# CONTRIBUTION À L'ANALYSE DE L'IMPACT DE MISES À JOUR SUR DES VUES ET SUR DES CONTRAINTES D'INTÉGRITÉ XML.

Hicham Idabal


Centre de Recherche en Informatique

# Contribution à l'analyse de l'impact de Mises à jour sur des Vues et sur des Contraintes d'Intégrité XML

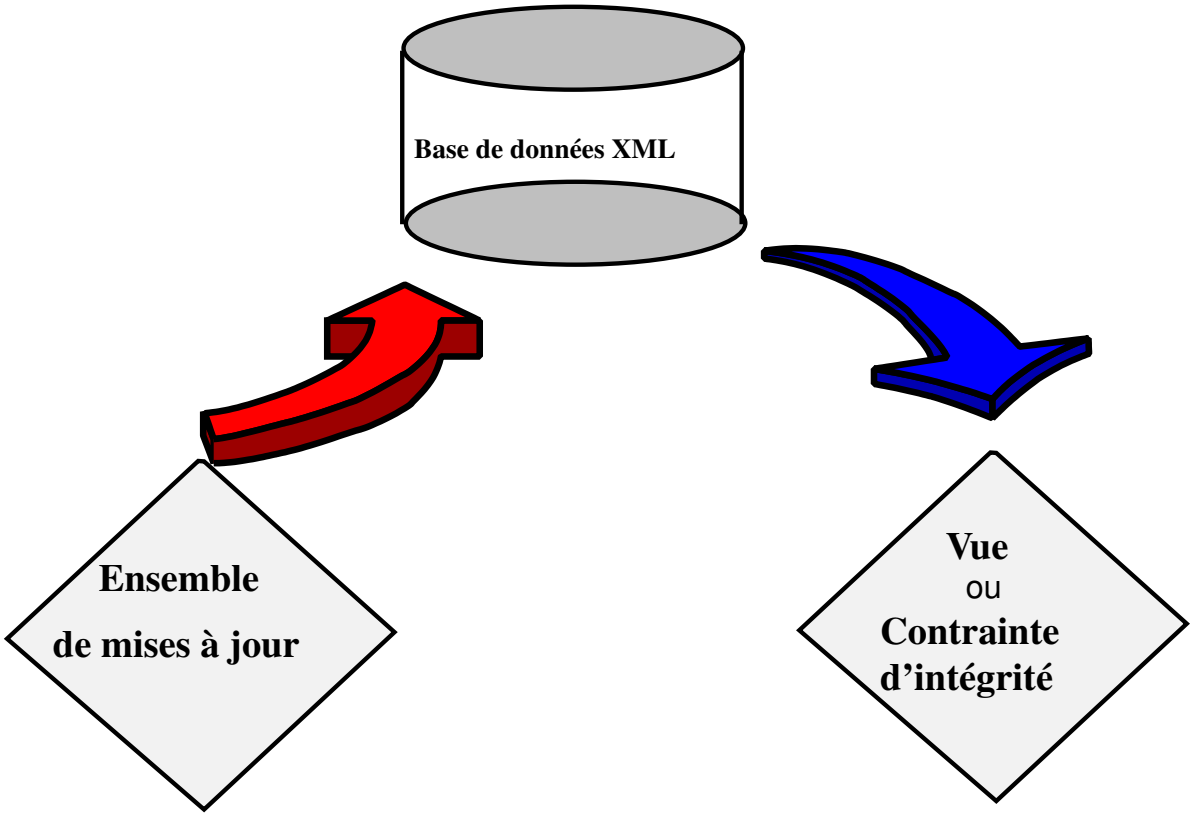
Hicham Idabal

Encadré par **Pr. Françoise Gire**

25 novembre 2010

# Sommaire

# Introduction : contexte général



# Introduction : positionnement

L'impact des mises à jours : un problème classique largement étudié.

Principales différences par rapport aux travaux antérieurs :

1. La plupart des travaux utilisent des informations :
   - provenant des sources
   - provenant de couches intermédiaires

   $\Rightarrow$ On n'utilise que la définition des requêtes (Vue, MAJ, CI) et un schéma s'il existe.

2. Les langages utilisés sont différents :
   - La plupart des travaux utilisent des langages proches des standards du W3C.

   $\Rightarrow$ On fait le choix d'un langage formel puissant à base de motifs réguliers arborescents : $\mathcal{RARs}$

# Introduction : positionnement

L'impact des mises à jours : un problème classique largement étudié.

Principales différences par rapport aux travaux antérieurs :

1. La plupart des travaux utilisent des informations :
   - provenant des sources
   - provenant de couches intermédiaires

   $\Rightarrow$ On n'utilise que la définition des requêtes (Vue, MAJ, CI) et un schéma s'il existe.

2. Les langages utilisés sont différents :
   - La plupart des travaux utilisent des langages proches des standards du W3C.

   $\Rightarrow$ On fait le choix d'un langage formel puissant à base de motifs réguliers arborescents : $\mathcal{RARS}$

# Introduction : positionnement

L'impact des mises à jours : un problème classique largement étudié.

Principales différences par rapport aux travaux antérieurs :

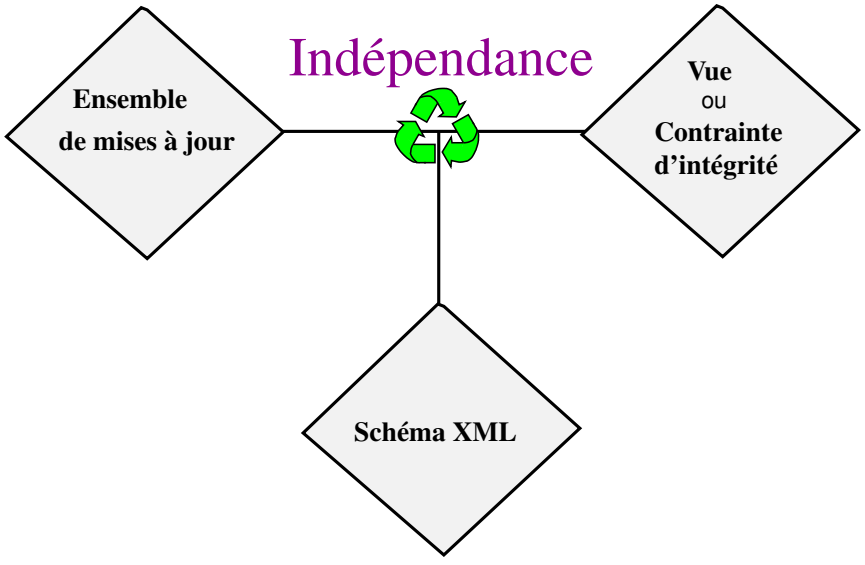
1. La plupart des travaux utilisent des informations :
   - provenant des sources
   - provenant de couches intermédiaires

   $\Rightarrow$ On n'utilise que la définition des requêtes (Vue, MAJ, CI) et un schéma s'il existe.

2. Les langages utilisés sont différents :
   - La plupart des travaux utilisent des langages proches des standards du W3C.

   $\Rightarrow$ On fait le choix d'un langage formel puissant à base de motifs réguliers arborescents : $\mathcal{RARs}$

# Introduction : problématique




**But : détection de l'indépendance**

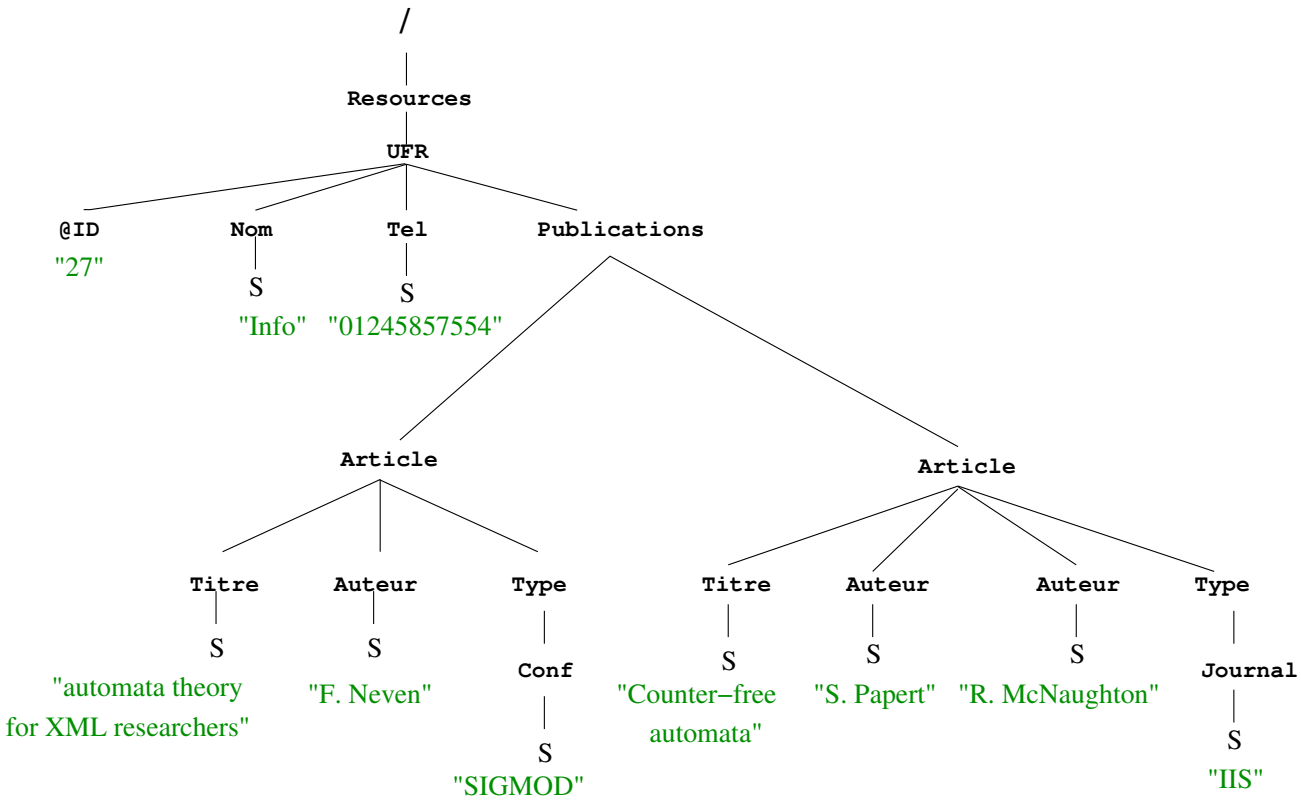
Eviter :

- une réévaluation de la Vue, ou
- une revalidation de la Contrainte d'intégrité

# Sommaire

# Un document XML

Une représentation en arbres ordonnés étiquetés des documents XML : $\mathcal{D}=(d, \lambda, val)$

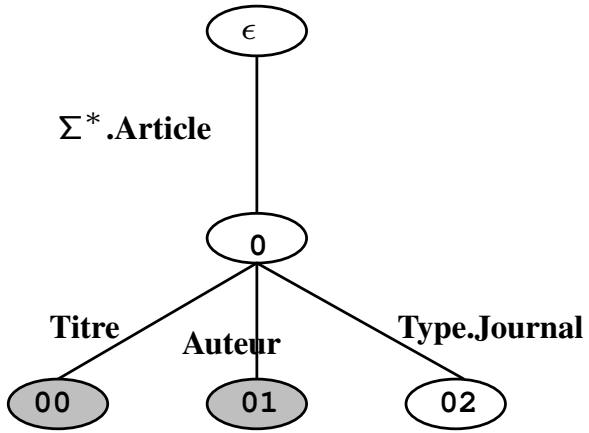


# Requêtes Arbres Régulières ($\mathcal{RARs}$)

$\Rightarrow$ **Modéliser le processus de sélection de nœuds par un arbre**

## Exemple (Vue)

- "Donner les (Titre,Auteur) des articles publiés dans un journal"



# $\mathcal{RAR}$ : définition formelle

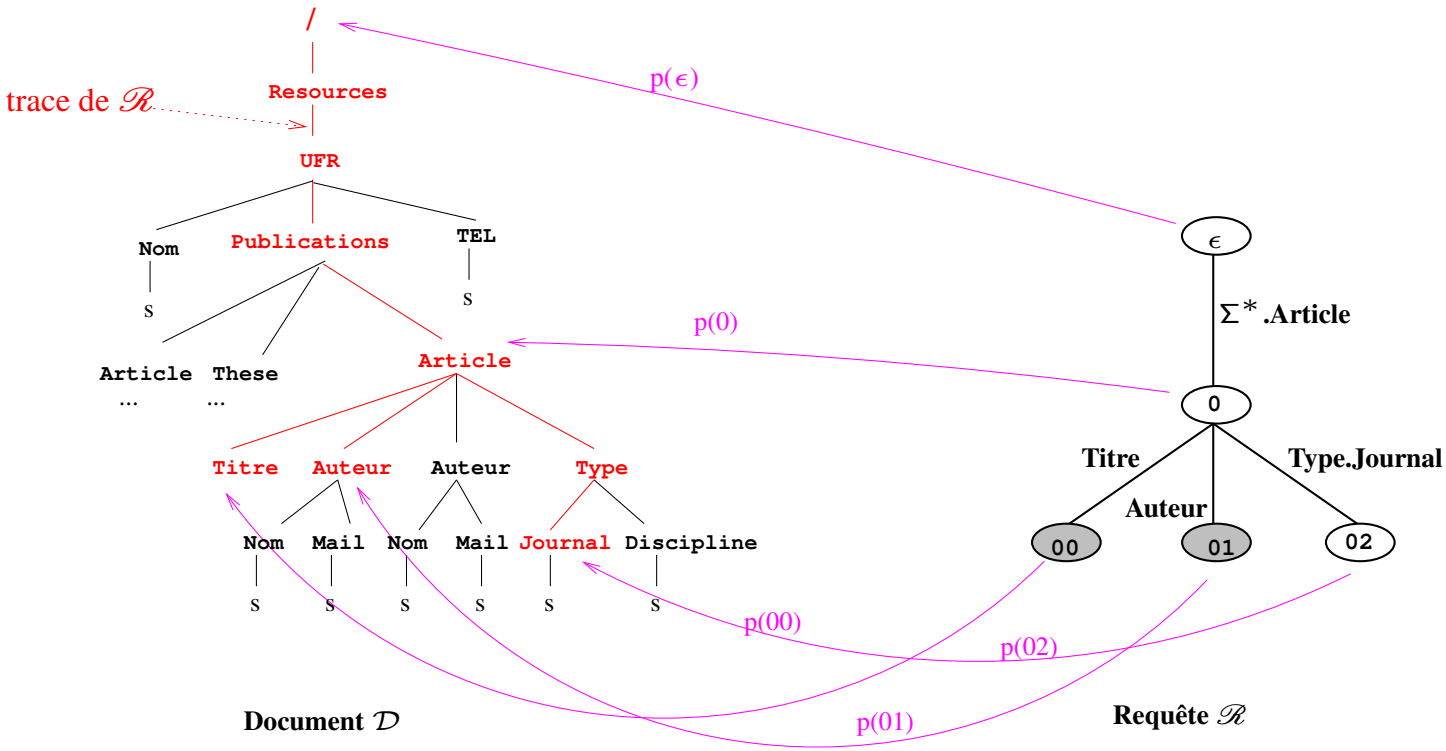
**Définition :** Une requête $\mathcal{RAR}$ $\mathscr{R}$ est définie par $\mathscr{R} = (\mathscr{T}, \vec{s})$ où $\mathscr{T} = (\Sigma, N, M, \mathcal{E})$ est appelé arbre template de $\mathscr{R}$

- $N \subseteq \mathbb{N}^*$ : le domaine de l'arbre.
- $M \subseteq N \times N$ : ensemble de ses arcs.
- $\mathcal{E} : M \longrightarrow REG(\Sigma)$ : application qui associe à chaque arc $(w, w')$ de $M$ une expression notée $\mathcal{E}_{(w,w')}$ qui est, soit vide, soit régulière propre.
- $\vec{s}=(w_1, ..., w_n)$ : n-uplet de nœuds de $N$ à sélectionner

# Évaluation d'une $\mathcal{RAR}$

Soit $\mathscr{R} = (\mathscr{T}, \vec{s})$ une $\mathcal{RAR}$ où $\mathscr{T} = (\Sigma, N, M, E)$ une requête arbre et $\mathcal{D}$ un document XML

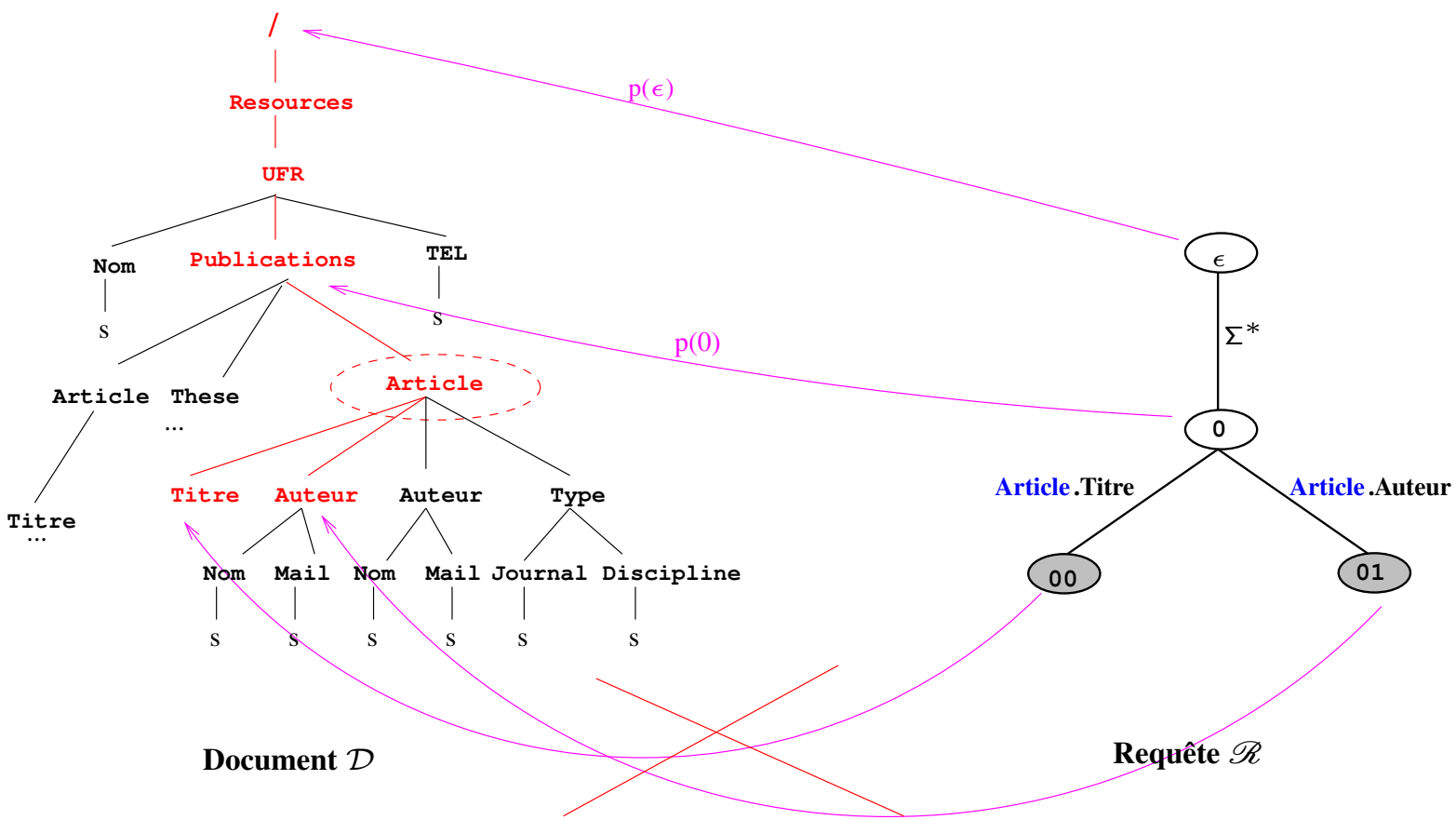
**Définition :** Un plongement de $\mathscr{R}$ dans $\mathcal{D}$ est une application $p : N \longrightarrow \mathcal{N}(\mathcal{D})$



# Évaluation d'une $\mathcal{RAR}$

Soit $\mathscr{R} = (\mathscr{T}, \vec{s})$ une $\mathcal{RAR}$ où $\mathscr{T} = (\Sigma, N, M, \mathcal{E})$ une requête arbre et $\mathcal{D}$ un document XML

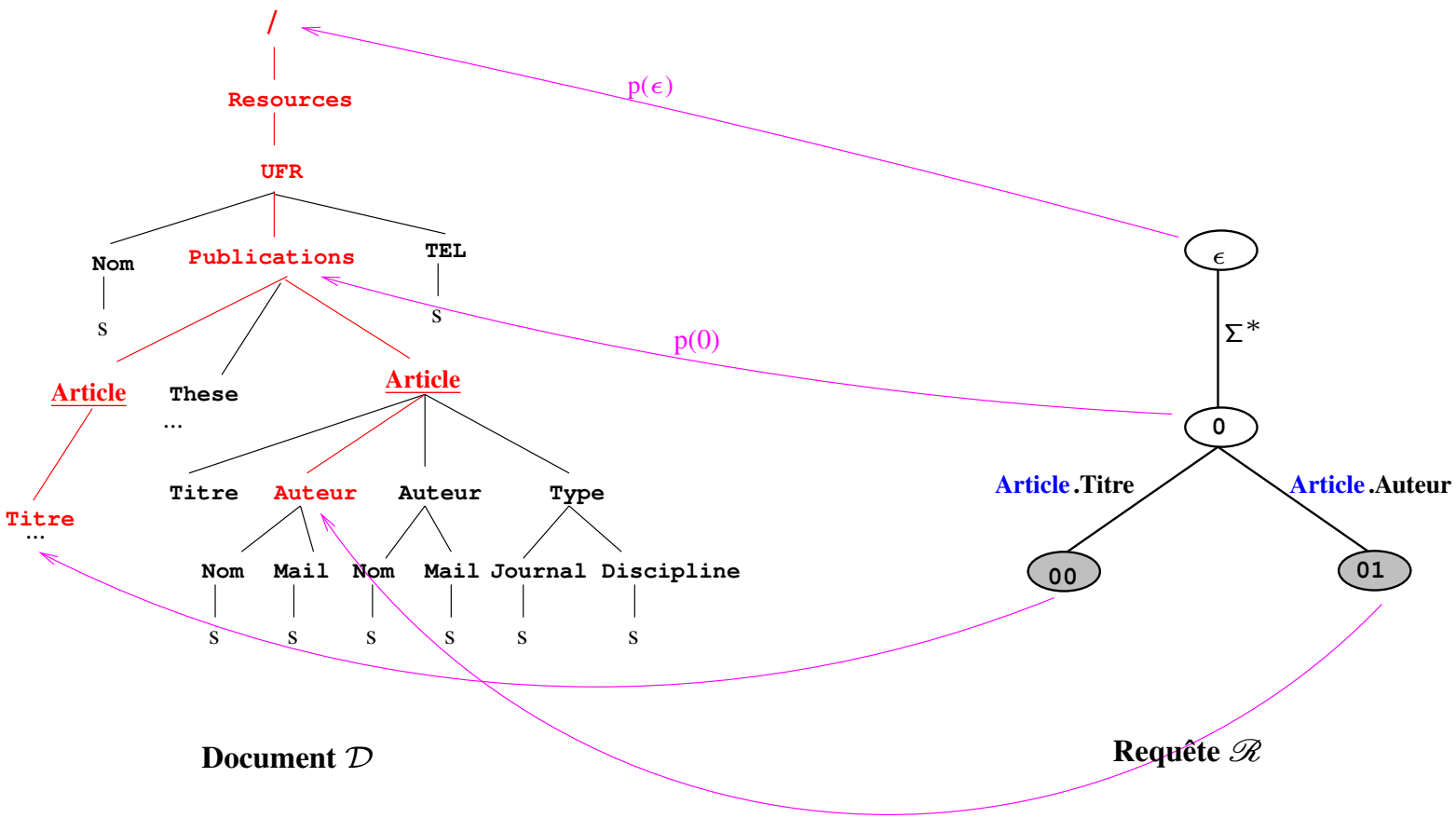
**Définition :** Un plongement de $\mathscr{R}$ dans $\mathcal{D}$ est une application $p : N \longrightarrow \mathcal{N}(\mathcal{D})$



# Évaluation d'une $\mathcal{RAR}$

Soit $\mathscr{R} = (\mathscr{T}, \vec{s})$ une $\mathcal{RAR}$ où $\mathscr{T} = (\Sigma, N, M, \mathcal{E})$ une requête arbre et $\mathcal{D}$ un document XML

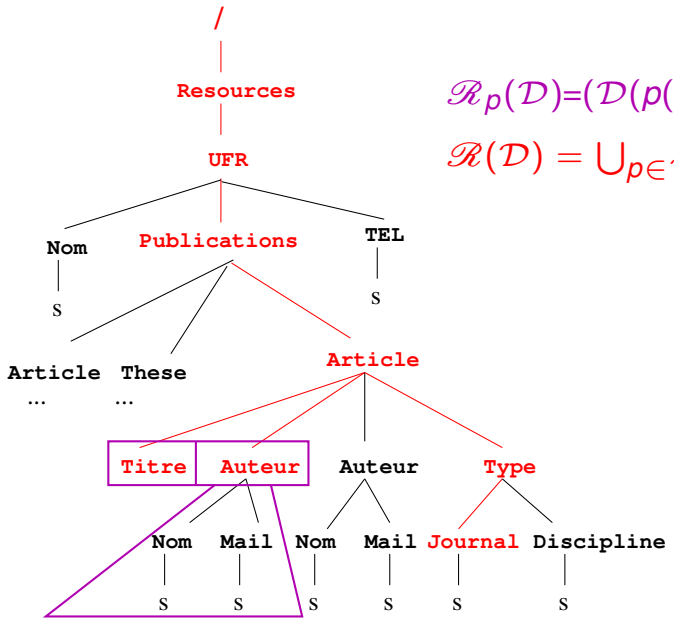
**Définition :** Un plongement de $\mathscr{R}$ dans $\mathcal{D}$ est une application $p : N \longrightarrow \mathcal{N}(\mathcal{D})$



# Évaluation d'une $\mathcal{RAR}$

Soit $\mathscr{R} = (\mathscr{T}, \vec{s})$ une $\mathcal{RAR}$ où $\mathscr{T} = (\Sigma, N, M, E)$ une requête arbre et $\mathcal{D}$ un document XML

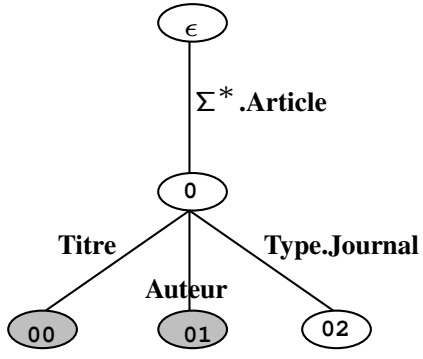
**Définition :** Un plongement de $\mathscr{R}$ dans $\mathcal{D}$ est une application $p : N \longrightarrow \mathcal{N}(\mathcal{D})$




Document $\mathcal{D}$

$\mathscr{R}_p(\mathcal{D})=(\mathcal{D}(p(w_1)), \ldots, \mathcal{D}(p(w_n)))$

$\mathscr{R}(\mathcal{D}) = \bigcup_{p \in \mathcal{P}} \mathscr{R}_p(\mathcal{D})$

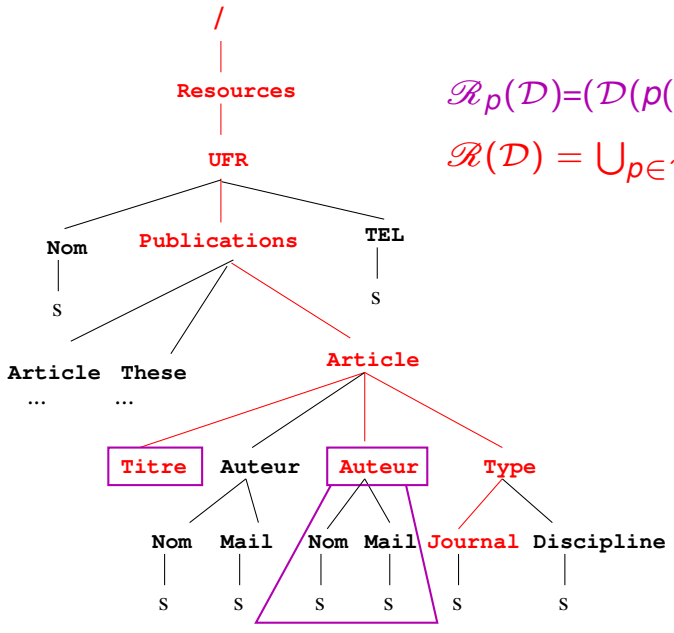



Requête $\mathscr{R}$

# Évaluation d'une $\mathcal{RAR}$

Soit $\mathscr{R} = (\mathscr{T}, \vec{s})$ une $\mathcal{RAR}$ où $\mathscr{T} = (\Sigma, N, M, E)$ une requête arbre et $\mathcal{D}$ un document XML

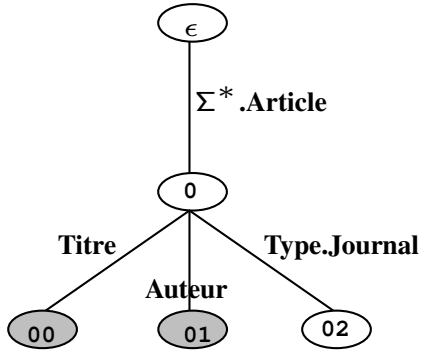
**Définition :** Un plongement de $\mathscr{R}$ dans $\mathcal{D}$ est une application $p : N \longrightarrow \mathcal{N}(\mathcal{D})$




Document $\mathcal{D}$

$\mathscr{R}_p(\mathcal{D})=(\mathcal{D}(p(w_1)), \ldots, \mathcal{D}(p(w_n)))$

$\mathscr{R}(\mathcal{D}) = \bigcup_{p \in \mathcal{P}} \mathscr{R}_p(\mathcal{D})$




Requête $\mathscr{R}$

# XPath

*XPath* **:**
Langage de requête, standard du W3C (1998).

$\Rightarrow$ sélectionne un ensemble d'éléments d'un document XML répondant à certaines contraintes structurelles ou textuelles.

Path = succession d'étapes (step).
Step = **axe**::**type[filtre]**

# $\mathcal{RAR}$ et XPath

## Modèle $\mathcal{RAR}$ :

- Modèle formel puissant.
- Exprime certaines requêtes que le langage standard *XPath* ne permet pas d'exprimer.

# $\mathcal{RAR}$ et XPath

**Modèle $\mathcal{RAR}$ :**

- Modèle formel puissant.
- Exprime certaines requêtes que le langage standard *XPath* ne permet pas d'exprimer.

$\Rightarrow$ *XPath* et le modèle des requêtes $\mathcal{RAR}s$ sont incomparables.

# $\mathcal{RAR}$ et XPath

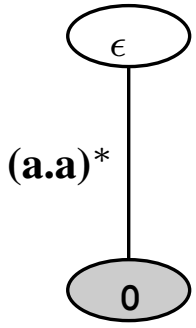
**Modèle $\mathcal{RAR}$ :**

- Modèle formel puissant.
- Exprime certaines requêtes que le langage standard *XPath* ne permet pas d'exprimer.

$\Rightarrow$ *XPath* et le modèle des requêtes $\mathcal{RAR}$s sont incomparables.

**Contre-exemples :**

1. "Donner tous les nœuds dont chaque ancêtre est étiqueté par un 'a' et le nombre des ancêtres est pair" : non exprimable par XPath



# $\mathcal{RAR}$ et XPath

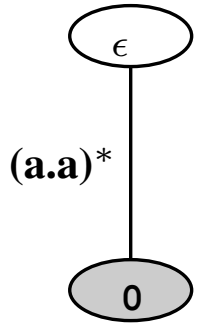
## Modèle $\mathcal{RAR}$ :

- Modèle formel puissant.
- Exprime certaines requêtes que le langage standard *XPath* ne permet pas d'exprimer.

$\Rightarrow$ *XPath* et le modèle des requêtes $\mathcal{RAR}$s sont incomparables.

## Contre-exemples :

1. "Donner tous les nœuds dont chaque ancêtre est étiqueté par un 'a' et le nombre des ancêtres est pair" : non exprimable par XPath
2. //a[not(./b)] : non exprimable par $\mathcal{RAR}$



# CoreXPath positif

*CoreXPath*$^+$ **:** Fragment navigationnel de *XPath* (sans négation, sans disjonction ).

Formellement, un chemin absolu $P_a$ de *CoreXPath*$^+$ est défini par la grammaire :

$$\begin{aligned} P_a &\equiv /P \\ P &\equiv axis :: \lambda \ \| \ P/axis :: \lambda \ \| \ P[F] \ \| \ P|P \\ F &\equiv axis :: \lambda \ \| \ axis :: \lambda[F] \ \| \ /F \end{aligned}$$

# CoreXPath positif

*CoreXPath*$^+$ : Fragment navigationnel de *XPath* (sans négation, sans disjonction ).

Formellement, un chemin absolu $P_a$ de *CoreXPath*$^+$ est défini par la grammaire :

$$\begin{aligned} P_a &\equiv /P \\ P &\equiv axis :: \lambda \ \| \ P/axis :: \lambda \ \| \ P[F] \ \| \ P|P \\ F &\equiv axis :: \lambda \ \| \ axis :: \lambda[F] \ \| \ /F \end{aligned}$$

axis : `self, child, parent, descendant, descendant_or_self, ancestor, ancestor_or_self, following_sibling, following, preceding_sibling, preceding`

# CoreXPath positif

*CoreXPath*$^+$ **:** Fragment navigationnel de *XPath* (sans négation, sans disjonction ).

Formellement, un chemin absolu $P_a$ de *CoreXPath*$^+$ est défini par la grammaire :

$$\begin{aligned} P_a &\equiv /P \\ P &\equiv axis :: \lambda \ \| \ P/axis :: \lambda \ \| \ P[F] \ \| \ P|P \\ F &\equiv axis :: \lambda \ \| \ axis :: \lambda[F] \ \| \ /F \end{aligned}$$

- *CoreXPath* a la même expressivité que $FO^2{}_{tree}$

# $\mathcal{RAR}$ et CoreXPath positif

**Proposition 1**

Tout chemin $P_a$ de $CoreXPath^+$ est exprimable par une union finie de requêtes $\mathcal{RAR}$ monadiques.

# XPath conditionnel : CXPath

*CXPath* **:** (M. Marx, 2005)

$$
\begin{array}{rcl}
\mathit{prim\_axis} & ::= & \mathit{child} \parallel \mathit{parent} \parallel \mathit{right} \parallel \mathit{left}. \\
P & ::= & \mathit{prim\_axis} \parallel P/P \parallel P \cup P \parallel (\mathit{prim\_axis}/?F)^* \parallel ?F \\
F & ::= & \lambda \parallel \top \parallel < P > \parallel \mathit{not}\ F \parallel F\ \mathit{and}\ F \parallel F\ \mathit{or}\ F
\end{array}
$$

# XPath conditionnel : CXPath

*CXPath* **:** (M. Marx, 2005)

$$
\begin{array}{rcl}
\mathit{prim\_axis} & ::= & \mathit{child} \parallel \mathit{parent} \parallel \mathit{right} \parallel \mathit{left}. \\
P & ::= & \mathit{prim\_axis} \parallel P/P \parallel P \cup P \parallel (\mathit{prim\_axis}/?F)^* \parallel ?F \\
F & ::= & \lambda \parallel \top \parallel < P > \parallel \mathit{not}\ F \parallel F\ \mathit{and}\ F \parallel F\ \mathit{or}\ F
\end{array}
$$

- CXPath est $FO_{tree}$-complet

# XPath conditionnel : CXPath

*CXPath* **:** (M. Marx, 2005)

$$
\begin{array}{rcl}
\mathit{prim\_axis} & ::= & \mathit{child} \parallel \mathit{parent} \parallel \mathit{right} \parallel \mathit{left}. \\
P & ::= & \mathit{prim\_axis} \parallel P/P \parallel P \cup P \parallel (\mathit{prim\_axis}/?F)^* \parallel ?F \\
F & ::= & \lambda \parallel \top \parallel < P > \parallel \mathit{not}\ F \parallel F\ \mathit{and}\ F \parallel F\ \mathit{or}\ F
\end{array}
$$

- CXPath est $FO_{tree}$-complet

**Exemple :**

Soit la requête qui sélectionne les nœuds origines d'un chemin dont tous les nœuds sont étiquetés par b sauf le dernier qui est étiqueté par a.

En CXPath : $?<(\mathit{child}/?b)^*/\mathit{child}/?a>$ (ou $//*[./b^*/a]$)

# XPath conditionnel : CXPath

*CXPath* **:** (M. Marx, 2005)

$$\begin{array}{rcl} \mathit{prim\_axis} & ::= & \mathit{child} \parallel \mathit{parent} \parallel \mathit{right} \parallel \mathit{left}. \\ P & ::= & \mathit{prim\_axis} \parallel P/P \parallel P \cup P \parallel (\mathit{prim\_axis}/?F)^* \parallel ?F \\ F & ::= & \lambda \parallel \top \parallel < P > \parallel \mathit{not}\ F \parallel F\ \mathit{and}\ F \parallel F\ \mathit{or}\ F \end{array}$$

- CXPath est $FO_{tree}$-complet

**Exemple :**

Soit la requête qui sélectionne les nœuds origines d'un chemin dont tous les nœuds sont étiquetés par b sauf le dernier qui est étiqueté par a.

En CXPath : $?<(\mathit{child}/?b)^*/\mathit{child}/?a>$ (ou //∗[./b*/a])

En $FO_{tree}$ : $\exists y(xR_{\Downarrow}y \wedge O_a(y) \wedge \forall z((xR_{\Downarrow}z \wedge zR_{\Downarrow}y) \rightarrow O_b(z)))$.

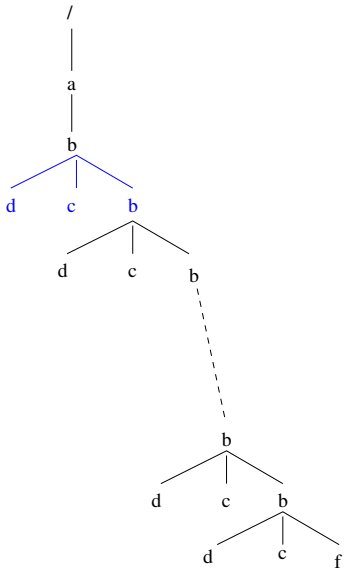
# $\mathcal{RAR}$ versus CXPath

## Proposition 2

1. $CXPath^{+} \not\subseteq \cup\, \mathcal{RAR}$
2. $\cup\, \mathcal{RAR} \not\subseteq CXPath^{+}$

## Preuve par des contre-exemples :

1- /a/(b[d and c])$^{+}$/f) : non exprimable par $\cup\, \mathcal{RAR}$



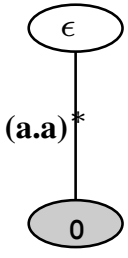
# $\mathcal{RAR}$ versus CXPath

## Proposition 2

1. $CXPath^{+} \not\subseteq \cup\, \mathcal{RAR}$
2. $\cup\, \mathcal{RAR} \not\subseteq CXPath^{+}$

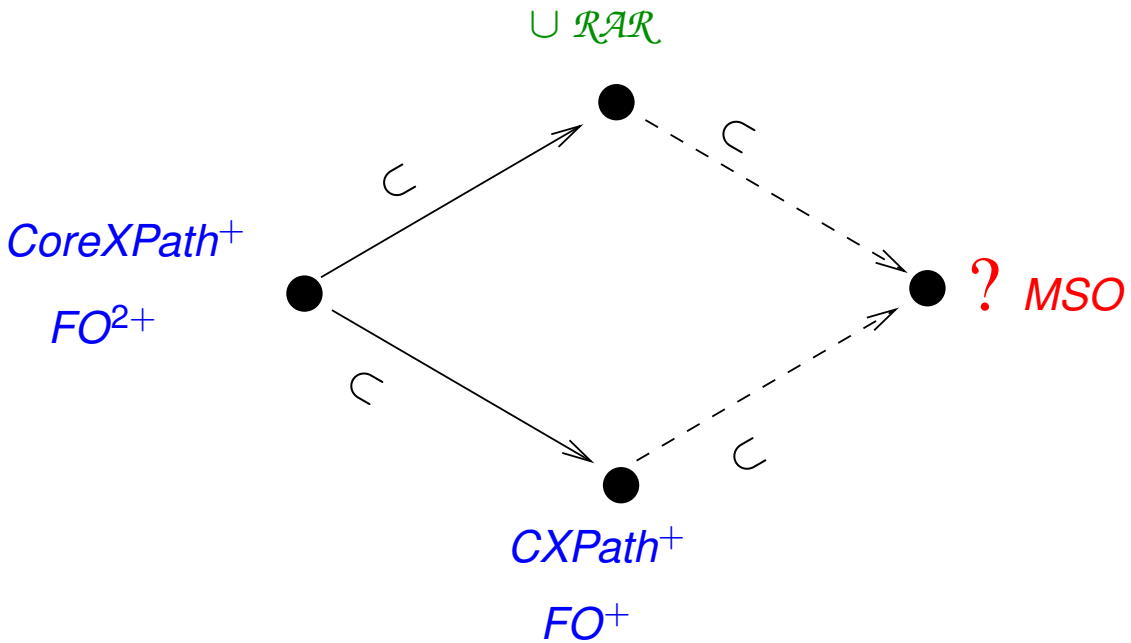
## Preuve par des contre-exemples :

2-$(a.a)^*$ n'est pas sans étoile $\Rightarrow$ non définissable par $FO_{tree}$



# Récapitulatif de Comparaison

## $\mathcal{RAR}$ versus *CoreXPath* et ses extensions



# XPath régulier : RegularXPATH

*RegularXPath* **:** (M. Marx, 2005)

$$
\begin{array}{rcl}
\mathit{prim\_axis} & ::= & \mathit{child} \parallel \mathit{parent} \parallel \mathit{right} \parallel \mathit{left}. \\
P & ::= & \mathit{prim\_axis} \parallel P/P \parallel P \cup P \parallel P^* \parallel ?F \\
F & ::= & \lambda \parallel <P> \parallel \mathit{not}\ F \parallel F\ \mathit{and}\ F \parallel F\ \mathit{or}\ F
\end{array}
$$

- La puissance expressive de ce langage est strictement incluse entre celle des logiques $FO_{tree}$ et *MSO*

**Proposition 3**

Toute requête $\mathcal{RAR}$ monadique $\mathscr{R}$ est exprimable par une expression de chemin de RegularXPath.

# $\mathcal{RAR}$ versus Logique

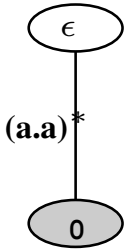
**FOREG** (F. Neven et T. Schwentick, 2000) est l'extension de *FO* par des expressions régulières de chemin verticales et horizontales.

## Proposition 4

Toute requête $\mathcal{RAR}$ $\mathscr{R}$ est exprimable par une propriété définie par une formule $\phi_{\mathscr{R}}$ de **FOREG** ($Eval_{\mathcal{D}}(\mathscr{R})$=$\{ n \in \mathcal{D} / \mathcal{D} \models \phi_{\mathscr{R}}(n)\}$).

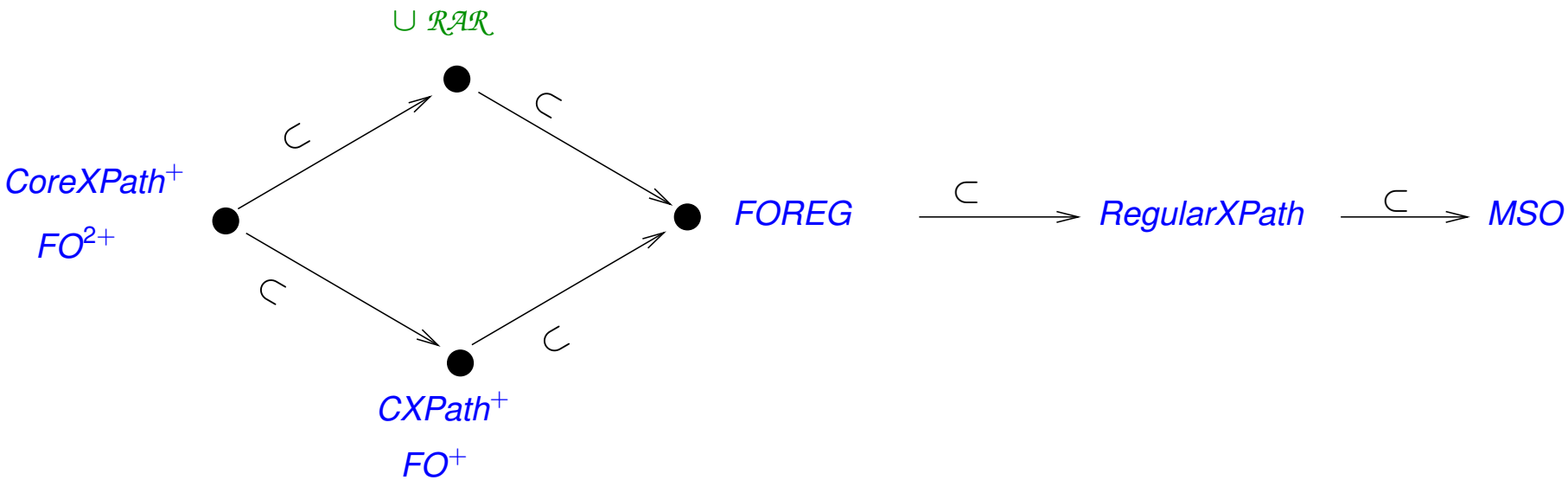
## Exemple

$\phi(x,y)$= $[(\theta_{aa}(s,t)\theta_{aa}(s,t))^*\theta_{aa}(s,t)]^{\downarrow}_{s,t}(x,y)$ avec
$\theta_{aa}(s,t)$=$O_a(s) \wedge O_a(t)$



# Récapitulatif de Comparaison

## $\mathcal{RAR}$ versus *CoreXPath* et ses extensions



# Sommaire

# État de l'art

- Travaux utilisant les sources :
  -[ S. Abiteboul & al, 1998] "*Incremental Maintenance for Materialized Views over Semistructured Data*"
  -[M. Onizuka & al, 2003] "*Incremental Maintenance for Materialized XPath/XSLT Views*"
  -[A. Sawires & al, 2005-2006] "*Incremental maintenance of path-expression views*"
  -[V. Sans 2007] "*Maintenance de vues XML matérialisées à partir de sources web*"
- Travaux n'utilisant pas les sources :
  -[M. Raghavachari and O. Shmueli, 2006] "Conflicting XML Updates"
  -[G. Ghelli & al, 2007] "Commutativity Analysis in XML Update Languages"
  -[M. Benedikt and J. Cheney, 2009] "Schema-Based Independence Analysis for XML Updates"
  -[N. Bidoit & al, 2010] "Detecting XML Query-Update Independence"

# État de l'art

- Travaux utilisant les sources :
  -[ S. Abiteboul & al, 1998] "*Incremental Maintenance for Materialized Views over Semistructured Data*"
  -[M. Onizuka & al, 2003] "*Incremental Maintenance for Materialized XPath/XSLT Views*"
  -[A. Sawires & al, 2005-2006] "*Incremental maintenance of path-expression views*"
  -[V. Sans 2007] "*Maintenance de vues XML matérialisées à partir de sources web*"
- Travaux n'utilisant pas les sources :
  -[M. Raghavachari and O. Shmueli, 2006] "Conflicting XML Updates"
  -[G. Ghelli & al, 2007] "Commutativity Analysis in XML Update Languages"
  -[M. Benedikt and J. Cheney, 2009] "Schema-Based Independence Analysis for XML Updates"
  -[N. Bidoit & al, 2010] "Detecting XML Query-Update Independence"

# Vue, Mise à jour

- Requête de Vue n-aire : Extraction de tuples de nœuds ($E$) + personnalisation ($P$)
  $\Rightarrow \mathcal{V} = P \; o \; E$, dans ce travail $\mathcal{V}$ sera identifiée à $E$

- Mise à jour : sélection de nœuds ($\mathcal{C}$) + Remplacement ($R$)
  $\Rightarrow q = R \; o \; \mathcal{C}$

# Vue, Mise à jour

- Requête de Vue n-aire : Extraction de tuples de nœuds ($E$) + personnalisation ($P$)
  $\Rightarrow \mathcal{V} = P \circ E$, dans ce travail $\mathcal{V}$ sera identifiée à $E$

- Mise à jour : sélection de nœuds ($\mathcal{C}$) + Remplacement ($R$)
  $\Rightarrow q = R \circ \mathcal{C}$

**Remarque**

$\mathcal{C}$ et $E$ : processus commun de sélection de tuples de nœuds

# Classe de Mises à jour

- Une mise à jour $q = R \; o \; \mathcal{C}$ où $\mathcal{C}$ sélectionne les nœuds à modifier.
- $\mathcal{C}$ représente une classe de mises à jour : toutes les mises à jour qui modifient les nœuds sélectionnés par $\mathcal{C}$.

# Classe de Mises à jour

- Une mise à jour $q = R \; o \; \mathcal{C}$ où $\mathcal{C}$ sélectionne les nœuds à modifier.
- $\mathcal{C}$ représente une classe de mises à jour : toutes les mises à jour qui modifient les nœuds sélectionnés par $\mathcal{C}$.

## Exemple

1. "Modifier les auteurs d'articles en y ajoutant un N° de téléphone (Tel)"
2. "Modifier les auteurs d'articles en changeant Nom en (NomFamille, Prénom)".

# Indépendance : définition

**Définition 1 : impact**

Une mise à jour $q$ impacte une vue $\mathcal{V}$ sur $\mathcal{D}$ ssi
$\mathcal{V}(\mathcal{D}) \neq \mathcal{V}(q(\mathcal{D}))$

**Définition 2 : indépendance**

$\mathcal{V}$ est indépendante par rapport à la classe $\mathcal{C}$ ssi
$\forall\ \mathcal{D}, \forall\ q \in \mathcal{C}$, $q$ n'impacte pas $\mathcal{V}$ sur $\mathcal{D}$

**La problématique est alors :**

Étant données une vue $\mathcal{V}$ et une classe de mises à jour $\mathcal{C}$,
$\mathcal{V}$ est-elle indépendante par rapport à $\mathcal{C}$ ?

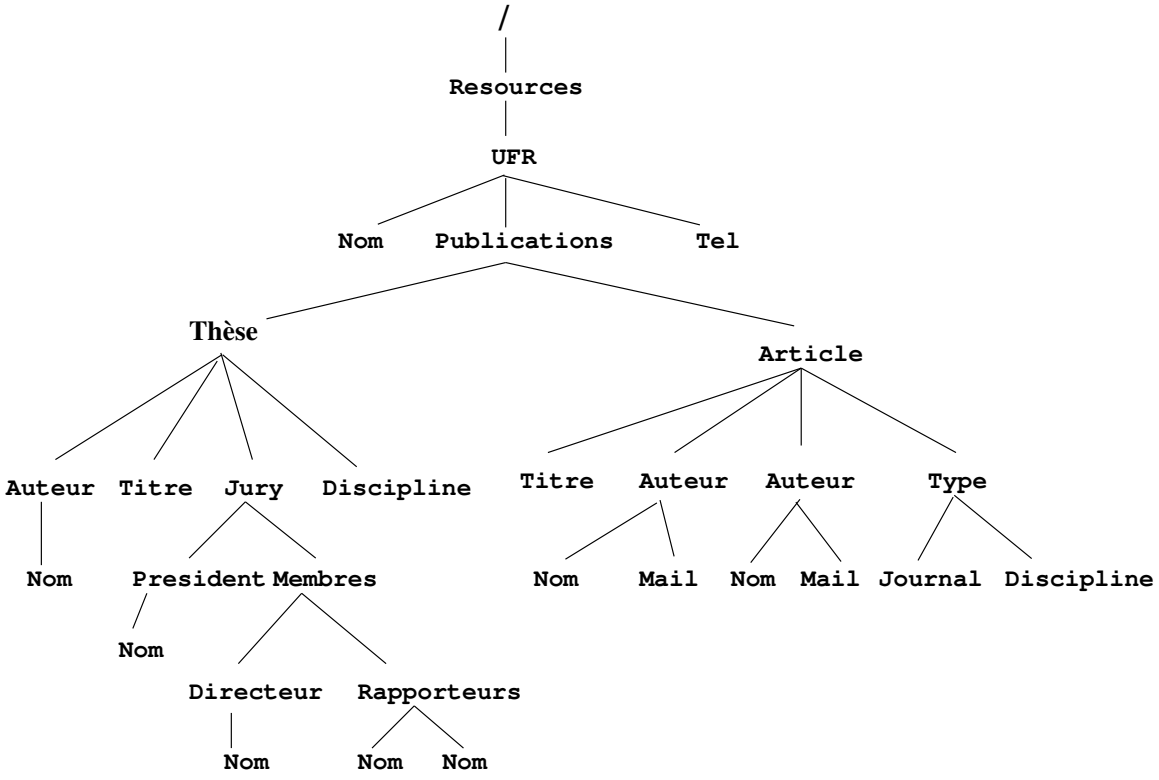
# Exemple d'indépendance

**Soit la vue $\mathcal{V}$ :**

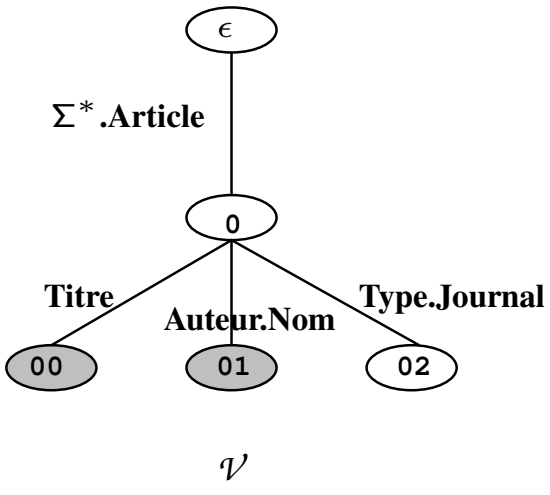
"Donner les (Titre, Nom d'auteur) des articles publiés dans un journal"

**Et la classe de mises à jour : $\mathcal{C}_1$**

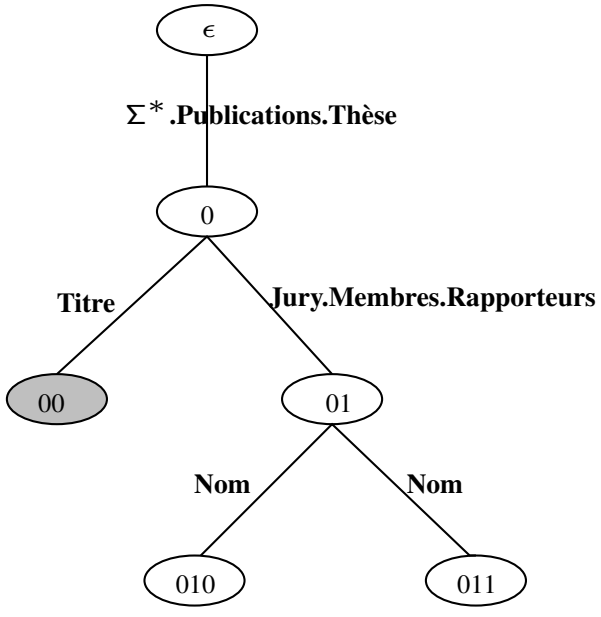
"Mettre à jour les titres des thèses qui ont au moins deux rapporteurs"



# Exemple d'indépendance




$\mathcal{V}$




$\mathcal{C}_1$

Il n'existe pas de mise à jour de $\mathcal{C}_1$ qui impacte $\mathcal{V}$

$\Rightarrow$ Indépendance de $\mathcal{V}$ par rapport à $\mathcal{C}_1$

# Exemple de dépendance

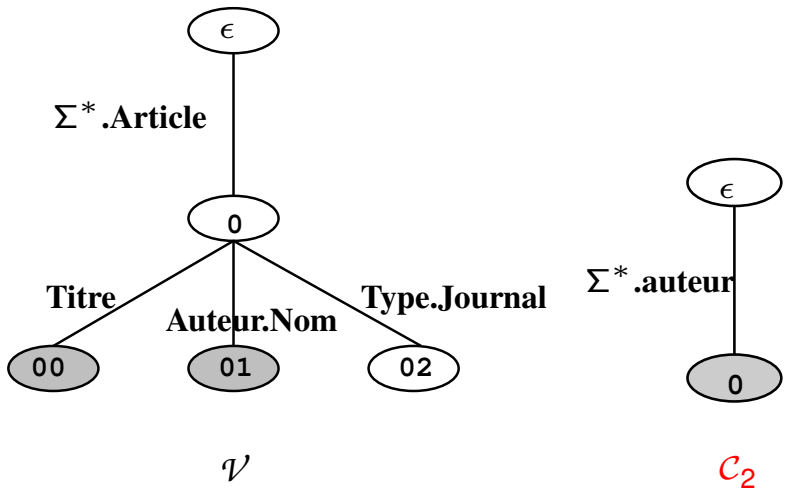
Avec la même requête $\mathcal{V}$

**Mise à jour :** $q_1 \in \mathcal{C}_2$

"Modifier les auteurs d'articles en y ajoutant un N° de téléphone (Tel)"

**Mise à jour :** $q_2 \in \mathcal{C}_2$

"Modifier les auteurs d'articles en changeant Nom en (NomFamille, Prénom)"




$\mathcal{V}$ $\mathcal{C}_2$

# Exemple de dépendance

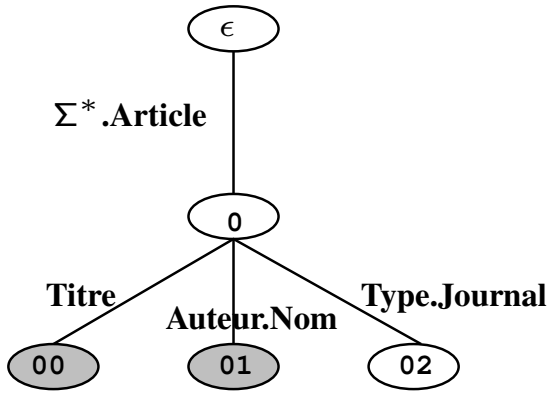
Avec la même requête $\mathcal{V}$

**Mise à jour : $q_1 \in \mathcal{C}_2$**

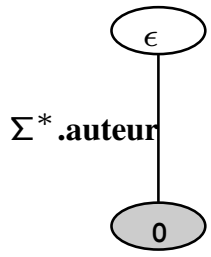
"Modifier les auteurs d'articles en y ajoutant un N° de téléphone (Tel)"

**Mise à jour : $q_2 \in \mathcal{C}_2$**

"Modifier les auteurs d'articles en changeant Nom en (NomFamille, Prénom)"




$\mathcal{V}$




$\mathcal{C}_2$

$q_1$ :pas d'impact mais $q_2$ : impact . $\Rightarrow$ dépendance de $\mathcal{V}$ par rapport à $\mathcal{C}_2$

# Indépendance par rapport à un schéma

Notons valid($\mathcal{S}c$) l'ensemble des documents valides par rapport à $\mathcal{S}c$

**Indépendance par rapport à un schéma**

Une vue $\mathcal{V}$ est indépendante par rapport à une classe de mises à jour $\mathcal{C}$ dans le contexte $\mathcal{S}c$ ssi $\forall\ \mathcal{D} \in$ valid($\mathcal{S}c$), $\forall\ q \in \mathcal{C}$ avec $q(\mathcal{D}) \in$ valid($\mathcal{S}c$), $q$ n'impacte pas $\mathcal{V}$ sur $\mathcal{D}$

**La problématique devient :**

Etant données une vue $\mathcal{V}$, une classe de mises à jour $\mathcal{C}$ et un schéma $\mathcal{S}c$, $\mathcal{V}$ est-elle indépendante par rapport à $\mathcal{C}$ dans le contexte $\mathcal{S}c$ ?

# Résultats sur la dépendance

**Proposition 1**

Décider si $\mathcal{V} = (\mathscr{T}_{\mathcal{V}}, \vec{s_{\mathcal{V}}})$ est indépendante par rapport à $\mathcal{C} = (\mathscr{T}_{\mathcal{C}}, \vec{s_{\mathcal{C}}})$ est un problème PSPACE-difficile.

Preuve réduire le problème d'inclusion des expressions régulières qui est PSPACE-difficile au problème de la dépendance.

# Analyse d'indépendance

Dans la suite nous exhibons une condition suffisante testable en temps polynomial d'indépendance.

# Analyse d'indépendance

Dans la suite nous exhibons une condition suffisante testable en temps polynomial d'indépendance.

Une vue $\mathcal{V}$ est dépendante d'une classe de mises à jour $\mathcal{C}$ dans le contexte $\mathcal{S}c$ ssi
$\exists\, \mathcal{D} \in \textbf{valid}(\mathcal{S}c), \exists q \in \mathcal{C}$ *tel que* $q(\mathcal{D}) \in \textbf{valid}(\mathcal{S}c)$ et
$\mathcal{V}(q(\mathcal{D})) \neq \mathcal{V}(\mathcal{D})$

# Analyse d'indépendance

Dans la suite nous exhibons une condition suffisante testable en temps polynomial d'indépendance.

Une vue $\mathcal{V}$ est dépendante d'une classe de mises à jour $\mathcal{C}$ dans le contexte $\mathcal{S}c$ ssi
$\exists\, \mathcal{D} \in$ **valid**$(\mathcal{S}c)$, $\exists q \in \mathcal{C}$ *tel que* $q(\mathcal{D}) \in$ **valid**$(\mathcal{S}c)$ et
$\mathcal{V}(q(\mathcal{D})) \neq \mathcal{V}(\mathcal{D})$
i.e

1. $\vec{n} \in \mathcal{V}(\mathcal{D})$ *et* $\vec{n} \notin \mathcal{V}(q(\mathcal{D}))$

# Analyse d'indépendance

Dans la suite nous exhibons une condition suffisante testable en temps polynomial d'indépendance.

Une vue $\mathcal{V}$ est dépendante d'une classe de mises à jour $\mathcal{C}$ dans le contexte $\mathcal{S}c$ ssi
$\exists\, \mathcal{D} \in$ **valid**$(\mathcal{S}c)$, $\exists q \in \mathcal{C}$ *tel que* $q(\mathcal{D}) \in$ **valid**$(\mathcal{S}c)$ et
$\mathcal{V}(q(\mathcal{D})) \neq \mathcal{V}(\mathcal{D})$
i.e

1. $\vec{n} \in \mathcal{V}(\mathcal{D})$ *et* $\vec{n} \notin \mathcal{V}(q(\mathcal{D}))$
2. ou $\vec{n} \notin \mathcal{V}(\mathcal{D})$ *et* $\vec{n} \in \mathcal{V}(q(\mathcal{D}))$

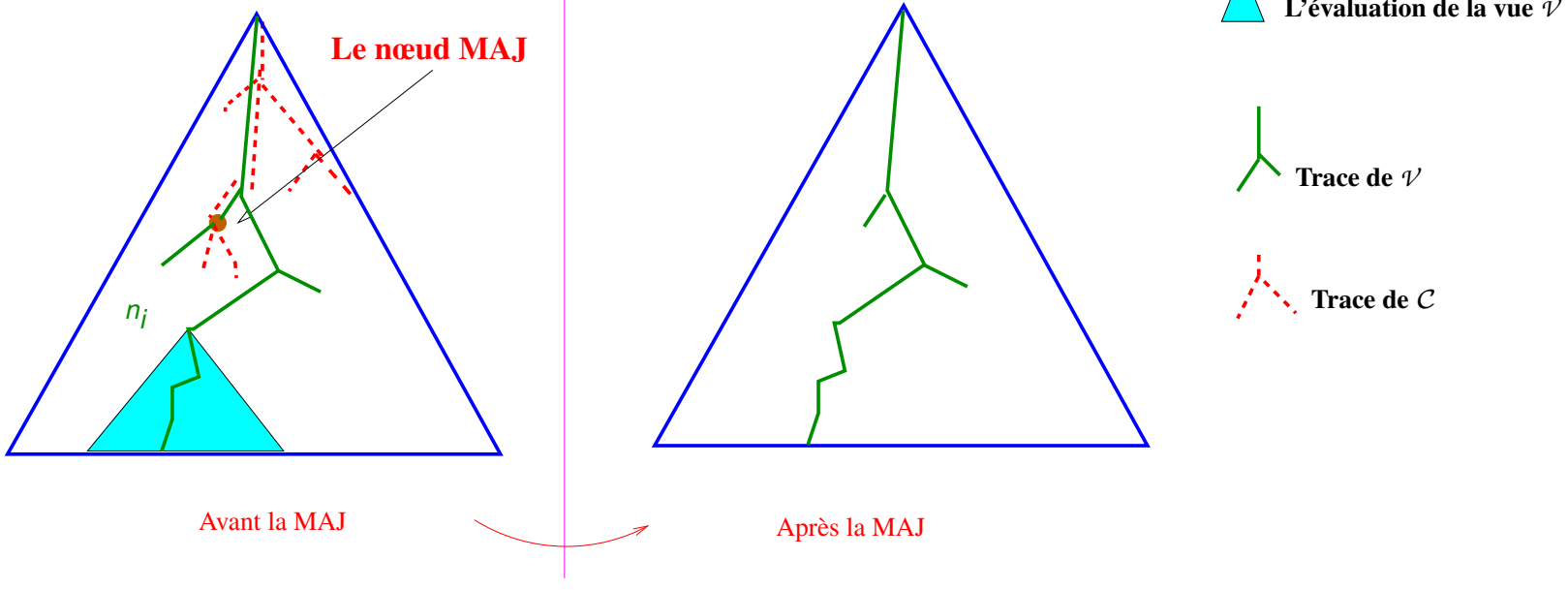
# Analyse d'indépendance

1- $\vec{n} \in \mathcal{V}(\mathcal{D})$ *et* $\vec{n} \notin \mathcal{V}(q(\mathcal{D})) \Rightarrow$

# Analyse d'indépendance

1- $\vec{n} \in \mathcal{V}(\mathcal{D})$ *et* $\vec{n} \notin \mathcal{V}(q(\mathcal{D})) \Rightarrow$

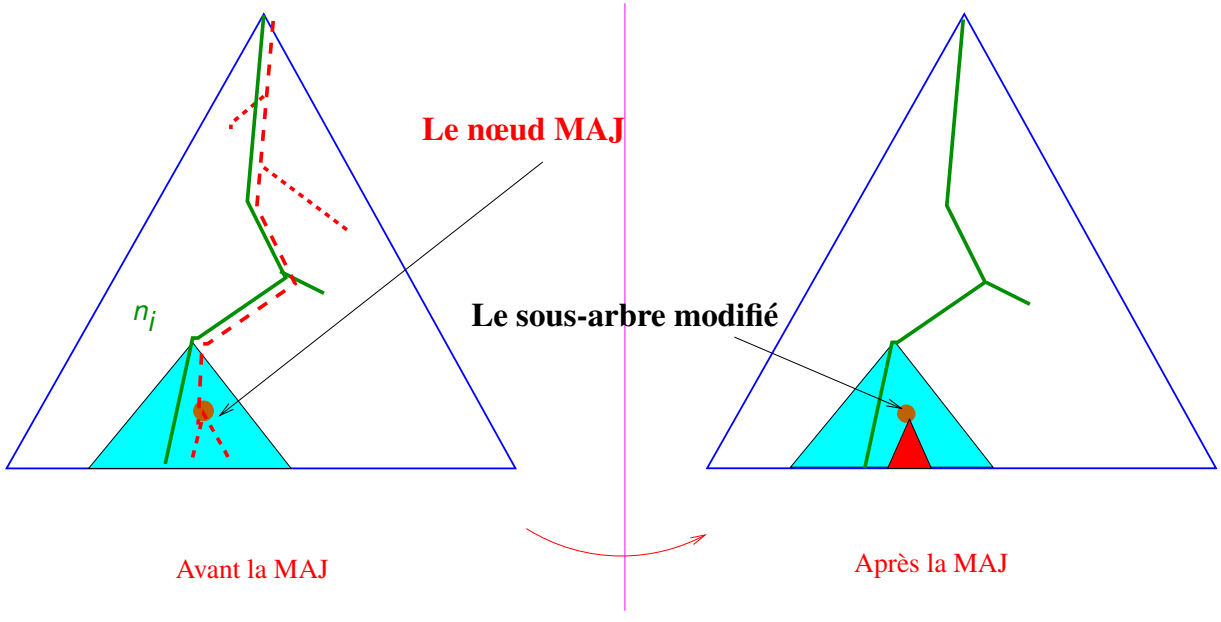
**Cas i**



# Analyse d'indépendance

1- $\vec{n} \in \mathcal{V}(\mathcal{D})$ *et* $\vec{n} \notin \mathcal{V}(q(\mathcal{D})) \Rightarrow$

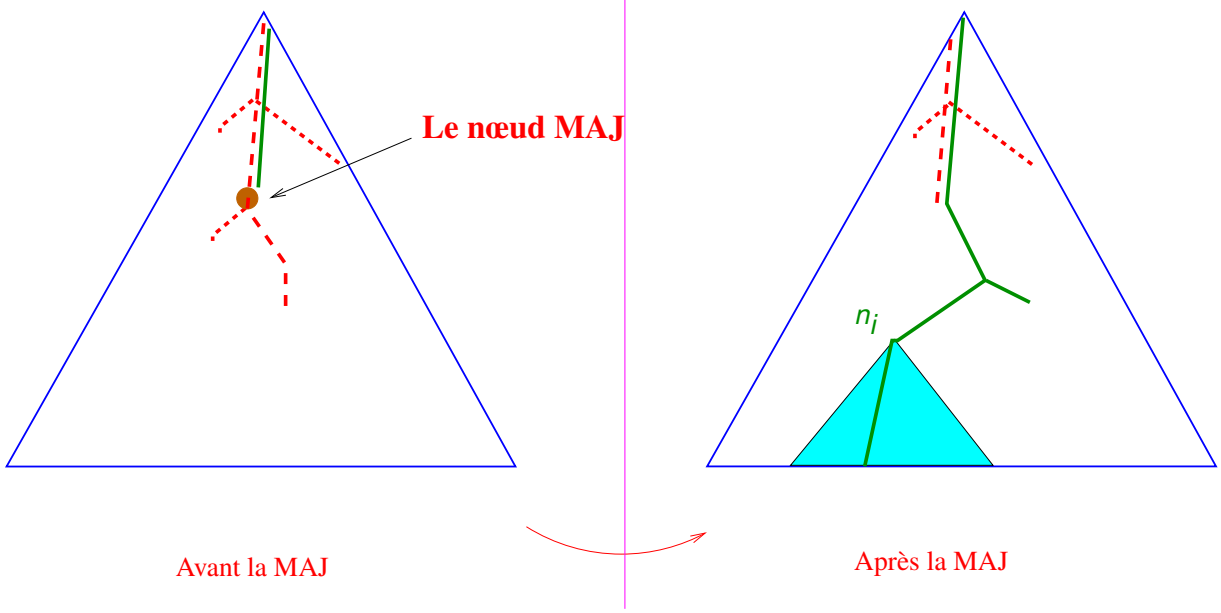
**Cas ii**



# Analyse d'indépendance

2- $\vec{n} \notin \mathcal{V}(\mathcal{D})$ *et* $\vec{n} \in \mathcal{V}(q(\mathcal{D})) \Rightarrow$

**Cas iii**



# Analyse d'indépendance

**plus formellement, nous écrivons :**

- (1) $\Longrightarrow \hat{p_c}(s_c) \in \mathcal{N}(trace_{p_v}(\mathcal{V}, \mathcal{D})) \cup \mathcal{N}(\mathcal{V}_{p_v}(\mathcal{D}))$
- (2) $\Longrightarrow \hat{p_c}(s_c) \in \mathcal{N}(trace_{p_v}(\mathcal{V}, q(\mathcal{D})))$

$p_v$ : un plongement de $\mathcal{V}$
$\hat{p_c}$ : un plongement partiel de $\mathcal{C}$ (plongement de $\widehat{\mathscr{R}} = (\widehat{\mathscr{T}}, \vec{s})$ dans $\mathcal{D}$).

et $\mathcal{N}(\vec{n}) = \cup_{i=1}^{n} \mathcal{N}(\mathcal{D}_i)$ est l'ensemble des nœuds de $\mathcal{D}_i$ avec $\vec{n} = (\mathcal{D}_1, ..., \mathcal{D}_n)$ un n-uplet d'arbres.

# Un langage régulier pour tester la dépendance

**Définition**

Soit $\mathcal{L}$ l'ensemble des arbres $\mathcal{D}$ vérifiant :

(i) $\mathcal{D} \in \texttt{valide}(\mathcal{S}c)$

(ii) il existe une trace $\texttt{trace}_{p_v}(\mathcal{V}, \mathcal{D})$ de $\mathcal{V}$ dans $\mathcal{D}$ selon un plongement $p_v$ et

il existe une trace partielle $\texttt{trace}_{\hat{p}_c}(\widehat{\mathcal{C}}, \mathcal{D})$ de $\mathcal{C}$ dans $\mathcal{D}$ selon un plongement partiel $\hat{p}_c$ et un nœud $s_{\mathcal{C}}$ de $\vec{s_{\mathcal{C}}}$ tel que $\hat{p}_c(s_{\mathcal{C}}) \in \mathcal{N}(\texttt{trace}_{p_v}(\mathcal{V}, \mathcal{D})) \cup \mathcal{N}(\mathcal{V}_{p_v}(\mathcal{D}))$.

# Automate d'arbre et indépendance

Soit $\mathcal{V}$ une requête de vue, et $\mathcal{C}$ une classe de mises à jour

**Proposition 2**

Si $\mathcal{L} = \varnothing$ alors $\mathcal{V}$ est indépendante par rapport $\mathcal{C}$ dans le contexte $\mathcal{S}c$

$\Longrightarrow \mathcal{L} = \varnothing$ est une condition suffisante pour l'indépendance de $\mathcal{V}$ par rapport $\mathcal{C}$

# Cas particulier : problème d'indépendance polynomial

Dans le cas où

1. les nœuds de sélection de la requête de vue $\mathcal{V} = (\mathscr{T}_V, \vec{s}_V)$ vérifient : "tout chemin de la racine à une feuille de $\mathscr{T}_V$ contient au moins un nœud de $\vec{s}_V$ ".
2. et les nœuds de mises à jour sont tous des feuilles.

## Cas particulier : problème d'indépendance polynomial

Dans le cas où

1. les nœuds de sélection de la requête de vue $\mathcal{V} = (\mathcal{T}_V, \vec{s}_V)$ vérifient : "tout chemin de la racine à une feuille de $\mathcal{T}_V$ contient au moins un nœud de $\vec{s}_V$ ".
2. et les nœuds de mises à jour sont tous des feuilles.

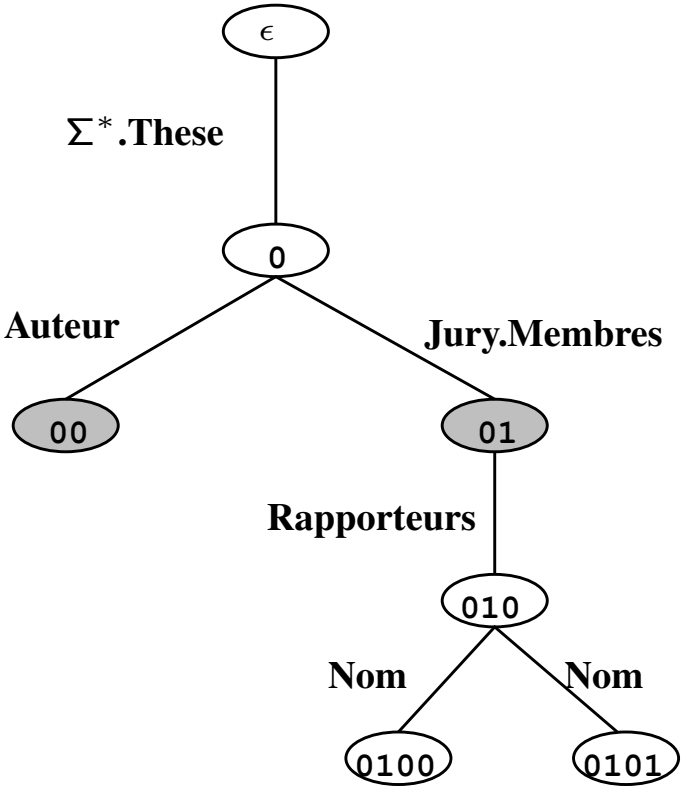
On a :

**Proposition 3**

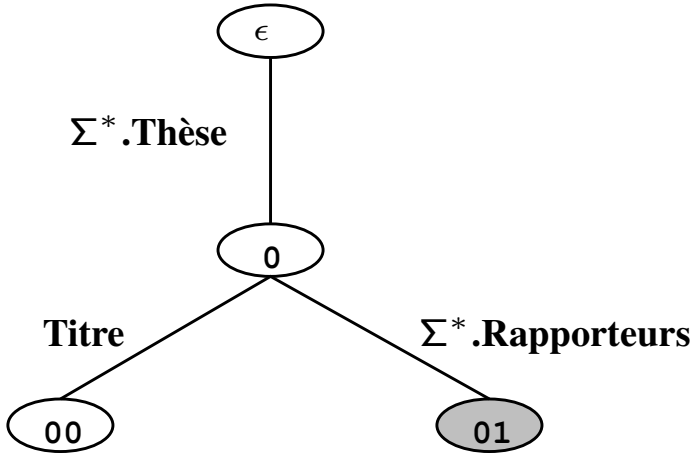
$\mathcal{V}$ est indépendante de la classe $\mathcal{C}$ si et seulement si $\mathcal{L}$ est vide.

# Cas particulier : problème d'indépendance polynomial

**Exemple :** "Donner les couples (Auteur, Membres) des thèses ayant au moins deux rapporteurs"




$\mathcal{V}$




$\mathcal{C}$

# Construction de $\mathcal{A}$

Pour vérifier la vacuité de $\mathcal{L}$, nous définissons l'automate d'arbre $\mathcal{A}$ qui reconnaît tous les arbres de $\mathcal{L}$.

# Construction de $\mathcal{A}$

Pour vérifier la vacuité de $\mathcal{L}$, nous définissons l'automate d'arbre $\mathcal{A}$ qui reconnaît tous les arbres de $\mathcal{L}$.

## Étapes de construction de $\mathcal{A}$ :

1. Construire l'automate $\mathcal{A}_{\widehat{c}}$ qui reconnaît l'ensemble des arbres contenant au moins une trace de $\widehat{\mathcal{C}}$

# Construction de $\mathcal{A}$

Pour vérifier la vacuité de $\mathcal{L}$, nous définissons l'automate d'arbre $\mathcal{A}$ qui reconnaît tous les arbres de $\mathcal{L}$.

## Étapes de construction de $\mathcal{A}$ :

1. Construire l'automate $\mathcal{A}_{\widehat{c}}$ qui reconnaît l'ensemble des arbres contenant au moins une trace de $\widehat{\mathcal{C}}$
2. Construire l'automate $\mathcal{A}_v$ qui reconnaît l'ensemble des arbres contenant au moins une trace de $\mathcal{V}$ + le modifier en $\bar{\mathcal{A}}_v$ en changeant qlq transitions pour identifier les nœuds descendants des nœuds de sélection.

# Construction de $\mathcal{A}$

Pour vérifier la vacuité de $\mathcal{L}$, nous définissons l'automate d'arbre $\mathcal{A}$ qui reconnaît tous les arbres de $\mathcal{L}$.

## Étapes de construction de $\mathcal{A}$ :

1. Construire l'automate $\mathcal{A}_{\widehat{c}}$ qui reconnaît l'ensemble des arbres contenant au moins une trace de $\widehat{\mathcal{C}}$
2. Construire l'automate $\mathcal{A}_v$ qui reconnaît l'ensemble des arbres contenant au moins une trace de $\mathcal{V}$ + le modifier en $\bar{\mathcal{A}}_v$ en changeant qlq transitions pour identifier les nœuds descendants des nœuds de sélection.
3. Définir $\mathcal{B} = \mathcal{A}_{\widehat{c}} \times \bar{\mathcal{A}}_v$ + le changer pour vérifier la condition $\exists\, p_v, \hat{p_c}$ tel que $\hat{p_c}(s_{\mathcal{C}}) \in \mathcal{N}(\texttt{trace}_{p_v}(\mathcal{V}, \mathcal{D})) \cup \mathcal{N}(\mathcal{V}_{p_v}(\mathcal{D}))$

# Construction de $\mathcal{A}$

Pour vérifier la vacuité de $\mathcal{L}$, nous définissons l'automate d'arbre $\mathcal{A}$ qui reconnaît tous les arbres de $\mathcal{L}$.

## Étapes de construction de $\mathcal{A}$ :

1. Construire l'automate $\mathcal{A}_{\widehat{c}}$ qui reconnaît l'ensemble des arbres contenant au moins une trace de $\widehat{\mathcal{C}}$
2. Construire l'automate $\mathcal{A}_v$ qui reconnaît l'ensemble des arbres contenant au moins une trace de $\mathcal{V}$ + le modifier en $\bar{\mathcal{A}}_v$ en changeant qlq transitions pour identifier les nœuds descendants des nœuds de sélection.
3. Définir $\mathcal{B} = \mathcal{A}_{\widehat{c}} \times \bar{\mathcal{A}}_v$ + le changer pour vérifier la condition $\exists\, p_v, \hat{p_c}$ tel que $\hat{p_c}(s_{\mathcal{C}}) \in \mathcal{N}(\texttt{trace}_{p_v}(\mathcal{V}, \mathcal{D})) \cup \mathcal{N}(\mathcal{V}_{p_v}(\mathcal{D}))$
4. Définir $\mathcal{A} = \mathcal{B} \times \mathcal{A}_s$ où $\mathcal{A}_s$ représente l'automate du schéma $\mathcal{S}c$

# Test polynomial du critère d'indépendance

**Proposition 4 : Taille de l'automate $\mathcal{A}$**

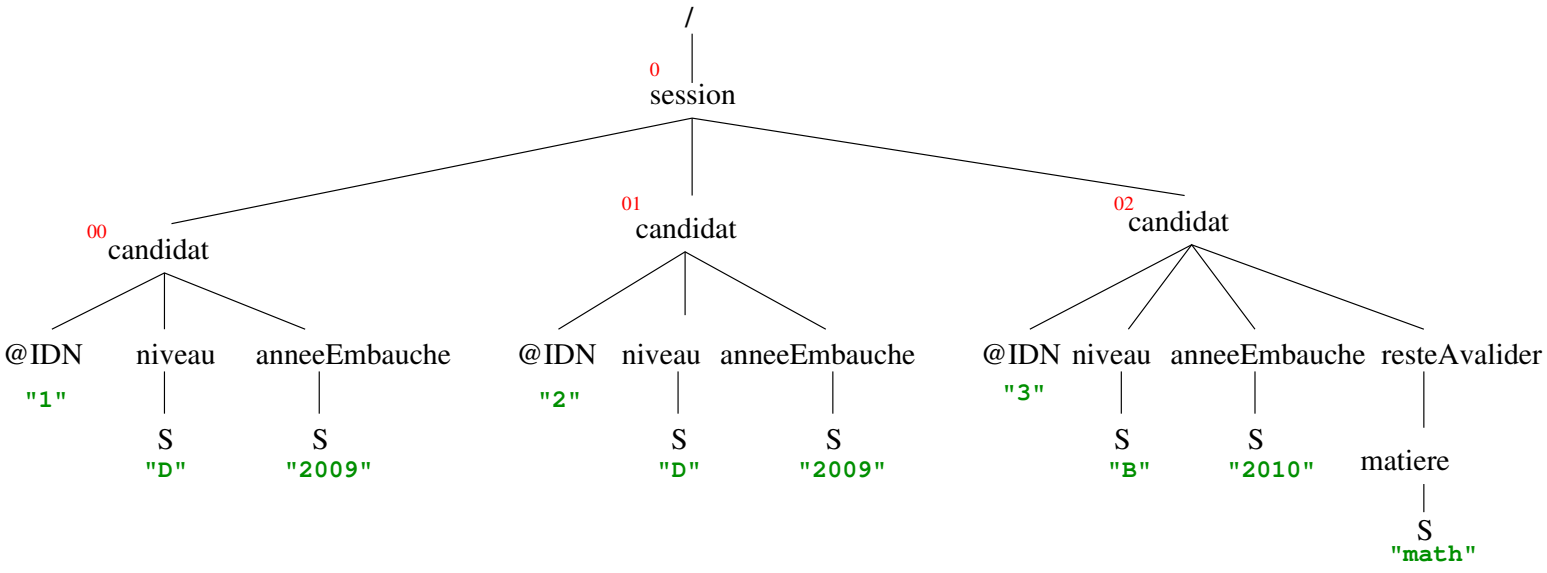
Si $a_c$ et $a_v$ désignent les arités maximales de $\mathcal{C}$ et $\mathcal{V}$ alors il existe une constante K tel que $|\mathcal{A}| \leq K|\Sigma|^3|\mathcal{A}_{\mathcal{S}c}||\mathcal{C}||\mathcal{V}|a_c a_v$.

**Proposition 5 : Test de vacuité**

Le test de vacuité de $L(\mathcal{A})$ est polynomial (en $O(|\Sigma|^6|\mathcal{A}_{\mathcal{S}c}|^2|\mathcal{C}|^2|\mathcal{V}|^2 a_c^2 a_v^2)$ ) .

# Sommaire

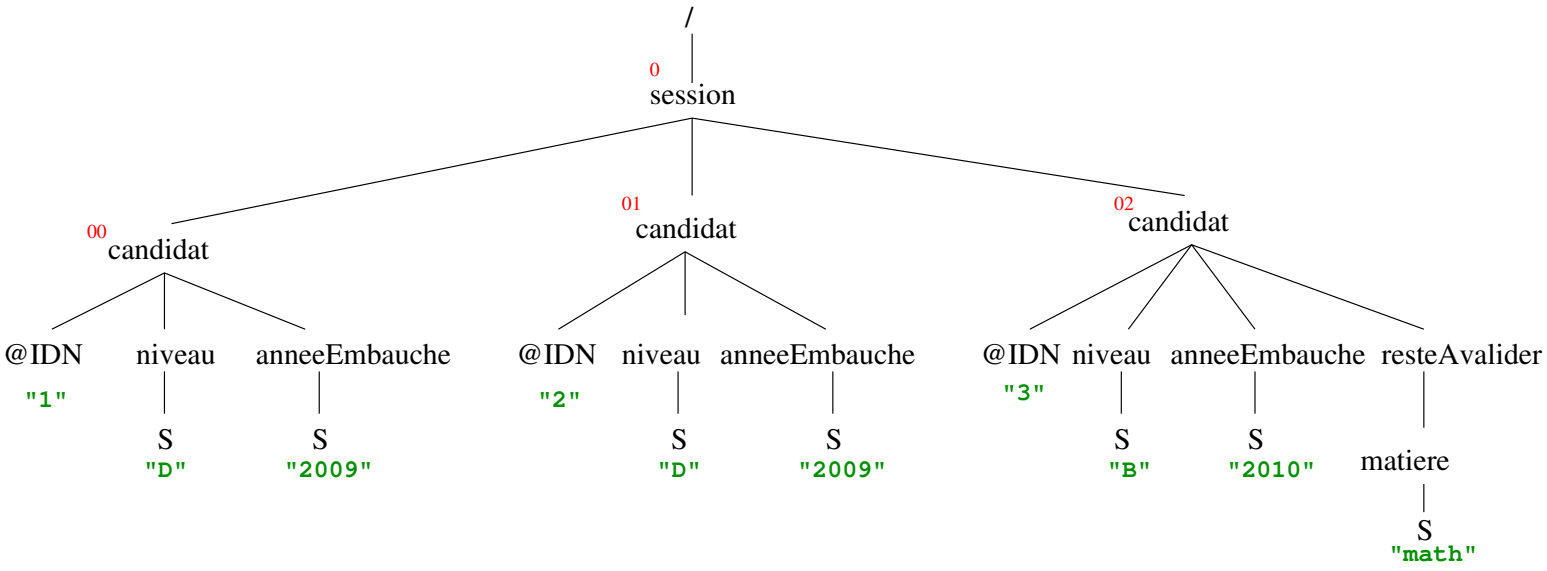
# Introduction : exemple




## $\mathcal{D}$ un document XML, $\mathcal{D}$=($D$, $\lambda$, $val$)

- $D \subset \mathbb{N}^*$ :un domaine d'arbre noté par $\mathcal{N}(\mathcal{D})$
- $\lambda : D \rightarrow \Sigma = El \cup A \cup \{S\}$
- $val : D \rightarrow D \cup I^*$

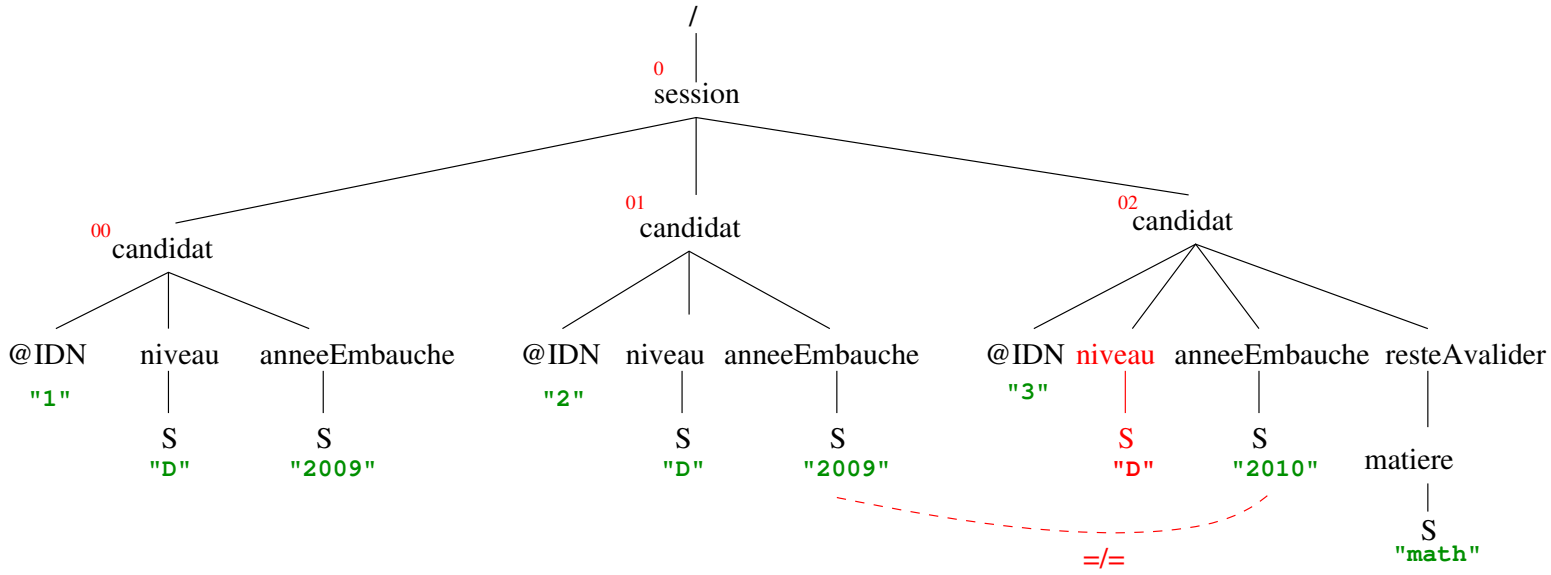
# Introduction : exemple




## Une dépendance fonctionnelle (satisfaite avant la MAJ)

"Deux candidats qui ont le même niveau académique et qui ont trouvé du travail, ont été embauchés la même année dans leur premier job".

# Introduction : exemple



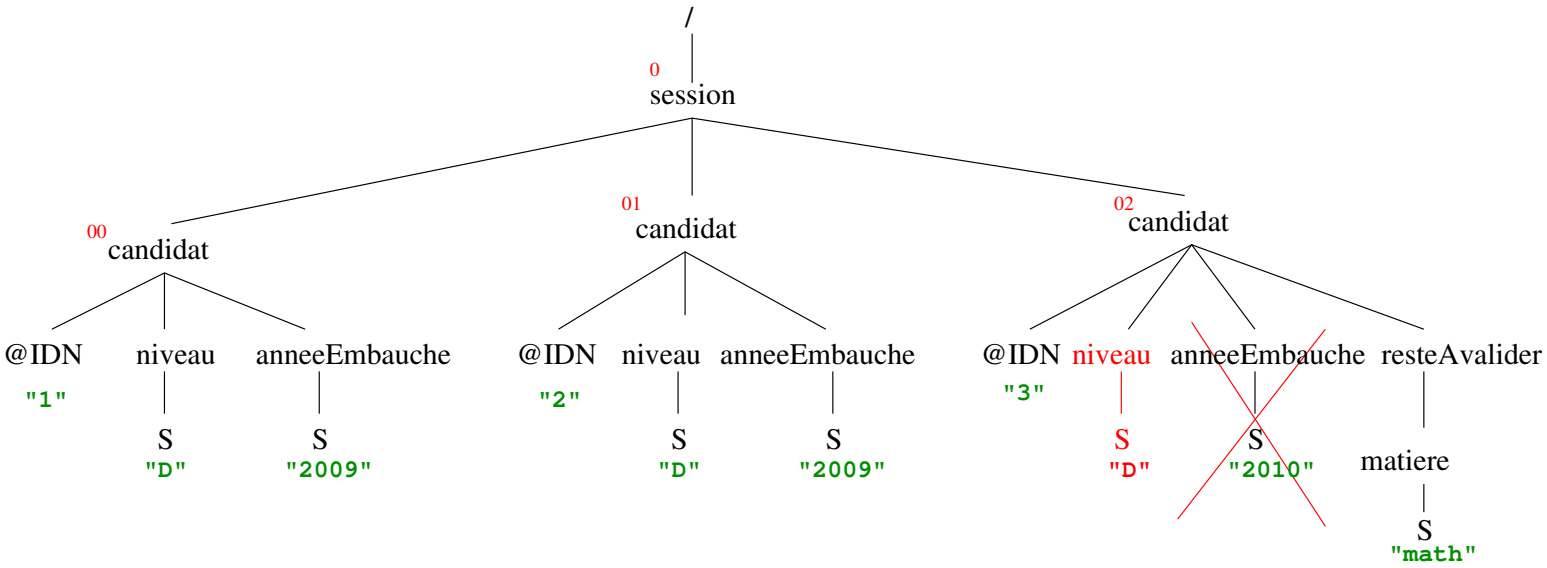

## Une MAJ

"Mettre à jour le niveau de tous les candidats auxquels il reste des épreuves à valider"

## La dépendance fonctionnelle (violée après la MAJ)

"Deux candidats qui ont le même niveau académique et qui ont trouvé du travail, ont été embauchés la même année dans leur premier job".

# Introduction : exemple




**Schema** $\mathcal{S}c$

candidat :(niveau,(anneEmbauche|resteAvalider))

La dépendance fonctionnelle : "Deux candidats qui ont le même niveau académique et qui ont trouvé du travail, ont été embauchés la même année dans leur premier job" reste satisfaite après la MAJ

# Travaux reliés

- Travaux sur la spécification :
  -[ W. Fan & al, 2000 - 2002] "*Integrity constraints for XML*"
  -[P. Buneman & al, 2003] "*Reasoning about Keys for XML*"
  -[S. Hartmann & S. Link, 2003] "*More Functional Dependencies for XML*"
  -[M. Arenas & L. Libkin, 2004] "*A normal form for XML documents*"
- Travaux traitant les mises à jour :
  -[Y. Chen & al, 2002] "XKvalidator : a constraint validator for XML"
  -[M. A. Lima, 2007] "Maintenance incrémentale des contraintes d'intégrité en XML"

# Travaux reliés

- Travaux sur la spécification :
  -[ W. Fan & al, 2000 - 2002] "*Integrity constraints for XML*"
  -[P. Buneman & al, 2003] "*Reasoning about Keys for XML*"
  -[S. Hartmann & S. Link, 2003] "*More Functional Dependencies for XML*"
  -[M. Arenas & L. Libkin, 2004] "*A normal form for XML documents*"
- Travaux traitant les mises à jour :
  -[Y. Chen & al, 2002] "XKvalidator : a constraint validator for XML"
  -[M. A. Lima, 2007] "Maintenance incrémentale des contraintes d'intégrité en XML"

# Expression des dépendances fonctionnelles

Le modèle le plus utilisé est le modèle des chemins linéaires simples.

**Exemple :**

"Deux candidats qui ont le même niveau académique et qui ont trouvé du travail, ont été embauchés la même année dans leur premier job".

(/session ,( {candidat/niveau} $\rightarrow$ candidat/anneeEmbauche))

# Expression des dépendances fonctionnelles

Le modèle le plus utilisé est le modèle des chemins linéaires simples.

**Exemple :**

"Deux candidats qui ont le même niveau académique et qui ont trouvé du travail, ont été embauchés la même année dans leur premier job".

$$(\underbrace{/session}_{\text{contexte}},(\underbrace{\{candidat/niveau\}}_{\text{condition}} \rightarrow \underbrace{candidat/anneeEmbauche}_{\text{cible}}))$$

# Expression des dépendances fonctionnelles

Le modèle le plus utilisé est le modèle des chemins linéaires simples.

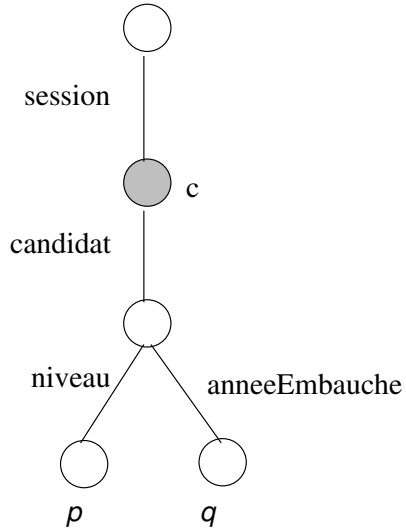
**Exemple :**

"Deux candidats qui ont le même niveau académique et qui ont trouvé du travail, ont été embauchés la même année dans leur premier job".

$$(\underbrace{/session}_{\text{contexte}},(\underbrace{\{candidat/niveau\}}_{\text{condition}} \rightarrow \underbrace{candidat/anneeEmbauche}_{\text{cible}}))$$

**Le modèle choisi :**

REQUÊTES ARBRES RÉGULIÈRES ($\mathcal{RARs}$)



# Modélisation des dfs par les $\mathcal{RAR}s$

**Définition**

*Une XDF-$\mathcal{RAR}$ est une expression $fd = (\mathcal{F}D, c)$ où :*

- $\mathcal{F}D = (\mathcal{T}, \overrightarrow{s} = \{p_1[E_1], p_2[E_2], ..., p_n[E_n], q[E_{n+1}]\})$ *est une $\mathcal{RAR}$.*
  *$p_1, ..., p_n$ and q sont associés à des types d'égalités $E_i \in \{V, N\}$ (i=1,..., n+1)*
- *c (nœud contexte) est un nœud ancêtre des $p_1, p_2, ..., p_n$ (nœuds conditions) et de q (nœud cible)*

V est l'égalité par valeur : ($w_1 \equiv_v w_2 \Leftrightarrow \mathcal{D}(w_1)$ et $\mathcal{D}(w_2)$ ont la même valeur.)

N est l'égalité de nœud : $w_1 \equiv_N w_2$ ssi $w_1 = w_2$

# Satisfaction d'une dépendance fonctionnelle

**Définition**

*Un document $\mathcal{D}$ satisfait la dépendance fonctionnelle $(\mathcal{F}D, c)$ ssi :*

# Satisfaction d'une dépendance fonctionnelle

**Définition**

*Un document $\mathcal{D}$ satisfait la dépendance fonctionnelle $(\mathcal{F}D, c)$ ssi :*

*Si pour deux traces,* $\tau_1 = \mathtt{trace}_{\pi_1}(\mathcal{F}D, \mathcal{D})$ *et* $\tau_2 = \mathtt{trace}_{\pi_2}(\mathcal{F}D, \mathcal{D})$, *avec*

*(a)* $\pi_1(c) =_N \pi_2(c)$

*(b)* $\forall i = 1, ..., n, \pi_1(p_i) =_{E_i} \pi_2(p_i)$,

# Satisfaction d'une dépendance fonctionnelle

**Définition**

*Un document $\mathcal{D}$ satisfait la dépendance fonctionnelle $(\mathcal{F}D, c)$ ssi :*

*Si pour deux traces,* $\tau_1 = \mathtt{trace}_{\pi_1}(\mathcal{F}D, \mathcal{D})$ *et* $\tau_2 = \mathtt{trace}_{\pi_2}(\mathcal{F}D, \mathcal{D})$, *avec*

*(a)* $\pi_1(c) =_N \pi_2(c)$

*(b)* $\forall i = 1, ..., n, \pi_1(p_i) =_{E_i} \pi_2(p_i)$,

*Alors* $\pi_1(q) =_{E_{n+1}} \pi_2(q)$

# Résultat sur l'analyse de dépendance

**Proposition**

Décider si une dépendance fonctionnelle $(\mathcal{F}D, c)$ est indépendante par rapport à une classe de MAJ $\mathcal{C}$ est un problème PSPACE-difficile

# Un critère d'indépendance

## Définition

*Soit $\mathcal{L}$ le langage des documents XML $\mathcal{D}$ satisfaisant :*
*(i)* $\mathcal{D} \in \mathtt{valid}(\mathcal{S}c)$
*(ii)* $\exists\ \tau_{\mathcal{F}D} = trace_\pi(\mathcal{F}D, \mathcal{D})$, *par rapport à un plongement* $\pi$ *de* $\mathcal{F}D$ *dans* $\mathcal{D}$, *et* $\exists\ \tau_{\mathcal{C}^p} = trace_{\pi'}(\mathcal{C}^p, \mathcal{D})$, *par rapport à un plongement partiel* $\pi'$ *de* $\mathcal{C}$ *dans* $\mathcal{D}$, *tel que :*

$$\mathcal{N}(\pi'(\overrightarrow{s}_{\mathcal{C}})) \bigcap (\mathcal{N}(trace_\pi(\mathcal{F}D, \mathcal{D})) \cup \mathcal{N}(\mathcal{F}D_\pi(\mathcal{D}))) \neq \emptyset$$

## Proposition 1

Si $\mathcal{L}$ est vide alors $(\mathcal{F}D, c)$ est indépendante par rapport à $\mathcal{C}$ dans le contexte $\mathcal{S}c$

## Proposition 2

La vacuité du langage $\mathcal{L}$ est testable en temps $O(a_{\mathcal{C}}^2\ a_{\mathcal{F}D}^2 \times |\Sigma|^4 \times |\mathcal{A}_{\mathcal{S}c}|^2 \times |\mathcal{C}|^2 \times |\mathcal{F}D|^2)$

# Conclusion

- Définition d'un langage formel de requêtes $\mathcal{RARS}$.
- Les requêtes $\mathcal{RARS}$ expriment les requêtes de *CoreXPath*$^{+}$.
- Modélisation des Vues, MAJ et DFs par les requêtes $\mathcal{RARS}$.
- Problème de l'indépendance PSPACE-difficile.
- Condition suffisante d'indépendance entre Vues et MAJ.
- Condition suffisante d'indépendance entre DFs et MAJ.

# Perspectives

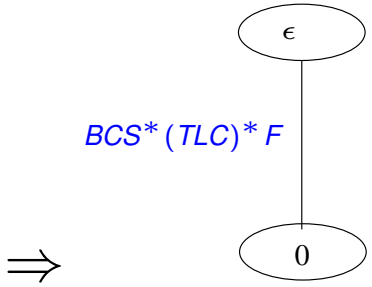
- Évaluation des $\mathcal{RARS}$.
- Étude de sous fragments par simplification des expressions régulières.
- Extension de l'analyse d'indépendance.
- Implémentation du critère d'indépendance.
- $\mathcal{RARS}$ et autres contraintes d'intégrités.

Merci de votre attention

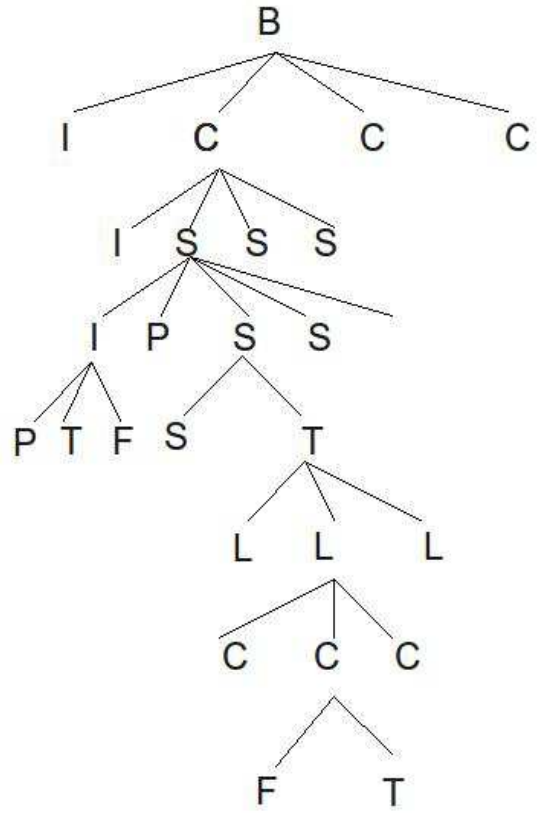
# Exemple d'une RAR

C : chapitre, I : introduction, S : section, T : tableau, L : ligne, C : colonne, F :figure

Exemple de requête :"Extraire toutes les figures des tableaux"




⇒




Book document