Regular tree patterns: a uniform formalism for update queries and functional dependencies in XML

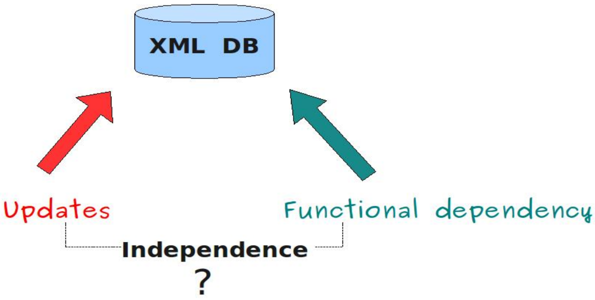
Hicham Idabal[1] Françoise Gire[1]

[1]CRI, Paris1 University, France

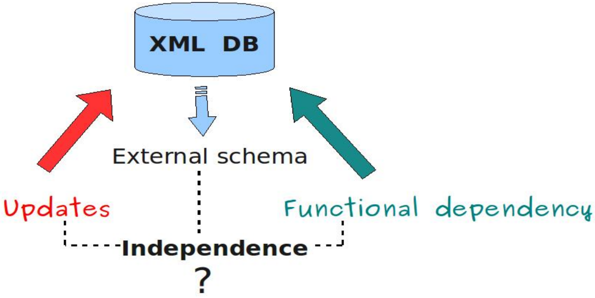
Updates in XML 2010

Outline

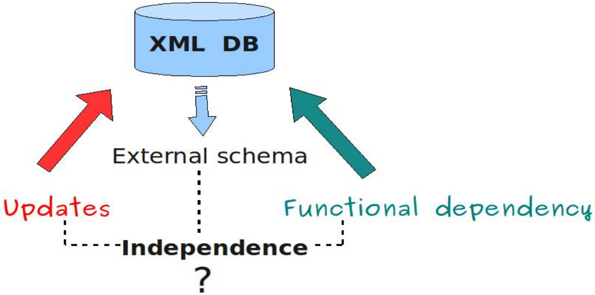
Introduction: the problem of independence



Introduction: the problem of independence



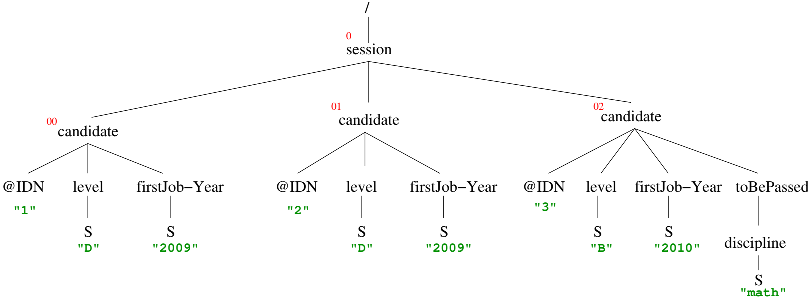
Introduction: the problem of independence



Goal

Detecting independence will help us to avoid a new verification of the functional dependency

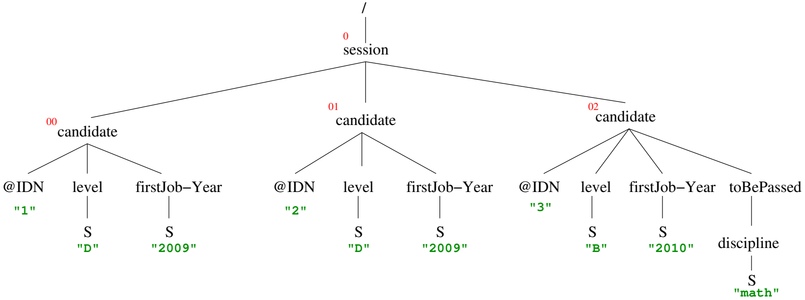
Introduction: example



XML documents $\mathcal{D}$=(D, λ, val)

- $D \subset \mathbb{N}^*$: a tree domain denoted by $\mathcal{N}(\mathcal{D})$
- $\lambda : D \to \Sigma = El \cup A \cup \{S\}$
- val: $D \to D \cup I^*$

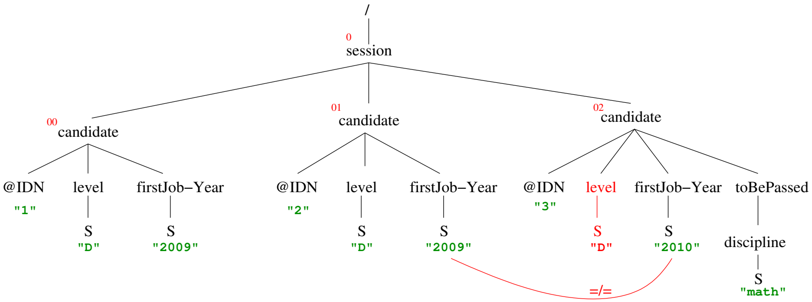
Introduction: example



A functional dependency (satisfied before the update)

“Two candidates with a job and a same academic level, have got their first job the same year.”

Introduction: example



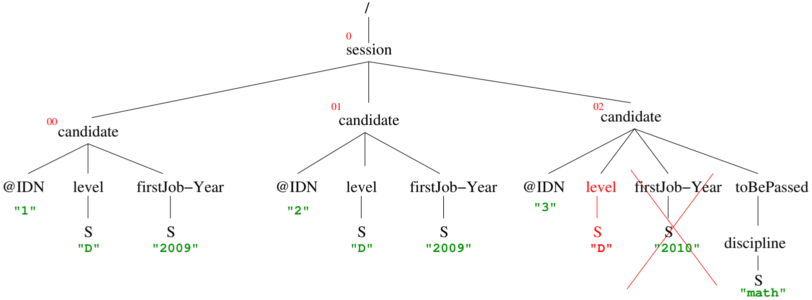
An update

“ Update the level of each candidate having to pass some remaining exams“

A functional dependency (not satisfied after updating)

“Two candidates with a job and a same academic level, have got their first job the same year.”

Introduction: example



Schema :$\mathcal{S}c$

candidate :(level,(firstJob-Year|toBePassed))

The functional dependency “Two candidates with a job and a same academic level, have got their first job the same year” remains satisfied after the update

Related work

- Some works in the area:
  -[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*"
- Similar works using updates:
  -[Y. Chen & al, 2002] "XKvalidator: a constraint validator for XML"
  -[M. A. Lima, 2007] "Maintenance incrémentale des contraintes d'intégrité en XML"

Related work

- Some works in the area:
  -[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*"
- Similar works using updates:
  -[Y. Chen & al, 2002] "XKvalidator: a constraint validator for XML"
  -[M. A. Lima, 2007] "Maintenance incrémentale des contraintes d'intégrité en XML"

Expressing functional dependencies

The most commonly used model is based on simple linear paths.

Example:

"Two candidates with a job and a same academic level, have got their first job the same year"

(/session ,({candidate/level} → candidate/firstJobYear))

Expressing functional dependencies

The most commonly used model is based on simple linear paths.

Example:

"Two candidates with a job and a same academic level, have got their first job the same year"

$$(\underbrace{/session}_{\text{context}},(\underbrace{\{candidate/level\}}_{\text{condition}} \rightarrow \underbrace{candidate/firstJobYear}_{\text{target}}))$$

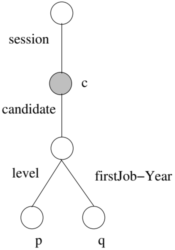
Expressing functional dependencies

The most commonly used model is based on simple linear paths.

Example:

"Two candidates with a job and a same academic level, have got their first job the same year"

$$(\underbrace{/session}_{\text{context}},(\underbrace{\{candidate/level\}}_{\text{condition}} \rightarrow \underbrace{candidate/firstJobYear}_{\text{target}}))$$



Our chosen model :

Regular tree patterns (RTPs)

Regular tree pattern: the definition

Σ is a finite alphabet of labels.

Definition ($\mathcal{R} = (\mathcal{T}, \overrightarrow{s})$: n-ary regular tree pattern over Σ)

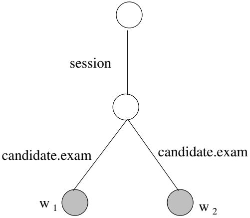
- $\mathcal{T} = (\Sigma, N, M, \mathcal{E})$ is the regular tree template composed of
  - a tree (N, M) with N as tree domain and $M \subseteq N \times N$ as associated set of edges.
  - an application $\mathcal{E} : M \longrightarrow REG(\Sigma)$
- $\overrightarrow{s}$=(w_1, ..., w_n) is the tuple of selected nodes.

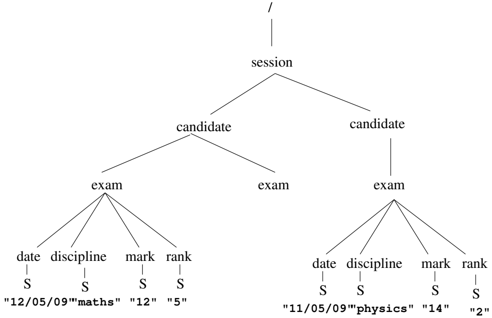
Regular tree pattern: the evaluation(1)

Let $\mathcal{R} = (\mathcal{T}, \overrightarrow{s})$ be a regular tree pattern with $\mathcal{T} = (\Sigma, N, M, \mathcal{E})$ and $\mathcal{D} = (D, \lambda, val)$ be an XML document

Mapping:

A mapping of $\mathcal{R}$ in $\mathcal{D}$ is a one-to-one function $\pi : N \longrightarrow D$



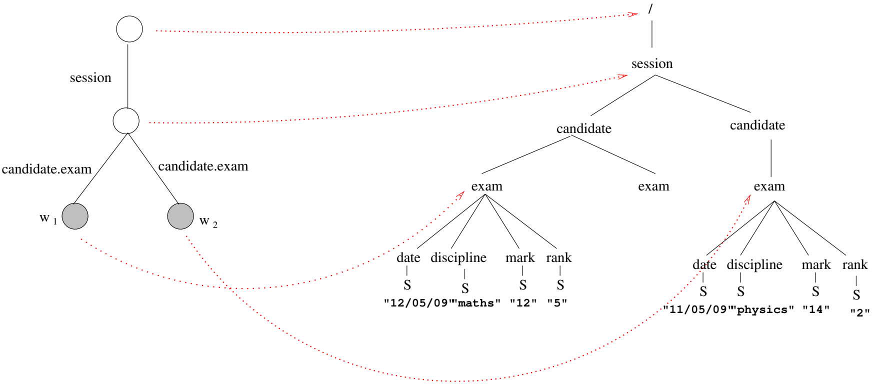


Regular tree pattern: the evaluation(1)

Let $\mathcal{R} = (\mathcal{T}, \overrightarrow{s})$ be a regular tree pattern with $\mathcal{T} = (\Sigma, N, M, \mathcal{E})$ and $\mathcal{D} = (D, \lambda, val)$ be an XML document

Mapping:

A mapping of $\mathcal{R}$ in $\mathcal{D}$ is a one-to-one function $\pi : N \longrightarrow D$

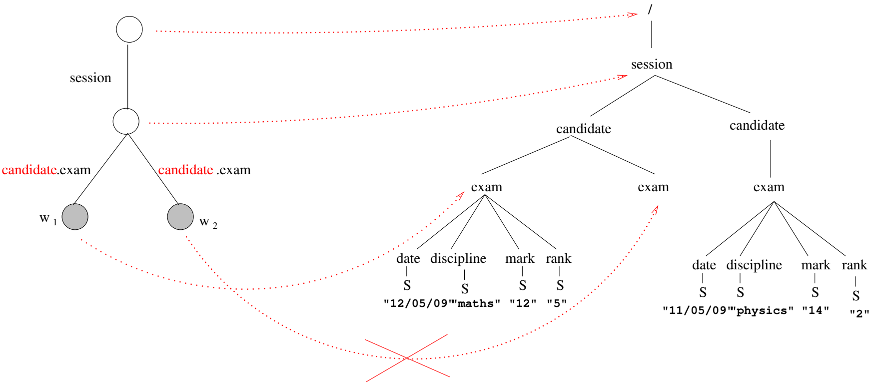


Regular tree pattern: the evaluation(1)

Let $\mathcal{R} = (\mathcal{T}, \overrightarrow{s})$ be a regular tree pattern with $\mathcal{T} = (\Sigma, N, M, \mathcal{E})$ and $\mathcal{D} = (D, \lambda, val)$ be an XML document

Mapping:

A mapping of $\mathcal{R}$ in $\mathcal{D}$ is a one-to-one function $\pi : N \longrightarrow D$

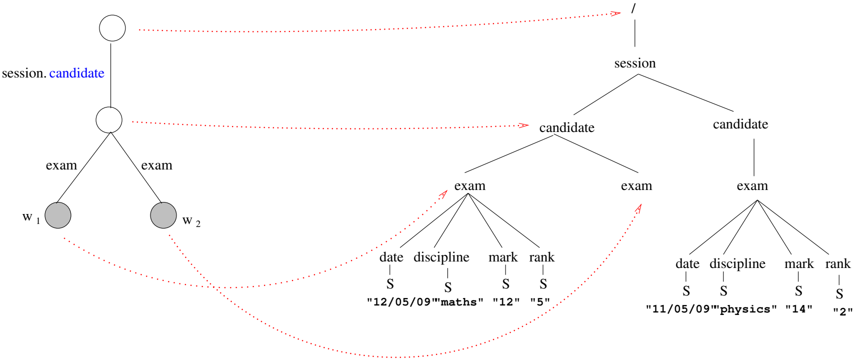


Regular tree pattern: the evaluation(1)

Let $\mathcal{R} = (\mathcal{T}, \overrightarrow{s})$ be a regular tree pattern with $\mathcal{T} = (\Sigma, N, M, \mathcal{E})$ and $\mathcal{D} = (D, \lambda, val)$ be an XML document

Mapping:

A mapping of $\mathcal{R}$ in $\mathcal{D}$ is a one-to-one function $\pi : N \longrightarrow D$

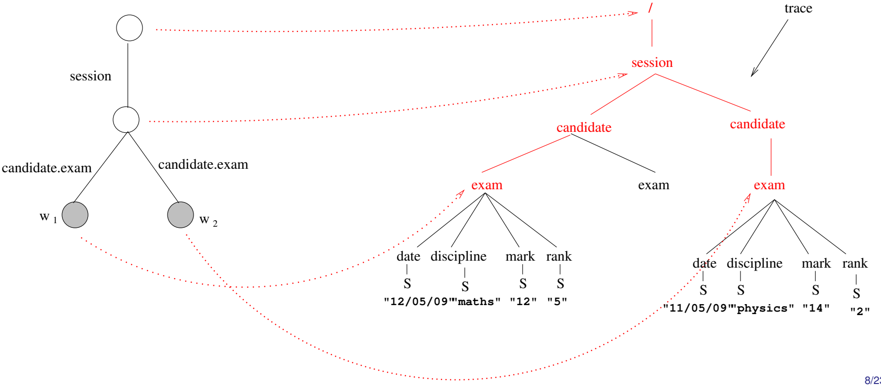


Regular tree pattern: the evaluation(1)

Let $\mathcal{R} = (\mathcal{T}, \overrightarrow{s})$ be a regular tree pattern with $\mathcal{T} = (\Sigma, N, M, \mathcal{E})$ and $\mathcal{D} = (D, \lambda, val)$ be an XML document

Mapping:

A mapping of $\mathcal{R}$ in $\mathcal{D}$ is a one-to-one function $\pi : N \longrightarrow D$

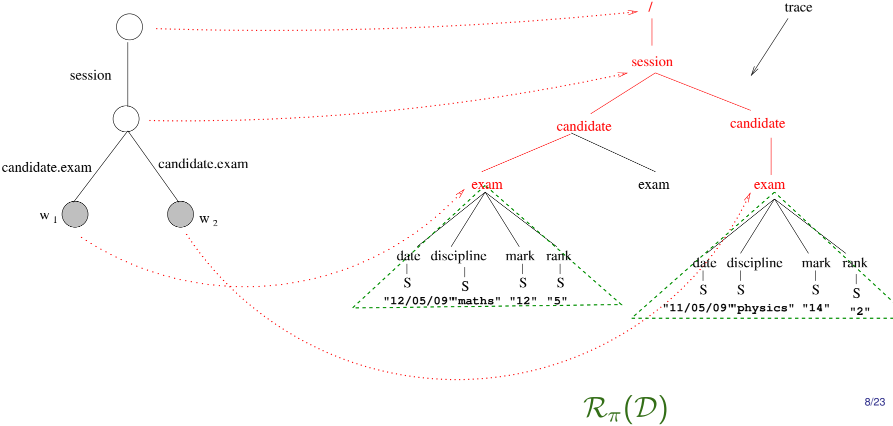


Regular tree pattern: the evaluation(1)

Let $\mathcal{R} = (\mathcal{T}, \overrightarrow{s})$ be a regular tree pattern with $\mathcal{T} = (\Sigma, N, M, \mathcal{E})$ and $\mathcal{D} = (D, \lambda, val)$ be an XML document

Mapping:

A mapping of $\mathcal{R}$ in $\mathcal{D}$ is a one-to-one function $\pi : N \longrightarrow D$



Regular tree pattern: the evaluation(2)

Evaluation of $\mathcal{R}$ over $\mathcal{D}$

- Let $\mathcal{P}$ be the set of all mappings of $\mathcal{R}$ in $\mathcal{D}$
- The evaluation of $\mathcal{R}$ over $\mathcal{D}$ according to $\pi \in \mathcal{P}$ is defined by : $\mathcal{R}_{\pi}(\mathcal{D})$=$(\mathcal{D}(\pi(w_1)), \ldots, \mathcal{D}(\pi(w_n)))$
  where $\overrightarrow{s}$=$(w_1, ..., w_n)$,
  and $\mathcal{D}(\pi(w_k))$ sub-tree rooted at $\pi(w_k)$.

Regular tree pattern: the evaluation(2)

Evaluation of $\mathcal{R}$ over $\mathcal{D}$

- Let $\mathcal{P}$ be the set of all mappings of $\mathcal{R}$ in $\mathcal{D}$
- The evaluation of $\mathcal{R}$ over $\mathcal{D}$ according to $\pi \in \mathcal{P}$ is defined by : $\mathcal{R}_\pi(\mathcal{D})=(\mathcal{D}(\pi(w_1)), \ldots, \mathcal{D}(\pi(w_n)))$
  where $\overrightarrow{s}=(w_1, ..., w_n)$,
  and $\mathcal{D}(\pi(w_k))$ sub-tree rooted at $\pi(w_k)$.
- The evaluation of $\mathcal{R}$ over $\mathcal{D}$ is then : $\mathcal{R}(\mathcal{D})= \bigcup_{\pi \in \mathcal{P}} \mathcal{R}_\pi(\mathcal{D})$

Modelling functional dependencies by RTPs

Definition

An XML functional dependency is an expression $fd = (\mathcal{F}D, c)$ where:

- $\mathcal{F}D = (\mathcal{T}, \overrightarrow{s} = \{p_1[E_1], p_2[E_2], ..., p_n[E_n], q[E_{n+1}]\})$ is a regular tree pattern.
  $p_1, ..., p_n$ and q are associated with an equality type $E_i \in \{V, N\}$ (i=1,..., n+1)
- c (context node) is an ancestor node of $p_1, p_2, ..., p_n$ (condition nodes) and of q (target node)

V is the value equality: ($w_1 \equiv_v w_2 \Leftrightarrow \mathcal{D}(w_1)$ and $\mathcal{D}(w_2)$ have the same value.)

N is the node equality: $w_1 \equiv_N w_2$ iff $w_1 = w_2$

Satisfaction of a functional dependency

Definition

A document $\mathcal{D}$ satisfies the functional dependency $(\mathcal{F}D, c)$ iff:

Satisfaction of a functional dependency

Definition

A document $\mathcal{D}$ satisfies the functional dependency $(\mathcal{F}D, c)$ iff:

IF for two traces, $\tau_1 = \mathtt{trace}_{\pi_1}(\mathcal{F}D, \mathcal{D})$ and $\tau_2 = \mathtt{trace}_{\pi_2}(\mathcal{F}D, \mathcal{D})$, with
(a) $\pi_1(c) =_N \pi_2(c)$
(b) $\forall i = 1, ..., n$, $\pi_1(p_i) =_{E_i} \pi_2(p_i)$,

Satisfaction of a functional dependency

Definition

A document $\mathcal{D}$ satisfies the functional dependency $(\mathcal{F}D, c)$ iff:

IF for two traces, $\tau_1 = \texttt{trace}_{\pi_1}(\mathcal{F}D, \mathcal{D})$ and $\tau_2 = \texttt{trace}_{\pi_2}(\mathcal{F}D, \mathcal{D})$, with
(a) $\pi_1(c) =_N \pi_2(c)$
(b) $\forall i = 1, ..., n$, $\pi_1(p_i) =_{E_i} \pi_2(p_i)$,

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

Modelling update classes by RTPs

We use the same model for updates.

- F. Gire and H. Idabal "Updates and Views Dependencies in Semi-structured Databases" IDEAS 2008

An update q is a composition of

$\longrightarrow$ a node selection process ($\mathcal{U}$)

Modelling update classes by RTPs

We use the same model for updates.

- F. Gire and H. Idabal "Updates and Views Dependencies in Semi-structured Databases" IDEAS 2008

An update q is a composition of

$\longrightarrow$ a node selection process ($\mathcal{U}$)

$\longrightarrow$ a replacement function (f)

Modelling update classes by RTPs

We use the same model for updates.

- F. Gire and H. Idabal "Updates and Views Dependencies in Semi-structured Databases" IDEAS 2008

An update q is a composition of

$\longrightarrow$ a node selection process ($\mathcal{U}$)

$\longrightarrow$ a replacement function (f)

$\Rightarrow q = f \; o \, \mathcal{U}$

Modelling update classes by RTPs

We use the same model for updates.

- F. Gire and H. Idabal "Updates and Views Dependencies in Semi-structured Databases" IDEAS 2008

An update q is a composition of

$\longrightarrow$ **a node selection process ($\mathcal{U}$)**

$\longrightarrow$ a replacement function (f)

$\Rightarrow q = f \; o \, \mathcal{U}$

for simplicity, we identify q to $\mathcal{U}$.

Modelling update classes by RTPs

We use the same model for updates.

- F. Gire and H. Idabal "Updates and Views Dependencies in Semi-structured Databases" IDEAS 2008

An update q is a composition of

$\longrightarrow$ **a node selection process ($\mathcal{U}$)**

$\longrightarrow$ a replacement function (f)

$\Rightarrow q = f \; o \; \mathcal{U}$

for simplicity, we identify q to $\mathcal{U}$.

$\mathcal{U}$ selects the nodes to be modified so it defines a **class of updates**

Modelling update classes by RTPs

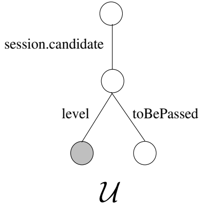
An update class $\mathcal{U}$:

“For each candidate still having to pass some remaining exams, update his level“

Modelling update classes by RTPs

An update class $\mathcal{U}$:

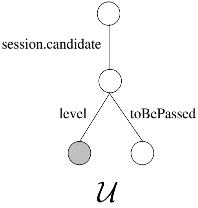
"For each candidate still having to pass some remaining exams, update his level"



Modelling update classes by RTPs

An update class $\mathcal{U}$:

"For each candidate still having to pass some remaining exams, update his level"



- $q_1 \in \mathcal{U}$: "For each candidate still having to pass some remaining exams, decrease his level to the level just below"
- $q_2 \in \mathcal{U}$: "For each candidate still having to pass some remaining exams, add a child node 'comment' to the 'level' node"

Independence Problem

Given

- Let *fd* be a functional dependency
- Let $\mathcal{U}$ be a class of updates
- Let $\mathcal{S}c$ be a Schema (given by an automaton $\mathcal{A}_{\mathcal{S}c}$)

Independence Problem

Given

- Let *fd* be a functional dependency
- Let $\mathcal{U}$ be a class of updates
- Let $\mathcal{S}c$ be a Schema (given by an automaton $\mathcal{A}_{\mathcal{S}c}$)

Independence Problem

Given

- Let *fd* be a functional dependency
- Let $\mathcal{U}$ be a class of updates
- Let $\mathcal{S}c$ be a Schema (given by an automaton $\mathcal{A}_{\mathcal{S}c}$)

Independence problem :

fd is independent w.r to $\mathcal{U}$ in the context of $\mathcal{S}c$ iff:
$\forall\, \mathcal{D} \in \mathtt{valid}(\mathcal{S}c), \forall\, q \in \mathcal{U}$ *with* $q(\mathcal{D}) \in \mathtt{valid}(\mathcal{S}c)$,

Independence Problem

Given

- Let *fd* be a functional dependency
- Let $\mathcal{U}$ be a class of updates
- Let $\mathcal{S}c$ be a Schema (given by an automaton $\mathcal{A}_{\mathcal{S}c}$)

Independence problem :

fd is independent w.r to $\mathcal{U}$ in the context of $\mathcal{S}c$ iff:
$\forall\, \mathcal{D} \in \mathtt{valid}(\mathcal{S}c), \forall\, q \in \mathcal{U}$ *with* $q(\mathcal{D}) \in \mathtt{valid}(\mathcal{S}c)$,
IF $\mathcal{D}$ satisfies fd

Independence Problem

Given

- Let *fd* be a functional dependency
- Let $\mathcal{U}$ be a class of updates
- Let $\mathcal{S}c$ be a Schema (given by an automaton $\mathcal{A}_{\mathcal{S}c}$)

Independence problem :

fd is independent w.r to $\mathcal{U}$ in the context of $\mathcal{S}c$ iff:
$\forall\, \mathcal{D} \in \mathtt{valid}(\mathcal{S}c), \forall\, q \in \mathcal{U}$ *with* $q(\mathcal{D}) \in \mathtt{valid}(\mathcal{S}c)$,
IF $\mathcal{D}$ satisfies fd THEN $q(\mathcal{D})$ satisfies fd as well.

Independence problem is PSPACE-hard

Proposition

Deciding whether a functional dependency *fd* is independent with respect to an update class $\mathcal{U}$ is a PSPACE-hard problem

Independence problem is PSPACE-hard

Proposition

Deciding whether a functional dependency *fd* is independent with respect to an update class $\mathcal{U}$ is a PSPACE-hard problem

Proof We reduce the well-known PSPACE-hard problem of the inclusion of two regular expressions, into the problem of independence.

Independence problem: static analysis

Independence problem: static analysis

fd is not independent w.r.t. to $\mathcal{U}$ in the context of $\mathcal{S}c$ iff:

Independence problem: static analysis

fd is not independent w.r. to $\mathcal{U}$ in the context of $\mathcal{S}c$ iff:

$\exists \mathcal{D} \in \mathtt{valid}(\mathcal{S}c)$, $\exists q \in \mathcal{U}$ with $q(\mathcal{D}) \in \mathtt{valid}(\mathcal{S}c)$ and,

Independence problem: static analysis

fd is not independent w.r. to $\mathcal{U}$ in the context of $\mathcal{S}c$ iff:

$\exists \mathcal{D} \in \texttt{valid}(\mathcal{S}c)$, $\exists q \in \mathcal{U}$ with $q(\mathcal{D}) \in \texttt{valid}(\mathcal{S}c)$ and,
$\mathcal{D}$ satisfies *fd* while $q(\mathcal{D})$ does not satisfy *fd*

Independence problem: static analysis

fd is not independent w.r.t. to $\mathcal{U}$ in the context of $\mathcal{S}c$ iff:

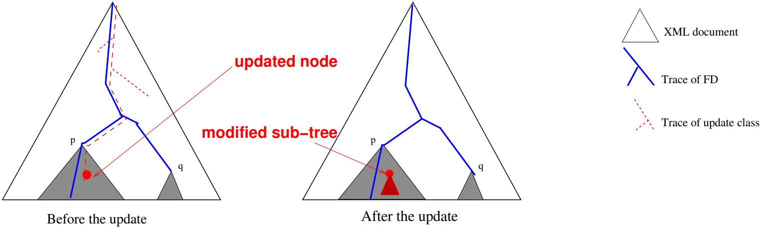
$\exists \mathcal{D} \in \mathtt{valid}(\mathcal{S}c)$, $\exists q \in \mathcal{U}$ with $q(\mathcal{D}) \in \mathtt{valid}(\mathcal{S}c)$ and,
$\mathcal{D}$ satisfies *fd* while $q(\mathcal{D})$ does not satisfy *fd*

So there is a node n of $\mathcal{D}$ whose update by q generates a witness of the violation of fd in $q(\mathcal{D})$.

Independence problem: static analysis

First case

n belongs to one of the sub-trees rooted at condition or target nodes

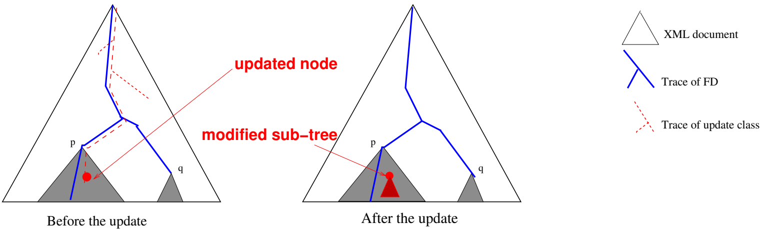


Independence problem: static analysis

First case

n belongs to one of the sub-trees rooted at condition or target nodes

$\longrightarrow$ its update doesn't modify the trace of $\mathcal{F}D$ in $\mathcal{D}$



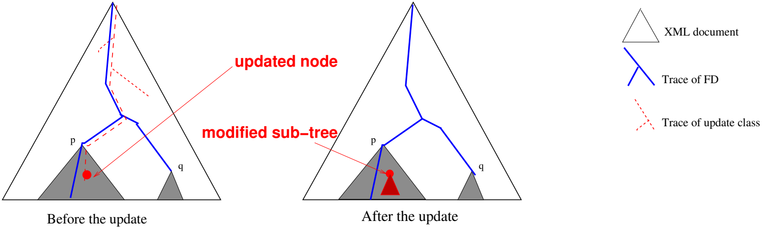
Independence problem: static analysis

First case

n belongs to one of the sub-trees rooted at condition or target nodes

→ its update doesn't modify the trace of $\mathcal{F}D$ in $\mathcal{D}$

→ but the modified value of this subtree generates the violation of fd in q($\mathcal{D}$).



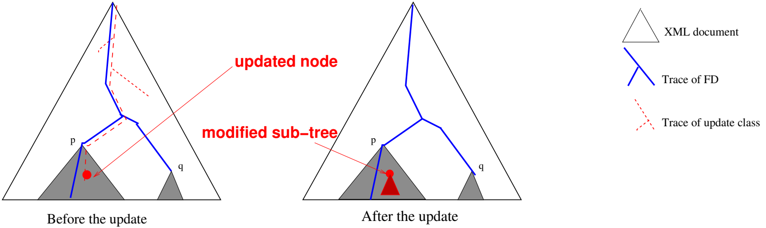
Independence problem: static analysis

First case

n belongs to one of the sub-trees rooted at condition or target nodes

$\longrightarrow$ its update doesn't modify the trace of $\mathcal{F}D$ in $\mathcal{D}$

$\longrightarrow$ but the modified value of this subtree generates the violation of fd in q($\mathcal{D}$).



$\exists$ a mapping π of $\mathcal{F}D$ on $\mathcal{D}$ and $\exists$ a mapping π' of $\mathcal{U}$ on $\mathcal{D}$ such that:

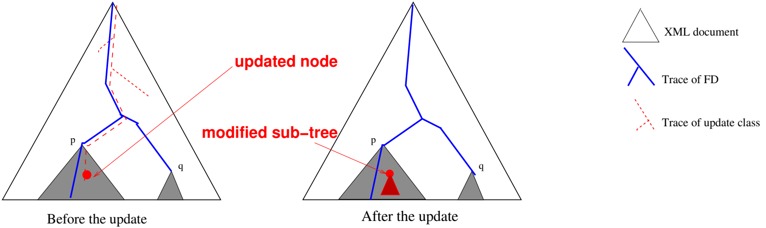
Independence problem: static analysis

First case

n belongs to one of the sub-trees rooted at condition or target nodes

$\longrightarrow$ its update doesn't modify the trace of $\mathcal{F}D$ in $\mathcal{D}$

$\longrightarrow$ but the modified value of this subtree generates the violation of fd in q($\mathcal{D}$).

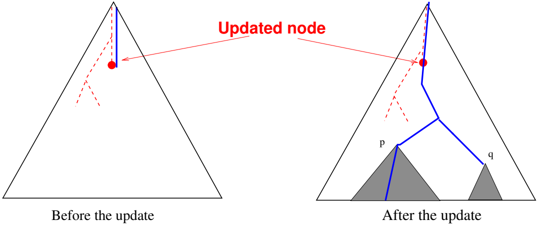


$\exists$ a mapping π of $\mathcal{F}D$ on $\mathcal{D}$ and $\exists$ a mapping π' of $\mathcal{U}$ on $\mathcal{D}$ such that:
$\mathcal{N}(\pi'(\overrightarrow{s}_{\mathcal{U}})) \cap \mathcal{N}(\mathcal{F}D_{\pi}(\mathcal{D})) \neq \emptyset$

Independence problem: static analysis

Second case

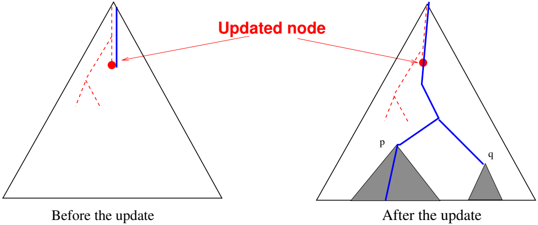
the updated node n generates a new trace of $\mathcal{F}D$ in q($\mathcal{D}$) that contributes to the violation of *fd*



Independence problem: static analysis

Second case

the updated node n generates a new trace of $\mathcal{F}D$ in q($\mathcal{D}$) that contributes to the violation of *fd*

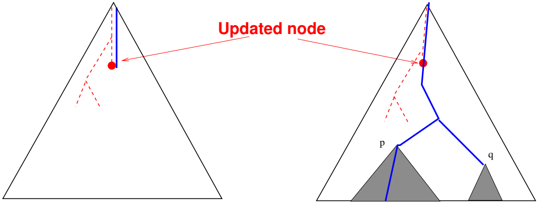


$\exists$ a mapping π of $\mathcal{F}D$ on $q(\mathcal{D})$ and $\exists$ a mapping π' of $\mathcal{U}$ on $q(\mathcal{D})$ such that:

Independence problem: static analysis

Second case

the updated node n generates a new trace of $\mathcal{F}D$ in q($\mathcal{D}$) that contributes to the violation of *fd*



Before the update

After the update

$\exists$ a mapping π of $\mathcal{F}D$ on $q(\mathcal{D})$ and $\exists$ a mapping π' of $\mathcal{U}$ on $q(\mathcal{D})$ such that:

$\mathcal{N}(\pi'(\overrightarrow{s}_{\mathcal{U}})) \cap \mathcal{N}(trace_{\pi}(\mathcal{F}D, q(\mathcal{D}))) \neq \emptyset$

An independence criterion

Definition

Let $\mathcal{L}$ be the language of XML documents $\mathcal{D}$ satisfying:
(i) $\mathcal{D} \in \texttt{valid}(\mathcal{S}c)$
(ii) $\exists\ \tau_{\mathcal{F}D} = trace_{\pi}(\mathcal{F}D, \mathcal{D})$, w.r to a mapping π of $\mathcal{F}D$ on $\mathcal{D}$,
and $\exists\ \tau_{\mathcal{U}} = trace_{\pi'}(\mathcal{U}, \mathcal{D})$,w.r to a mapping π' of $\mathcal{U}$ on $\mathcal{D}$, such that:
$\mathcal{N}(\pi'(\overrightarrow{s}_{\mathcal{U}})) \bigcap (\mathcal{N}(trace_{\pi}(\mathcal{F}D, \mathcal{D})) \cup \mathcal{N}(\mathcal{F}D_{\pi}(\mathcal{D}))) \neq \emptyset$

An independence criterion

Definition

Let $\mathcal{L}$ be the language of XML documents $\mathcal{D}$ satisfying:
(i) $\mathcal{D} \in \texttt{valid}(\mathcal{S}c)$
(ii) $\exists\ \tau_{\mathcal{F}D} = trace_{\pi}(\mathcal{F}D, \mathcal{D})$, w.r to a mapping π of $\mathcal{F}D$ on $\mathcal{D}$,
and $\exists\ \tau_{\mathcal{U}} = trace_{\pi'}(\mathcal{U}, \mathcal{D})$,w.r to a mapping π' of $\mathcal{U}$ on $\mathcal{D}$, such that:
$\mathcal{N}(\pi'(\overrightarrow{s}_{\mathcal{U}})) \bigcap (\mathcal{N}(trace_{\pi}(\mathcal{F}D, \mathcal{D})) \cup \mathcal{N}(\mathcal{F}D_{\pi}(\mathcal{D}))) \neq \emptyset$

Proposition[Independence criterion *IC*]

If $\mathcal{L}$ is empty then *fd* is independent w.r to $\mathcal{U}$ in the context of $\mathcal{S}c$.

Checking criterion *IC* & Complexity

- A regular Bottom-Up automaton $\mathcal{A}$ recognizing $\mathcal{L}$ can be built from the automaton $\mathcal{A}_{\mathcal{S}c}$ and the regular tree patterns $\mathcal{F}D$ and $\mathcal{U}$

Checking criterion *IC* & Complexity

- A regular Bottom-Up automaton $\mathcal{A}$ recognizing $\mathcal{L}$ can be built from the automaton $\mathcal{A}_{\mathcal{S}c}$ and the regular tree patterns $\mathcal{F}D$ and $\mathcal{U}$
- The size $|\mathcal{A}|$ of the automaton $\mathcal{A}$ is in $O(a_{\mathcal{U}} a_{\mathcal{F}D} \times |\Sigma|^2 \times |\mathcal{A}_{\mathcal{S}c}| \times |\mathcal{U}| \times |\mathcal{F}D|)$, where $a_{\mathcal{U}}$ and $a_{\mathcal{F}D}$ are the maximal arities of $\mathcal{U}$ and $\mathcal{F}D$ respectively

Checking criterion *IC* & Complexity

- A regular Bottom-Up automaton $\mathcal{A}$ recognizing $\mathcal{L}$ can be built from the automaton $\mathcal{A}_{\mathcal{S}c}$ and the regular tree patterns $\mathcal{F}D$ and $\mathcal{U}$
- The size $|\mathcal{A}|$ of the automaton $\mathcal{A}$ is in $O(a_{\mathcal{U}} a_{\mathcal{F}D} \times |\Sigma|^2 \times |\mathcal{A}_{\mathcal{S}c}| \times |\mathcal{U}| \times |\mathcal{F}D|)$, where $a_{\mathcal{U}}$ and $a_{\mathcal{F}D}$ are the maximal arities of $\mathcal{U}$ and $\mathcal{F}D$ respectively
- The independence criterion *IC* is polynomial: the emptiness of the language $\mathcal{L}$ is testable in $O(a_{\mathcal{U}}^2\ a_{\mathcal{F}D}^2 \times |\Sigma|^4 \times |\mathcal{A}_{\mathcal{S}c}|^2 \times |\mathcal{U}|^2 \times |\mathcal{F}D|^2)$ time.

Conclusion

Main results :

- An uniform formalism based on RTPs
- The independence Problem is PSPACE-hard
- A sufficient criterion for checking the independence testable in polynomial time

Conclusion

Main results :

- An uniform formalism based on RTPs
- The independence Problem is PSPACE-hard
- A sufficient criterion for checking the independence testable in polynomial time

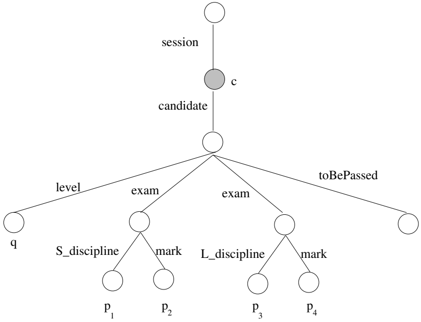
To do :

- Axiomatisation and verification problems
- Necessary and sufficient condition in the case of special fds
- Implementation

THANKS

Advantages of our model

"Two candidates with the same mark in at least two disciplines and also having some remaining exams to pass, receive the same level".



$\mathcal{S}c$:

- exam : ((S_discipline|L_discipline), mark).

- L_disciplines appear after S_disciplines.

- Labels of two edges outgoing from a same node can share a common prefix.
- Leaves of $\mathcal{F}D$ are not only condition or target nodes.
- The order is relevant.