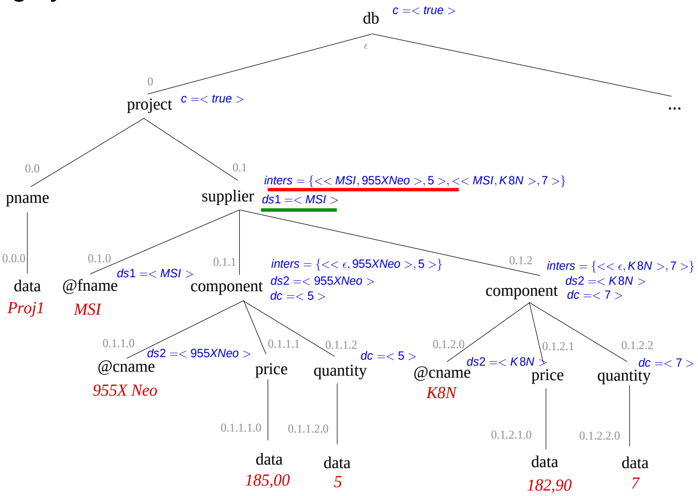
Example : Ascending direction

Computing synthesized attributes :



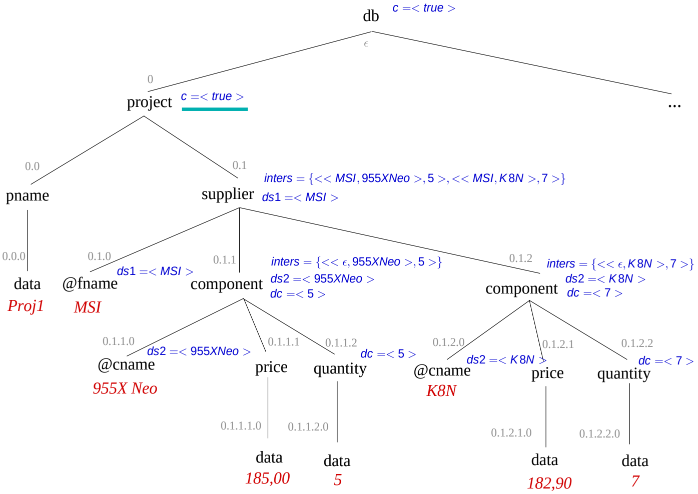
Example : Ascending direction

Computing synthesized attributes :



Example : Ascending direction

Computing synthesized attributes :

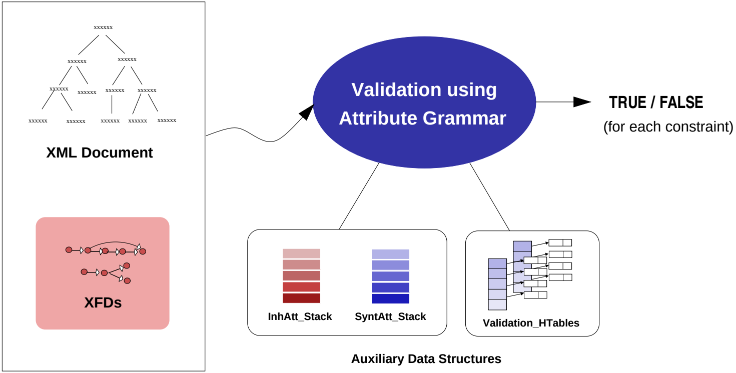


Just an example of a rule in the attribute grammar

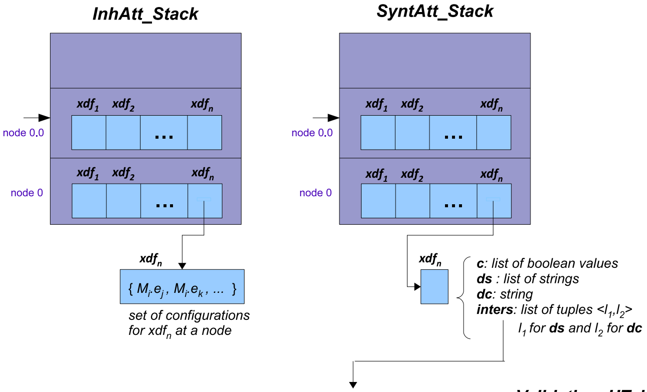
Production	**Attributes**

$A \rightarrow data$	*/* Synthesized Attributes */*
	for each configuration $\overline{M}.q$ in $A.conf$ do
	    if $(\overline{M} = T') \wedge (q \in F_{T'})$
	        $y := \lambda'_{T'}(q)$
	        $j := y.rank$
	        if $(y.equality = V)$
	            then $A.ds_j := < value(t, data) >$
	            else $A.ds_j := < value(t, A) >$
	    if $(\overline{M} = T'') \wedge (q \in F_{T''})$
	        if $(\lambda''_{T''}(q) = V)$
	            then $A.dc := < value(t, data) >$
	            else $A.dc := < pos(t, A) >$

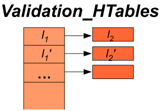
XFD Validation Overview



XFD Validation Overview : Auxiliary Structures



The values in tuples <l1,l2> are mantained in a hashtable, which is used to verify the XFD:



Constr.	Path expression	FSA	Attributes
XFD	$(C, (\{P_1\ [E_1], \ldots, P_k\ [E_k]\} \rightarrow Q\ [E]))$	M, T and T'	Inherit. : : *conf* Synth. : $c, inters, ds_j, dc$
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