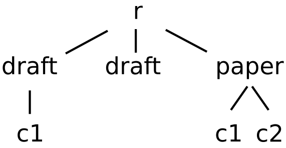
Translating Updates applied on Views

I.Boneva, A-C. Caron, B.Groz, Y.Roos, S.Staworko, S.Tison

LIFL, Université Lille 1

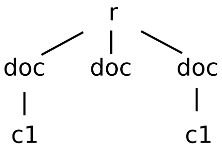
ICDT, March, 2011

Introduction: example



View

"hide all c2,
rename draft, papers into docs"



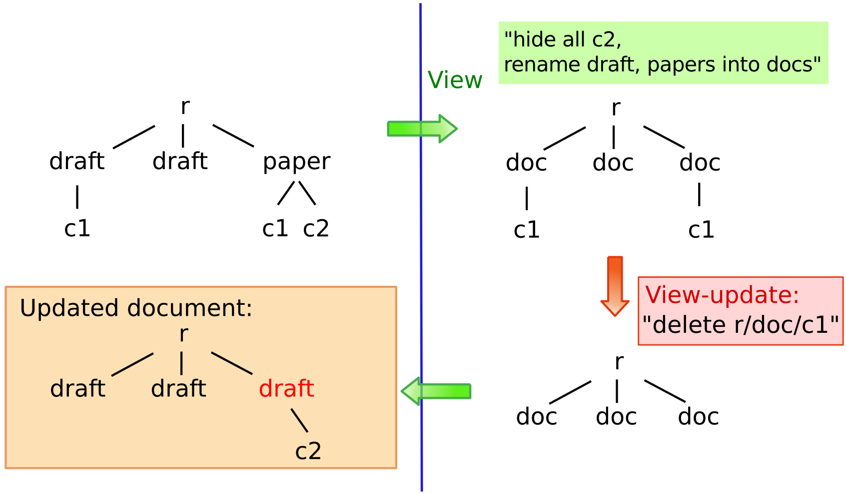
Schema:

r	$\longrightarrow$	(draft | paper)*
draft	$\longrightarrow$	c1? | c2?
paper	$\longrightarrow$	c1.c2

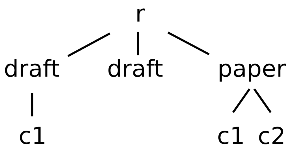
View-Schema:

r	$\longrightarrow$	doc*
doc	$\longrightarrow$	c1?

Introduction: example



Introduction: example



View

"hide all c2,
rename draft, papers into docs"

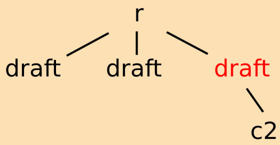
r
doc
doc
doc

Translation of the view update

"delete r/draft/c1,
delete r/paper/c1,
for $p in r/paper return rename node $p as draft"

View-update:
"delete r/doc/c1"

Updated document:



The *View-Update* pb

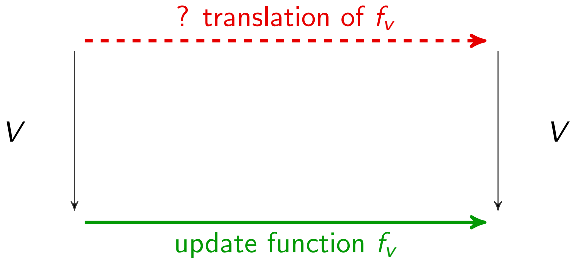


Figure: View update propagation: a synopsis.

Outline

1. Tree Alignments for Updates
   - Tree Alignments for Document Transformations
   - Results on Functionality

2. Update Translation
   - Polynomial time algorithms

3. Update Translation with Constraints
   - Intractability in general
   - A restriction that makes all problems decidable

Outline

1. Tree Alignments for Updates
   - Tree Alignments for Document Transformations
   - Results on Functionality
2. Update Translation
3. Update Translation with Constraints

Tree alignments

We note Σ_ε for $\Sigma \cup \{\varepsilon\}$.

Definition

Tree over $(\Sigma_\varepsilon)^k \backslash \{(\varepsilon, \ldots, \varepsilon)\}$ s.t. :

- label of the root is $(r, \ldots, r)$
- if node n has ε on i^{th} component, its descendants also.

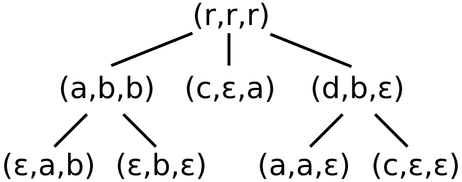


Figure: A tree alignment with $k = 3$

Projections

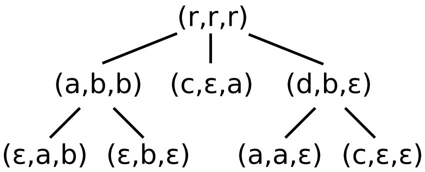


Figure: tree alignment t

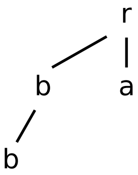


Figure: projection $\pi_3(t)$

Projections

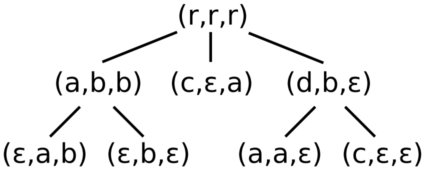


Figure: tree alignment t

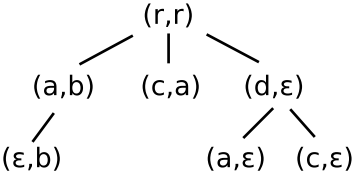


Figure: projection $\pi_{1,3}(t)$

Projections

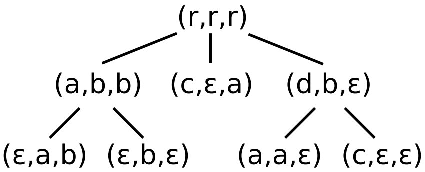


Figure: tree alignment t

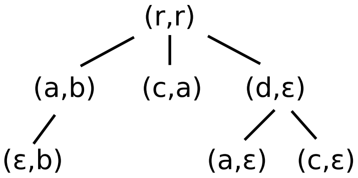


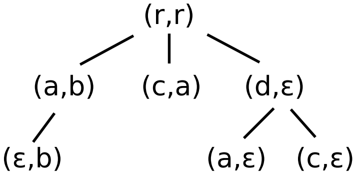
Figure: projection $\pi_{1,3}(t)$

Proposition

Projections of a regular set of alignments are also regular sets of alignments.

Updates=editing scripts

An *editing script* is a binary(k=2) tree alignment t.

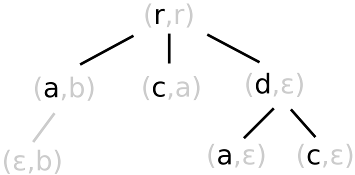


⇒: right subtree was deleted, leaf b was inserted, some nodes were relabeled

Updates=editing scripts

An *editing script* is a binary(k=2) tree alignment t.

Input=$\pi_1(t)$.

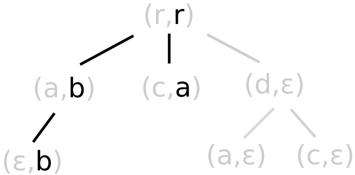


⇒: right subtree was deleted, leaf b was inserted, some nodes were relabeled

Updates=editing scripts

An *editing script* is a binary(k=2) tree alignment t.

Output=$\pi_2(t)$.

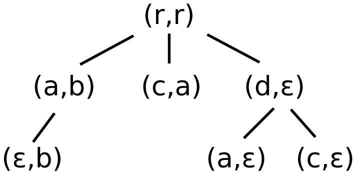


⇒: right subtree was deleted, leaf b was inserted, some nodes were relabeled

Updates=editing scripts

An *editing script* is a binary(k=2) tree alignment t.

Input$=\pi_1(t)$.
Output$=\pi_2(t)$.



$\Rightarrow$: right subtree was deleted, leaf b was inserted, some nodes were relabeled

Equivalence of editing scripts

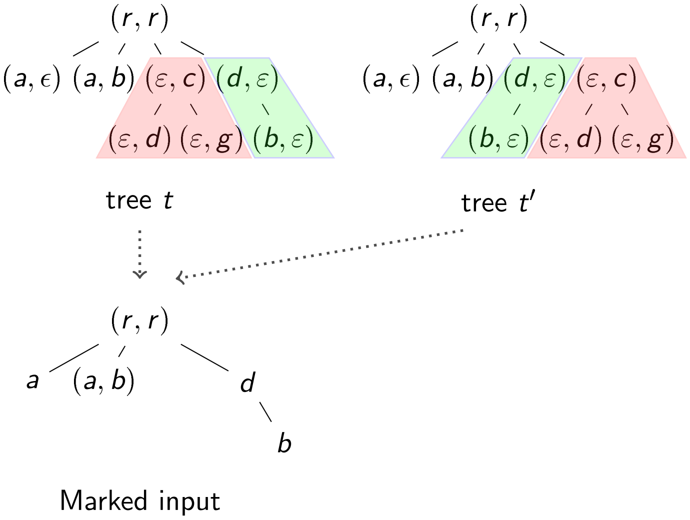


Figure: $t \sim_{eq} t'$

Equivalence of editing scripts

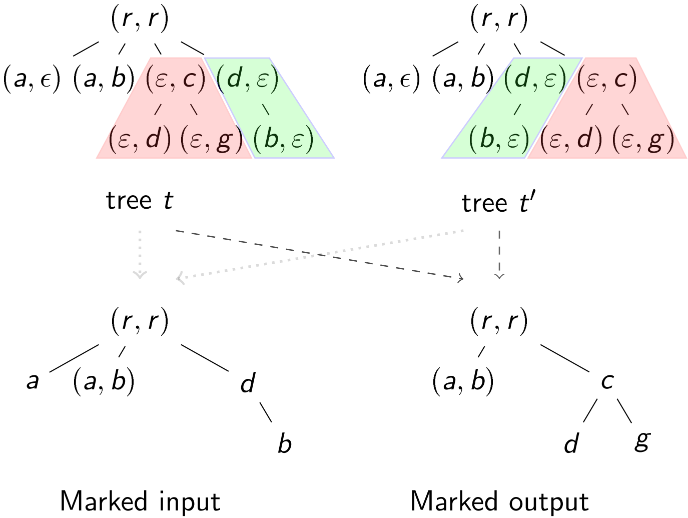


Figure: $t \sim_{eq} t'$

Equivalence of editing scripts

(r, r)

(ε, c) (d, ε)

tree t

(r, r)

d

Marked input(t)

(r, r)

(d, c)

tree t'

(r, r)

(d, c)

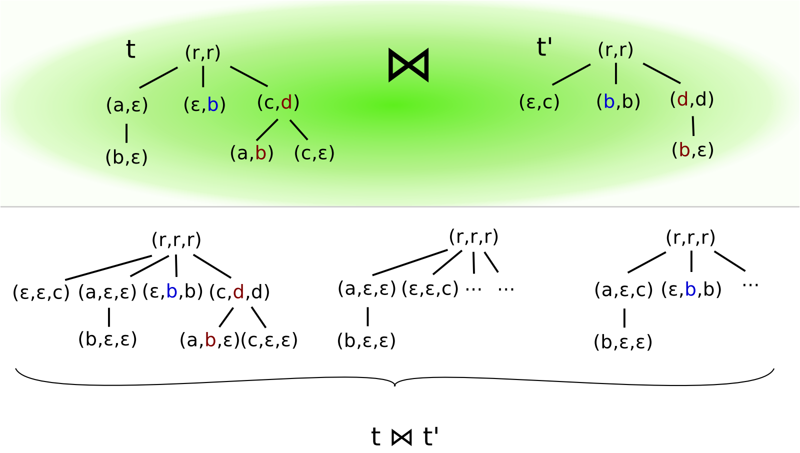
Marked input(t')

Figure: $t \not\sim_{eq} t'$

Synchronization

Definition: synchronization

L_1, L_2 sets of editing scripts.
$L_1 \bowtie L_2 = \{t \mid \pi_{1,2}(t) \in L_1 \wedge \pi_{2,3}(t) \in L_2\}$



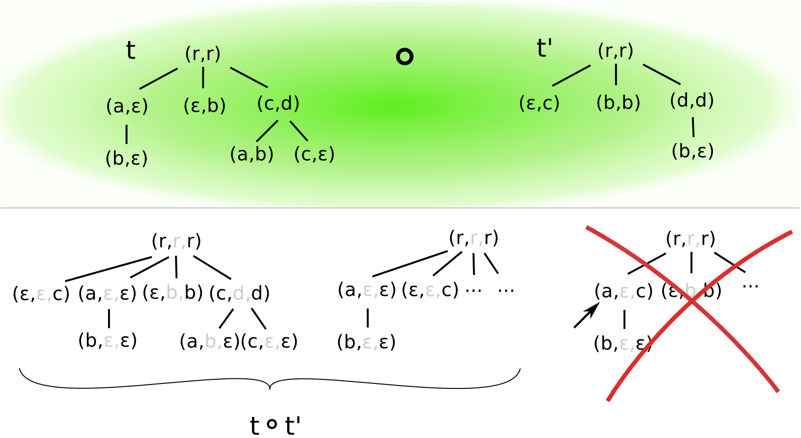
Composition

Definition: composition

L_1, L_2 sets of editing scripts.

$$L_1 \circ L_2 = \pi_{1,3}\left((L_1 \bowtie L_2) \cap L_{\cancel{\Sigma\times\{\varepsilon\}\times\Sigma}}\right)$$

… "deletions are forever"!



Properties of compositions

Proposition

Composition is associative

In a nutshell

We defined tree alignments, editing scripts as the special binary case.
We defined

- projections $\pi_{i_1,i_2,\ldots i_j}$: alignment $\mapsto$ alignment,
- composition $L_1 \circ L_2$: editing scripts $\mapsto$ set of editing scripts

Those operations preserve regularity.

In a nutshell

We defined tree alignments, editing scripts as the special binary case.
We defined

- projections $\pi_{i_1,i_2,\ldots i_j}$: alignment $\mapsto$ alignment,
- composition $L_1 \circ L_2$: editing scripts $\mapsto$ set of editing scripts

Those operations preserve regularity.

- the equivalence relation $\sim_{eq}$ between trees

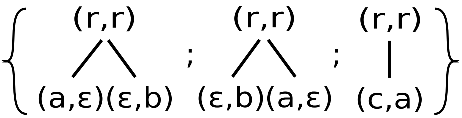
The closure under equivalence does not preserve regularity.

Update function

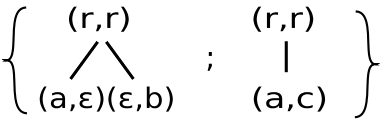
Definition

An *update function* is a **regular** set of editing scripts f_v such that $\forall t, t' \in f_v.\pi_1(t) = \pi_1(t') \Rightarrow t \sim_{eq} t'$.

Example :



update function



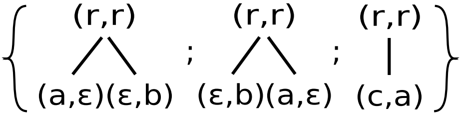
not an update function

Update function

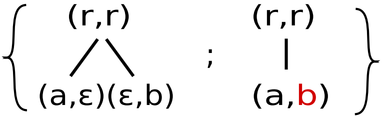
Definition

An *update function* is a **regular** set of editing scripts f_v such that $\forall t, t' \in f_v . \pi_1(t) = \pi_1(t') \Rightarrow t \sim_{eq} t'$.

Example :



update function



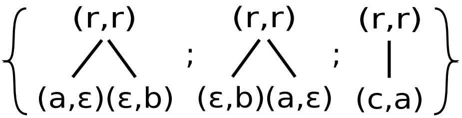
?

Update function

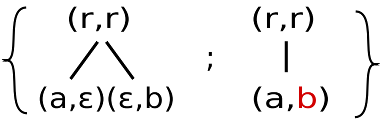
Definition

An *update function* is a **regular** set of editing scripts f_v such that $\forall t, t' \in f_v.\pi_1(t) = \pi_1(t') \Rightarrow t \sim_{eq} t'$.

Example :



update function



not an update function

Testing functionality

Proposition

Given a regular set L of editing scripts, we can test in polynomial time whether L is an update function

Plandowski's algorithm for testing equivalence of two morphisms on a context-free language.

Making a transformation functional

Theorem

Given a regular set of editing scripts L, we can 'effectively' compute an update function f_L such that $f_L \subseteq L$ and the domains of f_L and L are equal $(\pi_1(f_L) = \pi_1(L))$.

Outline

Views

Reminder

An update u is an editing script (binary alignment).
An update function f_v is a regular set of Ed.S that defines a functional transformation.

Definition

A *view* V is an update function over alphabet $\Sigma \times \Sigma_\varepsilon$: no insertions; only deletions.

The view update problem: defining translations

Definition

a set of Ed.S. L_s is a *translation* of f_v (w.r.t. V) iff $L_s \circ V \sim_{eq} V \circ f_v$.

$$\begin{tikzcd}
{} \arrow[r, dashed, "\text{? translation of } f_v"] \arrow[d, "V"'] & {} \arrow[d, "V"] \\
{} \arrow[r, "\text{update function } f_v"'] & {}
\end{tikzcd}$$

Figure: View update propagation: a synopsis.

The problems we solve

Problem: [Finding a translation]

Input: view V, update function f_v
Output: *a translation for f_v on the source.*

Problem: [Testing a translation]

Input: view V, update function f_v, and regular set of source updates L_s
Question: *is L_s a translation for f_v on the source?*

Translations for unconstrained updates

Proposition[Compute a translation]

From V and f_v, if $\pi_2(f_v) \subseteq \pi_2(V)$ we can compute an automaton A such that $L(A)$ is a translation of f_v, in polynomial time.

$\Rightarrow$: Using previous theorem we can compute a functional translation of f_v.

Translations for unconstrained updates

Proposition[Compute a translation]

From V and f_v, if $\pi_2(f_v) \subseteq \pi_2(V)$ we can compute an automaton A such that $L(A)$ is a translation of f_v, in polynomial time.

$\Rightarrow$: Using previous theorem we can compute a functional translation of f_v.

Proposition[Test a translation]

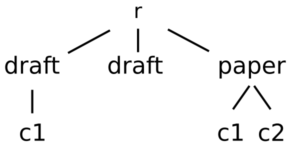
Given view V, update function f_v, and set of source updates L_s, it is decidable whether L_s is a translation of f_v.

We check that $L_s \circ V \cup V \circ f_v$ is an update function, which can be achieved in $\textsc{Ptime}$ once equality of the domains is checked.

Outline

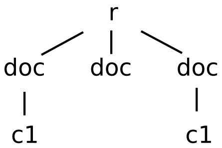
Constraints on updates

r	$\longrightarrow$	(draft | paper)*
draft	$\longrightarrow$	c1? | c2?
paper	$\longrightarrow$	c1.c2



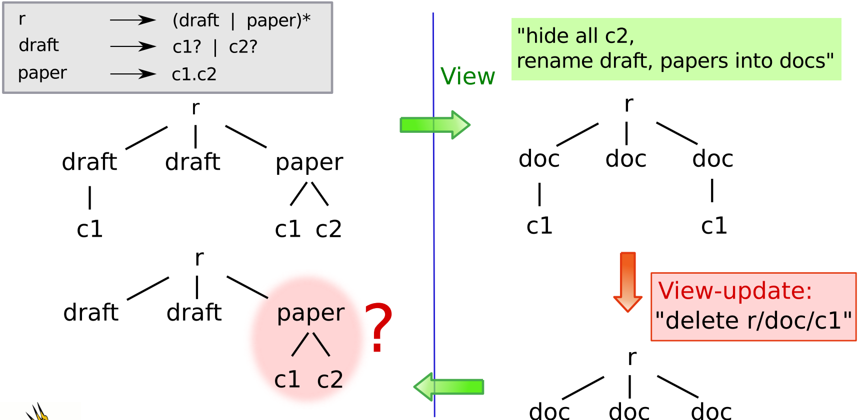
View

"hide all c2,
rename draft, papers into docs"



" *No paper shall revert to draft status!* "

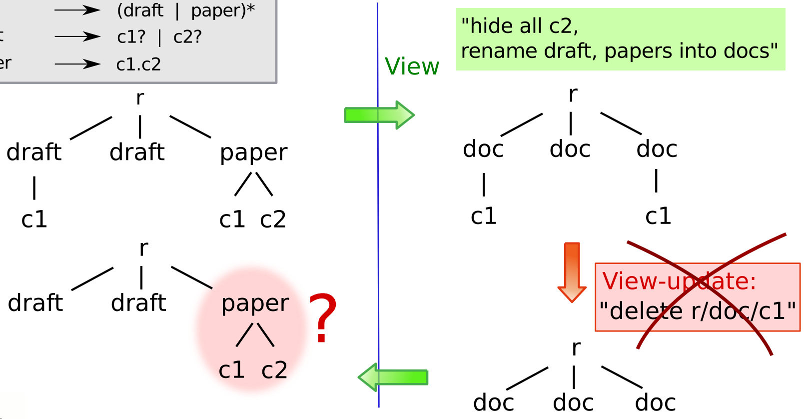
Constraints on updates



" *No paper shall revert to draft status!* "

Constraints on updates

r $\longrightarrow$ (draft | paper)*
draft $\longrightarrow$ c1? | c2?
paper $\longrightarrow$ c1.c2



" *No paper shall revert to draft status!* "

Translatability with constraints

We fix a view V, and a regular set of Ed.S. $\mathcal{U}_s$ representing authorized source updates.

Change from unconstrained setting: u_v may have a translation from some source document and have no translation from another source document.
$\Rightarrow$: *leads to unacceptable behaviour from the user's point of view*

Definition

f_v is *translatable* iff $\exists L \subseteq \mathcal{U}_s . L \circ V \sim_{eq} V \circ f_v$.

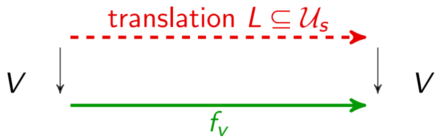


Figure: View update propagation: a synopsis.

The problems we want to solve

Problem: [Test translatability]

Input: view V, set of authorized source updates $\mathcal{U}_s$, update function f_v
Question: *Is f_v translatable?*

Problem: [Compute translatable updates]

Input: view V, set of authorized source updates $\mathcal{U}_s$
Output: *a regular expression for the set of all translatable view updates.*

The problems we want to solve

Problem: [Test translatability]

Input: view V, set of authorized source updates $\mathcal{U}_s$, update function f_v
Question: *Is f_v translatable?*

Proposition[Testing translatability]

In general, testing translatability of f_v is undecidable.

Problem: [Compute translatable updates]

Input: view V, set of authorized source updates $\mathcal{U}_s$
Output: *a regular expression for the set of all translatable view updates.*

Proposition[Computing the translatable updates]

In general, emptiness for the set of translatable updates is undecidable

K-synchronized updates

Definition

An editing script t is k-synchronized if for every sequence $n_1 \ldots, n_{k+1}$ of following siblings labeled with an insertion tag ($\{\varepsilon\} \times \Sigma$), there exists a node n between n_1 and n_{k+1} such that n is tagged with a relabeling ($\Sigma \times \Sigma$).

Example:

$$\underset{1}{(\varepsilon, a)}(d, \varepsilon)\underset{2}{(\varepsilon, c)}\underset{3}{(\varepsilon, a)}(b, c)\underset{1}{(\varepsilon, a)}\underset{2}{(\varepsilon, b)}(a, a)$$

$\rightarrow$: 3-synchronized, but not 2-synchronized.

K-synchronized updates

Definition

An editing script t is k-synchronized if for every sequence $n_1 \ldots, n_{k+1}$ of following siblings labeled with an insertion tag $(\{\varepsilon\} \times \Sigma)$, there exists a node n between n_1 and n_{k+1} such that n is tagged with a relabeling $(\Sigma \times \Sigma)$.

Example:

$$\underset{1}{(\varepsilon, a)}(d, \varepsilon)\underset{2}{(\varepsilon, c)}\underset{3}{(\varepsilon, a)}{\color{red}(b, c)}\underset{1}{(\varepsilon, a)}\underset{2}{(\varepsilon, b)}{\color{red}(a, a)}$$

$\rightarrow$: 3-synchronized, but not 2-synchronized.

Proposition

We can test in polynomial time whether $\exists k.L$ is k-synchronized or not

Applications

Lemma

Fix $k \in \mathbb{N}$. Given a regular set L of k-synchronized editing scripts, $[L]_{eq} = \{t \mid \exists t' \in L. t \sim_{eq} t'\}$ is a regular set of k-synchronized editing scripts.

Proposition[Testing translatability]

Testing translatability of a regular set f_v of k-synchronized editing scripts is decidable.

Proposition

When f_v is a k-synchronized update function, we can compute a translation.

Theorem[Compute translatable updates]

When the set of authorized updates is such that the updates it induces on the view ($V^{-1} \circ \mathcal{U}_s \circ V$) are k-synchronized, we can compute an automaton for the set of all translatable view updates.

Conclusion

	No constraints	General $\mathcal{U}_s$	$\mathcal{U}_s$ & k-synchro. restrict
Compute translation	PTIME	NO	YES if translatable
Testing translation	PTIME	undecidable	decidable
Test transl.		undecidable	decidable
Compute translatable updates		NO	decidable

User-friendliness:

- view can be defined via annotated DTD/XPath annotations...
- editing script can express a small fragment of XQUF; of the form: "Apply atomic update to all nodes selected by XPath query"

Future work

Efficiency:

- how to efficiently evaluate non-deterministic transducers?
- defining a notion of locality

Study independence of updates, commutativity, reversibility.