Update propagation through security views

Sławek Staworko, Iovka Boneva, Benoît Groz

Université Lille 1, Mostrare project, INRIA

March 22, 2010

Outline

1. Updates and (security) Views
2. View Inversion
3. Update Propagation

Views and Updates

Database views:

- facilitate access to data
- remove irrelevant data
- restructure the presentation of the data

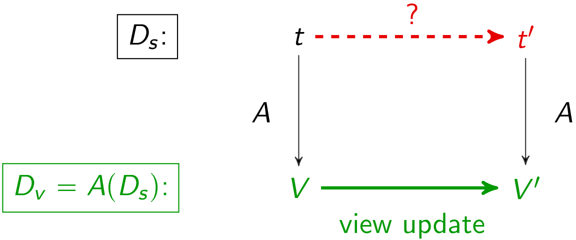
Database *security views*:

- hide sensitive data

View update propagation

- document t, view $V = A(t)$,
- view update by the user: $V \mapsto V'$
- find a propagation: $t \overset{?}{\mapsto} t'$

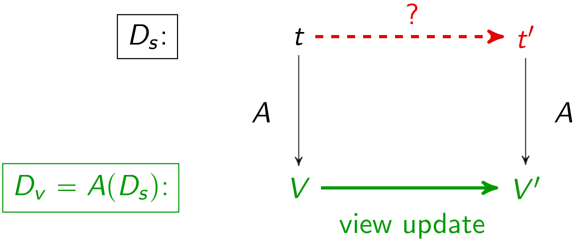
The update propagation problem



Find an update $t \mapsto t'$ (a propagation of $V \mapsto V'$) such that:

- $t \mapsto t'$ is *side-effect free*: $A(t') = V'$
- $t \mapsto t'$ is *schema compliant*: $t' \models D$;

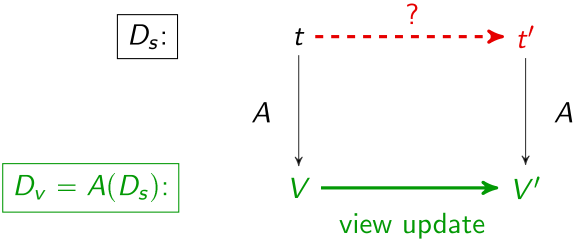
The update propagation problem



Find an update $t \mapsto t'$ (a propagation of $V \mapsto V'$) such that:

- $t \mapsto t'$ is *side-effect free*: $A(t') = V'$
- $t \mapsto t'$ is *schema compliant*: $t' \models D$;
- $t \mapsto t'$ has *constant complement* [Bancilhon, Spyratos'81] : no modification of the hidden parts

The update propagation problem



Find an update $t \mapsto t'$ (a propagation of $V \mapsto V'$) such that:

- $t \mapsto t'$ is *side-effect free*: $A(t') = V'$
- $t \mapsto t'$ is *schema compliant*: $t' \models D$;
- $t \mapsto t'$ is *optimal*: minimal modification of the hidden parts.

Our framework

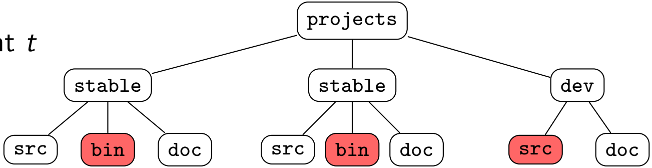
- XML documents: unranked, ordered trees, with node identifiers.
- Schema: DTD
- View given by an annotation[Farkas et al '02, Fan et al'04.]
  - $A(a, b) = \mathbb{0} \implies$ the nodes labeled b under a node a are not visible
  - $A(a, b) = \mathbb{1} \implies$ the nodes labeled b under a visible node a are visible

  Upward closed visibility of nodes: the descendants of a hidden node are hidden as well (regardless of their annotation)

Our framework

- XML documents: unranked, ordered trees, with node identifiers.
- Schema: DTD
- View given by an annotation

document t

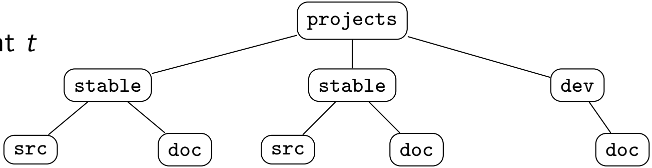


$$A(projects, stable) = A(projects, dev) = A(stable, src) = \ldots = \mathbb{1}$$
$$A(dev, src) = A(stable, bin) = \mathbb{0}$$

Our framework

- XML documents: unranked, ordered trees, with node identifiers.
- Schema: DTD
- View given by an annotation

document t



$$A(projects, stable) = A(projects, dev) = A(stable, src) = \ldots = \mathbb{1}$$
$$A(dev, src) = A(stable, bin) = \mathbb{0}$$

Our framework

- XML documents: unranked, ordered trees, with node identifiers.
- Schema: DTD
- View given by an annotation

and updates ?

Editing scripts

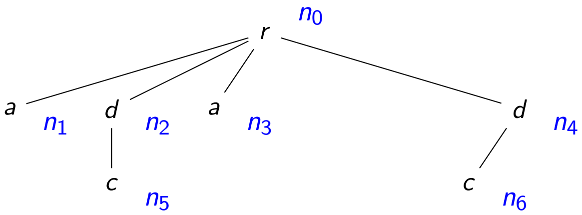
Update

Input tree with nodes labelled:

- *Del*: nodes to be deleted
- *Ins*: nodes to be inserted

* Deletion is recursive: delete whole subtrees

* Insertion of a subtree, not of internal nodes



Editing scripts

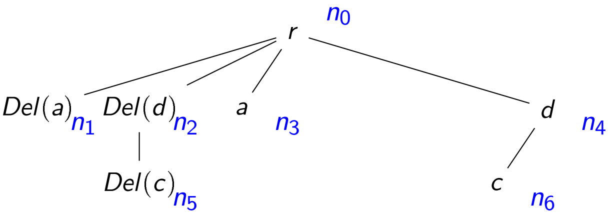
Update

Input tree with nodes labelled:

- *Del*: nodes to be deleted
- *Ins*: nodes to be inserted

* Deletion is recursive: delete whole subtrees

* Insertion of a subtree, not of internal nodes



Editing scripts

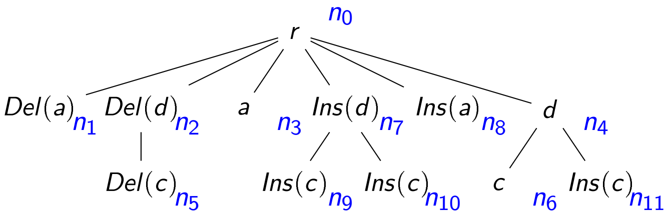
Update

Input tree with nodes labelled:

- *Del*: nodes to be deleted
- *Ins*: nodes to be inserted

* Deletion is recursive: delete whole subtrees

* Insertion of a subtree, not of internal nodes



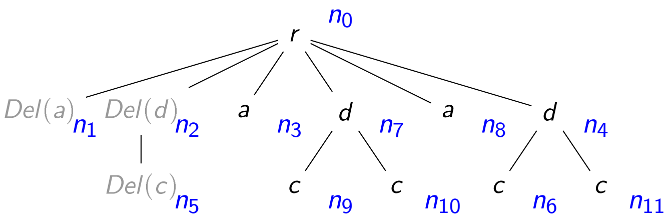
Editing scripts

Update

Input tree with nodes labelled:

- *Del*: nodes to be deleted
- *Ins*: nodes to be inserted

* Deletion is recursive: delete whole subtrees
* Insertion of a subtree, not of internal nodes



$\Longrightarrow$ captures XQuery Update snapshot semantics

Our framework

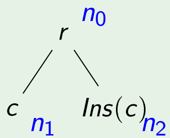
- XML documents: unranked, ordered trees, with node identifiers.
- Schema: DTD
- View given by an annotation.
- Updates are given as editing scripts: the alignment of the input and output document on their common nodes.

Nice properties:

- One can compute a DTD D_v that captures the set of all view documents. Hence the user will only apply view updates that have a propagation
- The constraints are local in a DTD: the modification of a node affects only its siblings and their descendants.

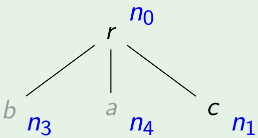
Identifiers, a choice of some consequence

Example

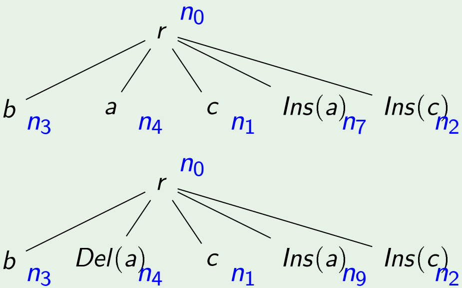


View update

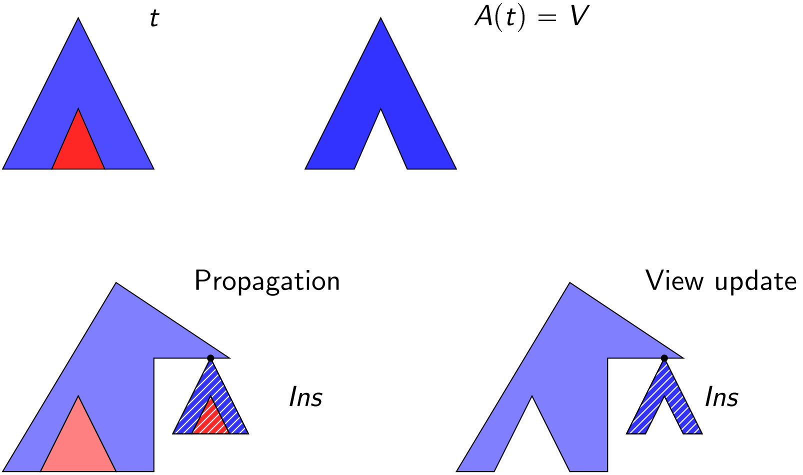
Schema: $r \to b^{\mathbb{0}}(c^{\mathbb{1}} + \epsilon)(a^{\mathbb{0}} \cdot c^{\mathbb{1}})^*$



Source tree

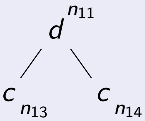


View Inversion



Computing the view inverse

Example

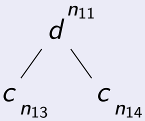


Schema and View:

$$d \to ((a^{\mathbb{0}} + b^{\mathbb{0}}) \cdot c^{\mathbb{1}})^*$$

Computing the view inverse

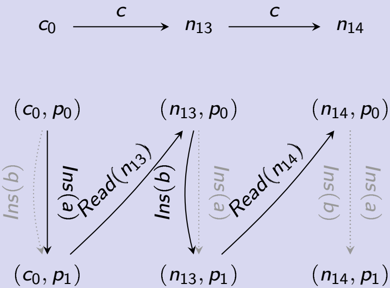
Example

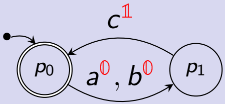


Schema and View:

$$d \rightarrow ((a^{\mathbb{0}} + b^{\mathbb{0}}) \cdot c^{\mathbb{1}})^*$$

Example of construction



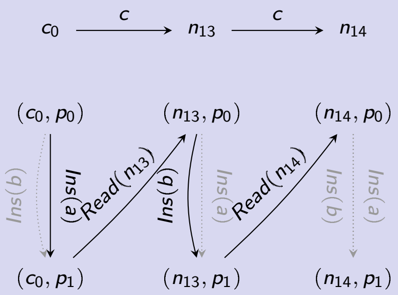


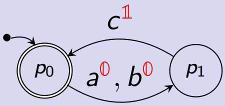
Computing the view inverse

Lemma

H_n captures all possible inversions for the sequence of children of the node n as paths from an initial to a final state.

Example of construction





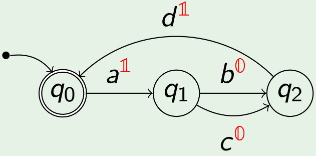
Computing the update propagation

(As for inverse), for every node n common to t_s and $Out(S_v)$, construct a graph G_n representing the set of all possible sequences of children of n in all $Out(S_s)$ for all update propagation S_s.

G_n has to handle insertion and deletion of nodes.

Example

The source DTD



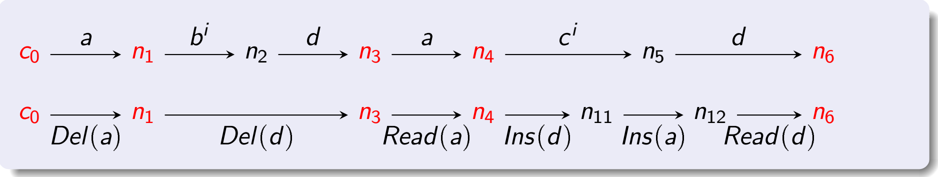
the sequence of children of n in t_s

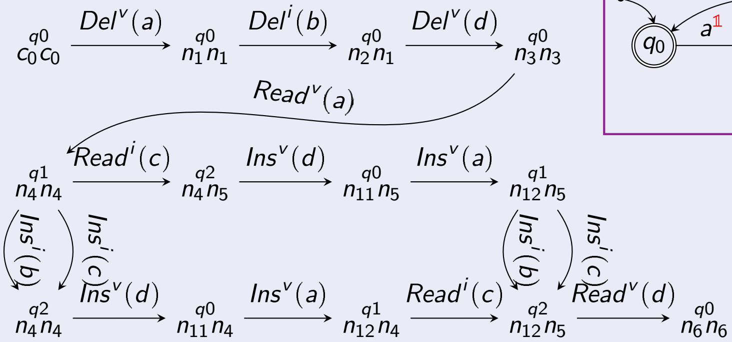
n_1	n_2	n_3	n_4	n_5	n_6
a	b	d	a	c	d

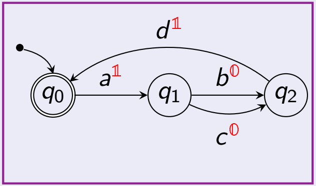
the edit script S_v

n_1	n_3	n_4	n_{11}	n_{12}	n_6
$Del(a)$	$Del(d)$	$Read(a)$	$Ins(d)$	$Ins(a)$	$Read(d)$

Computing the update propagation







Computing the update propagation

Lemma

For each node n, the graph G_n contains all possible update propagations of S_s restricted to the sequence of children of n as paths from an initial to a final node in the graph.

Theorem

The set of graphs G_n for all node n common to t_s and $Out(S_v)$ captures all side-effect free and schema-compliant update propagations of S_v.

As for inversion, the update propagation script S_s is computed bottom-up on the structure of t. Inversion is used when inserting a subtree.

Computing an optimal update propagation

For all node n, construct a graph G_n^* which contains only the optimal update propagations from G_n. The construction is bottom-up.

1. Associate a cost to each edge in G_n:
    - *delete* : the size of the deleted tree;
    - *invisible insert* : the size of the minimal tree with the corresponding root label;
    - *visible insert* : the size of the minimal inversion tree;
    - *invisible read* : 0;
    - *visible read* : the sum of the costs of the optimal propagation graphs for the children, computed recursively.
2. Remove from G_n all non-optimal paths.

Complexity and remarks

- The size of a minimal propagation may be exponential in size of D_s (the minimal tree satisfying a DTD can have an exponential size).

$$r \rightarrow a_n, \qquad a_i \rightarrow a_{i-1} a_{i-1}, \qquad a_0 \rightarrow \epsilon$$

- If for each label a, the tree to be inserted whenever a is (invisibly) inserted is input of the problem, then the optimal propagation is of polynomial size.
- The model can be extended with some simple, local preferences for choosing an optimal propagation (among all possible ones).

Future work

- generalize views, schema ...
- generalize updates
- add constraints to the updates (node typing ...)
- propagating update programs instead of editing script ($V \mapsto V'$)