Translating Updates applied on Views

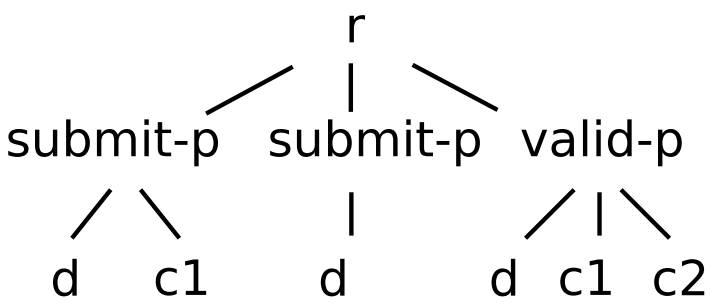
I.Boneva, A-C. Caron, B.Groz, Y.Roos, S.Staworko, S.Tison

LIFL, Université Lille 1

October 11, 2010

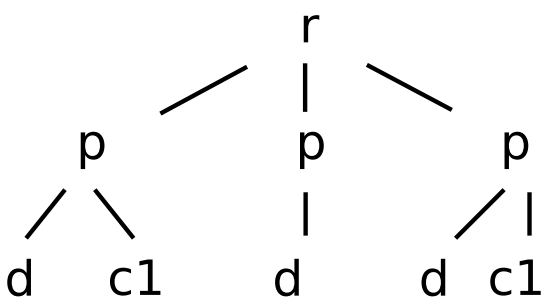
Outline

Introduction: example



View

"hide all c2,
rename xxx-p into p"



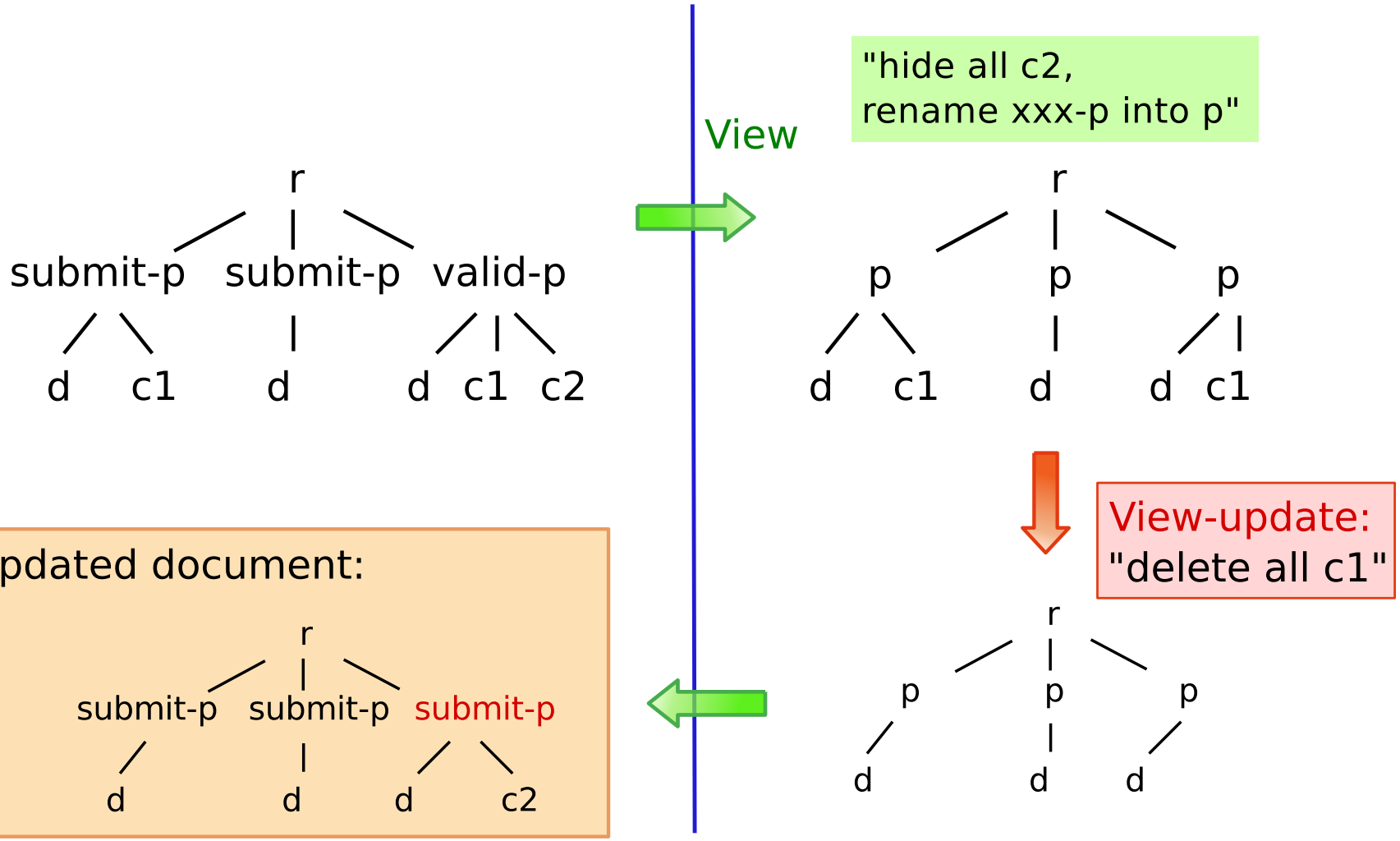
Schema:

submit-p	⟶	d + d.c1 + d.c2
valid-p	⟶	d.c1.c2

View-Schema:

p	⟶	d + d.c1

Introduction: example



The *View-Update* pb

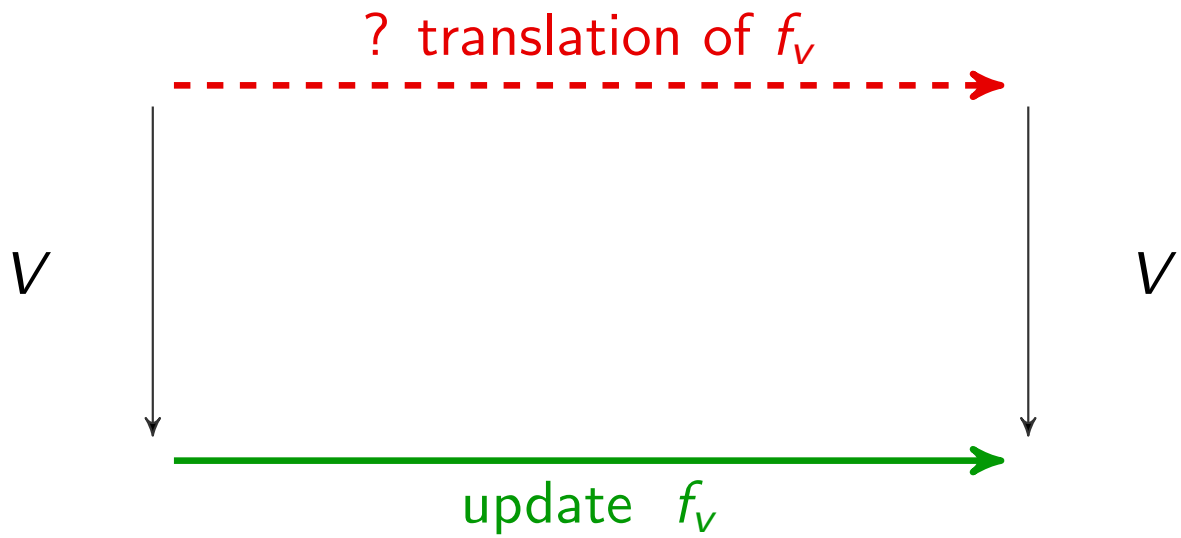


Figure: View update propagation: a synopsis.

The *View-Update* pb

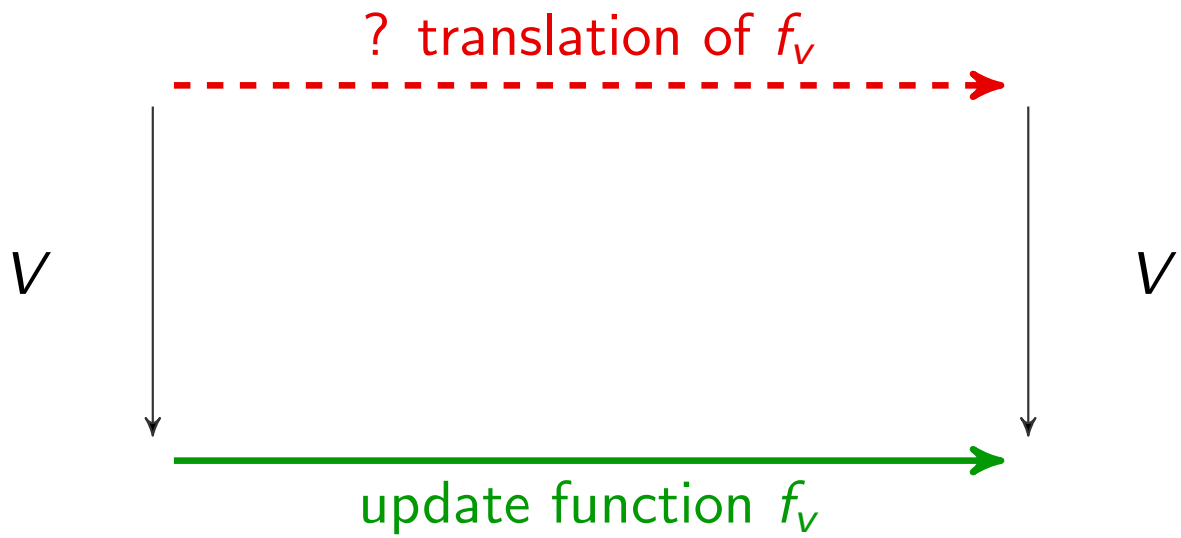


Figure: View update propagation: a synopsis.

The problems we study

In the unconstrained case

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?*

The problems we study

In the unconstrained case

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?*

Constraints on source updates, translatability depend on the source document.
A view update is *uniformly translatable* if one can find an authorized translation.

Problem: [Test unif. translatability]

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

Problem: [Compute uniform updates]

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

Outline

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.

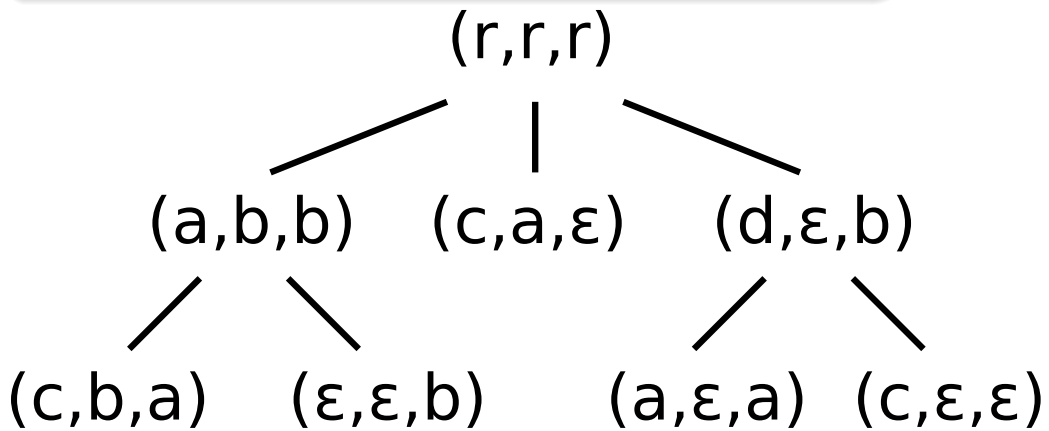


Figure: A tree alignment with $k = 3$

Projections

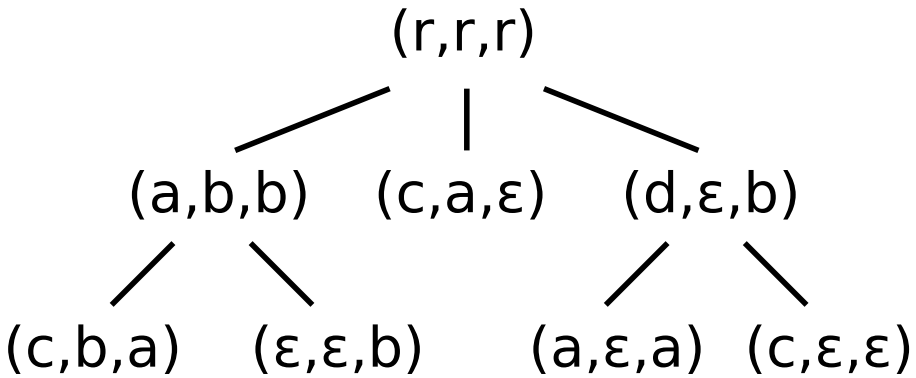


Figure: tree alignment t

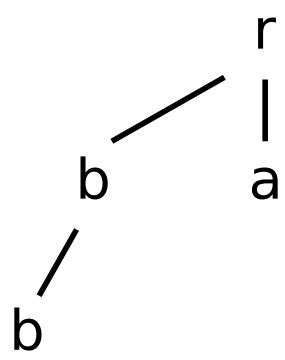


Figure: projection $\pi_3(t)$

Projections

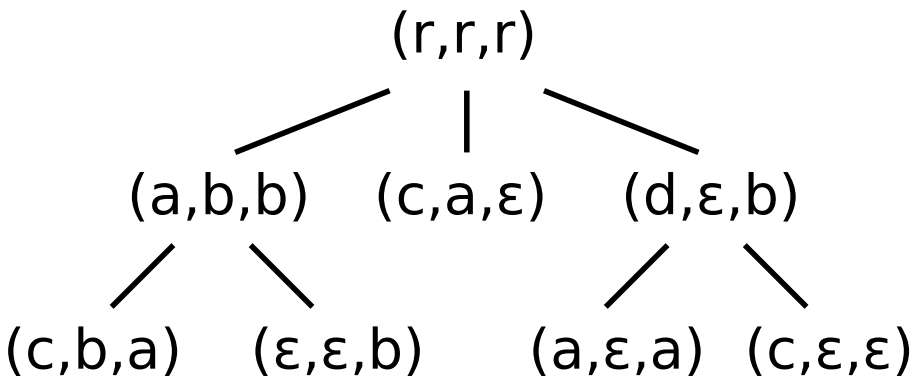


Figure: tree alignment t

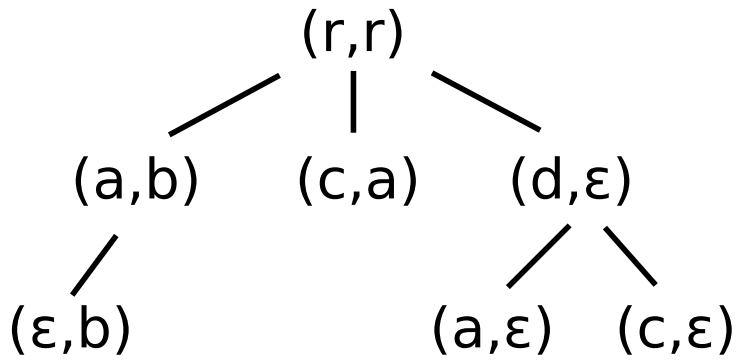


Figure: projection $\pi_{1,3}(t)$

Projections

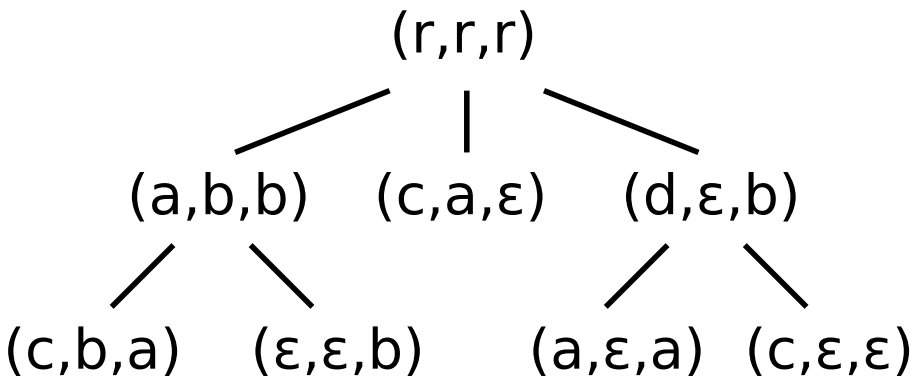


Figure: tree alignment t

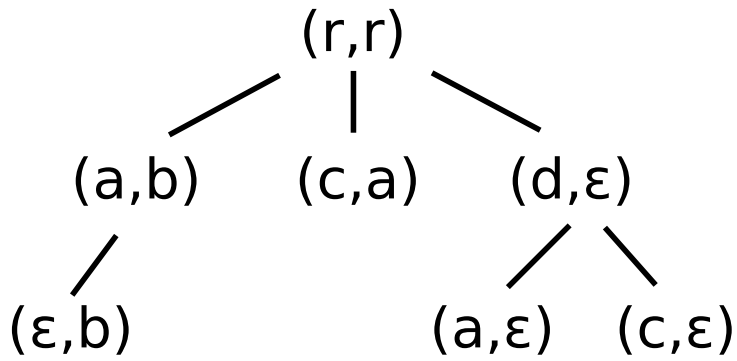


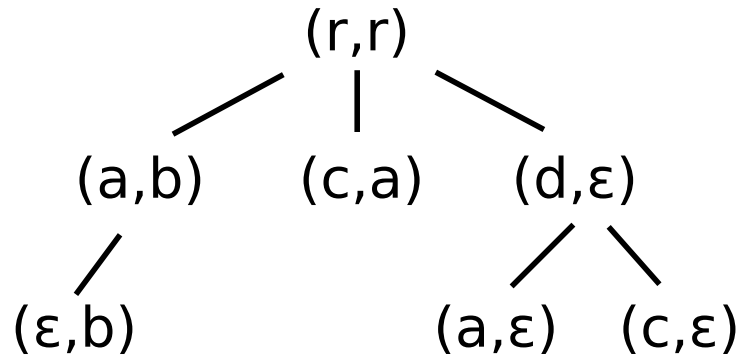
Figure: projection $\pi_{1,3}(t)$

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.

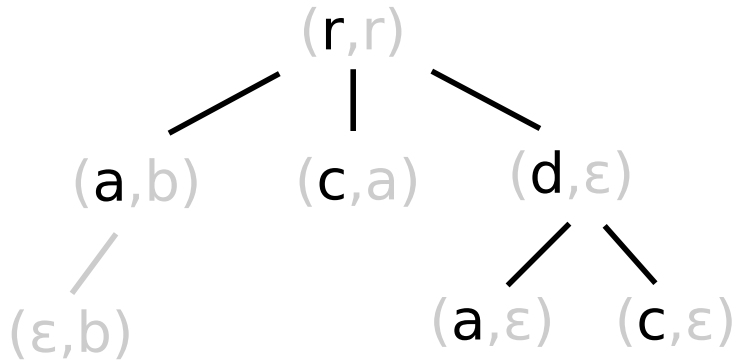


⇒: right subtree was deleted, leaf b was inserted, some nodes were relabeled

Updates=editing scripts

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

Input=$\pi_1(t)$.

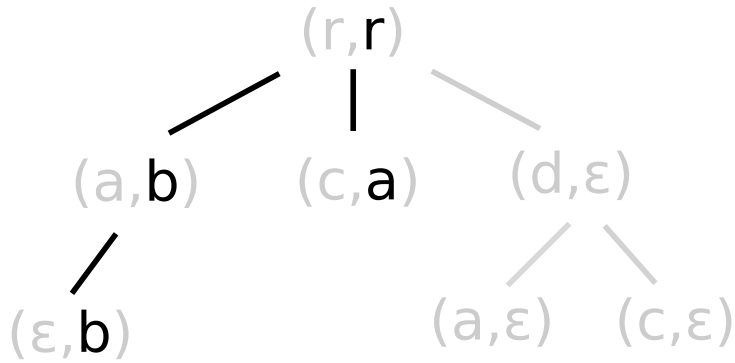


$\Rightarrow$: right subtree was deleted, leaf b was inserted, some nodes were relabeled

Updates=editing scripts

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

Output=$\pi_2(t)$.

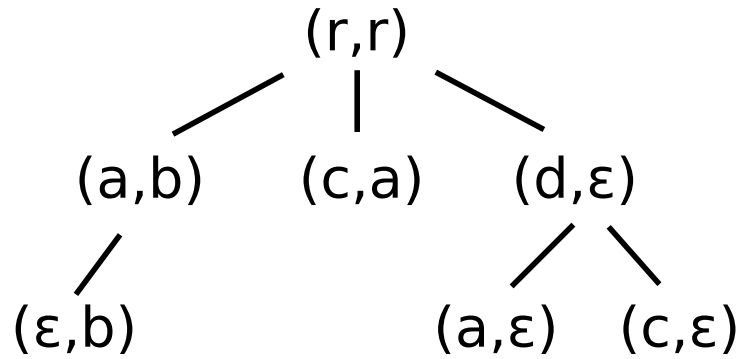


$\Rightarrow$: right subtree was deleted, leaf b was inserted, some nodes were relabeled

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

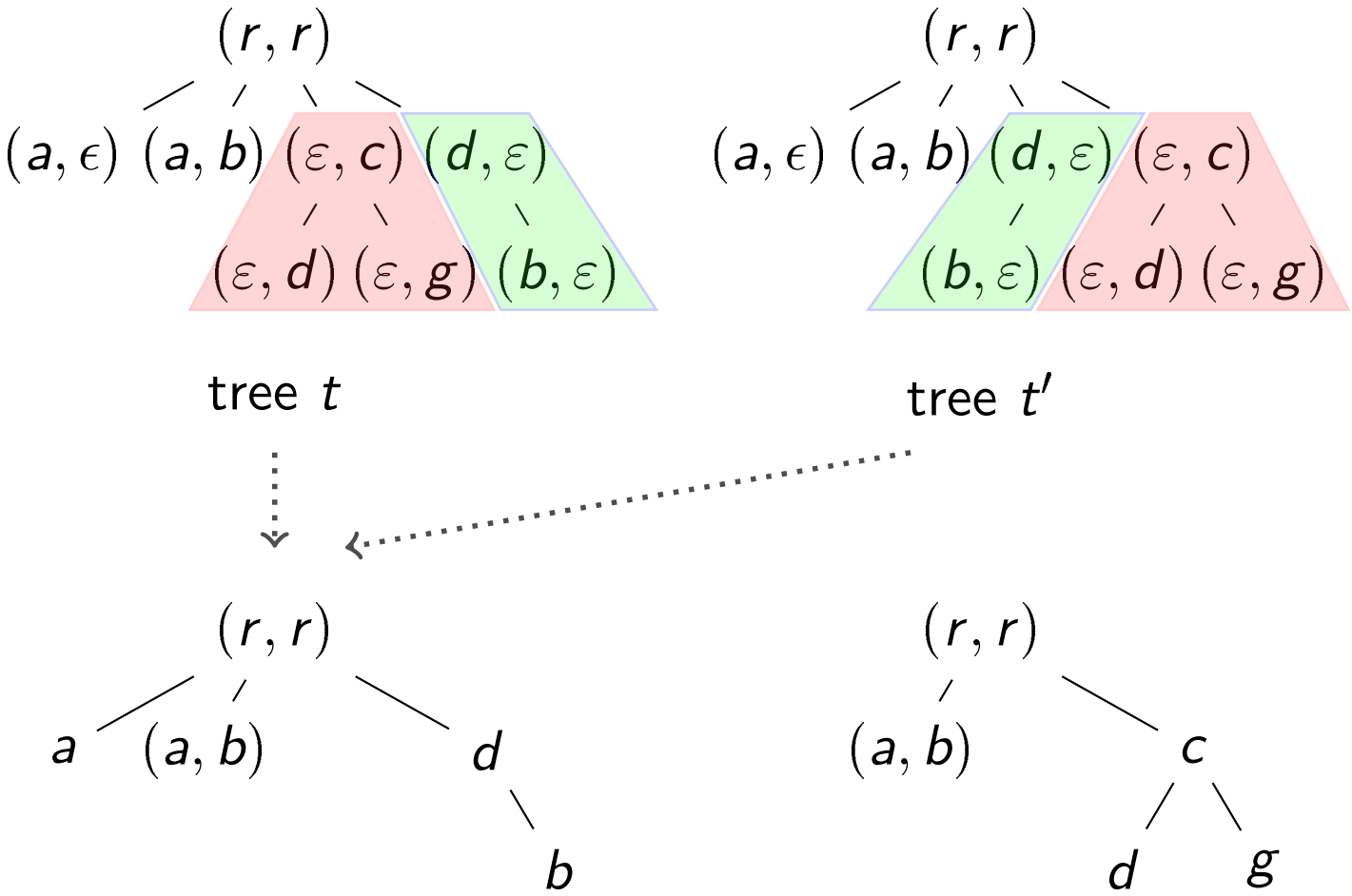


Figure: $t \sim_{eq} t'$

Equivalence of editing scripts

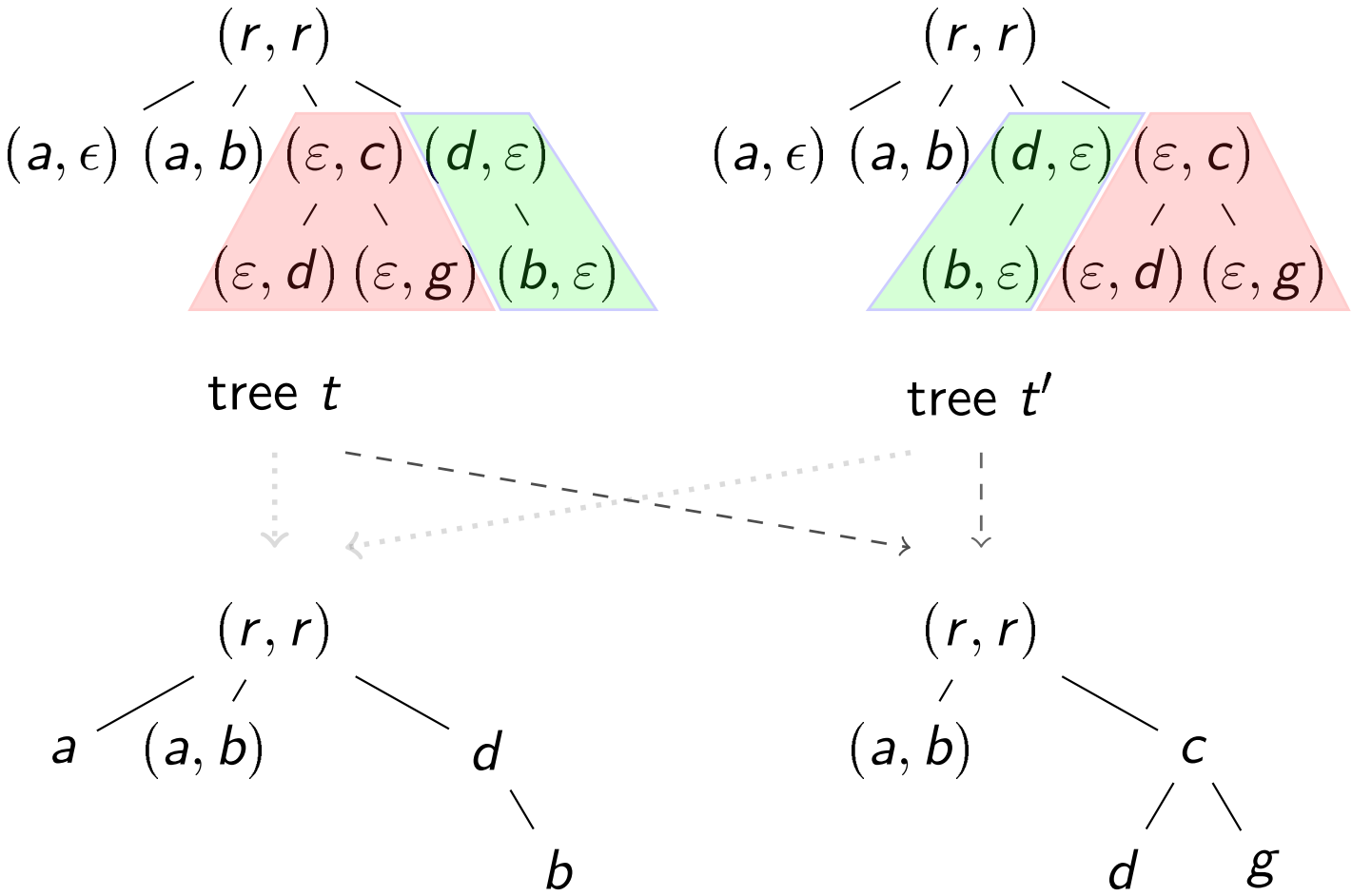


Figure: $t \sim_{eq} t'$

Equivalence of editing scripts

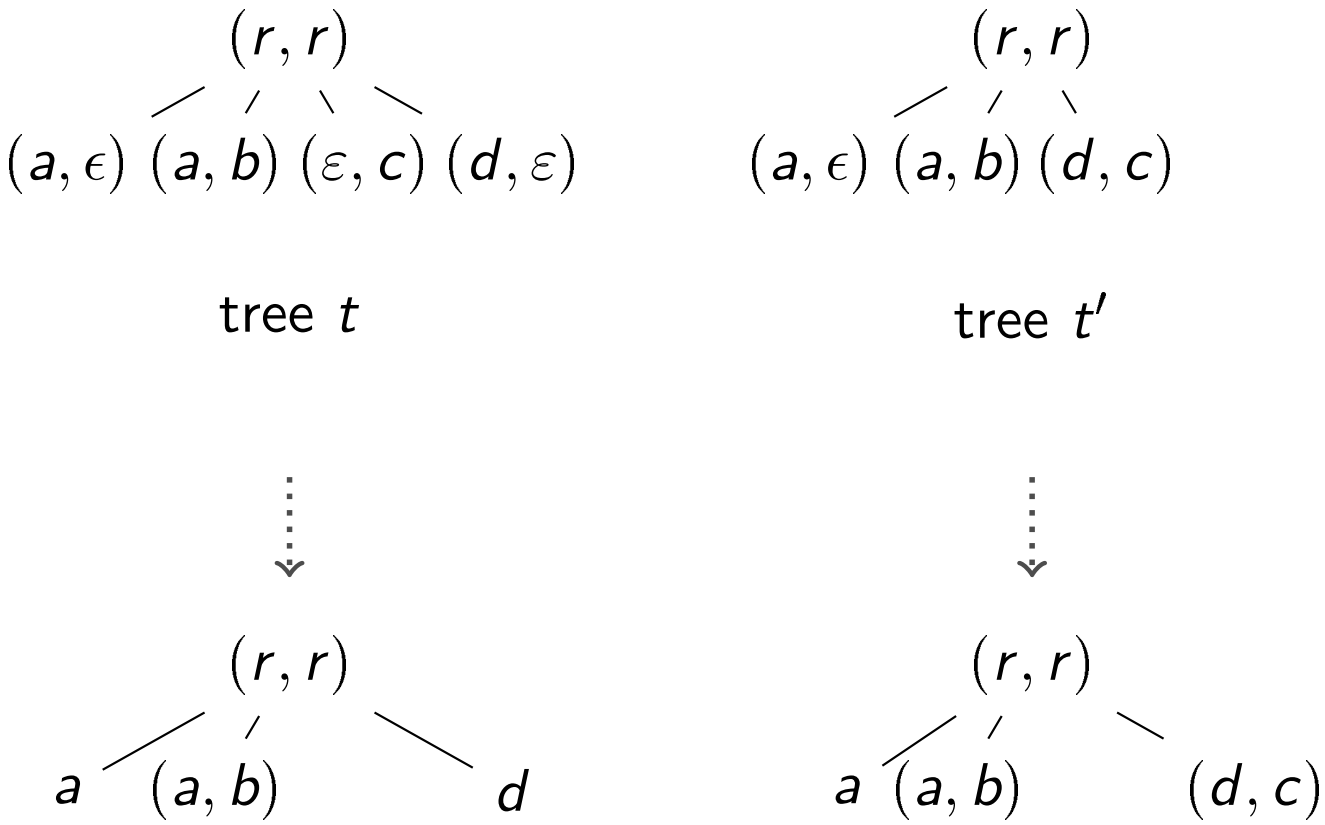


Figure: $t \not\sim_{eq} t'$

Equivalence of editing scripts

Weakness of the model

Given a regular language L of editing scripts, $[L]_{eq} = \{t \mid \exists t' \in L.t \sim_{eq} t'\}$ needs not be regular.
($\Rightarrow$ not even context free 'horizontally')

Example

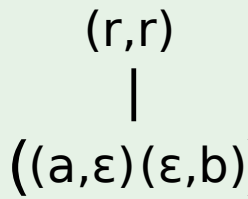


Figure: $[((a, \varepsilon)(\varepsilon, b))^*]_{eq} = \{w \mid |w|_{(a,\varepsilon)} = |w|_{(\varepsilon,b)}\}$: not regular

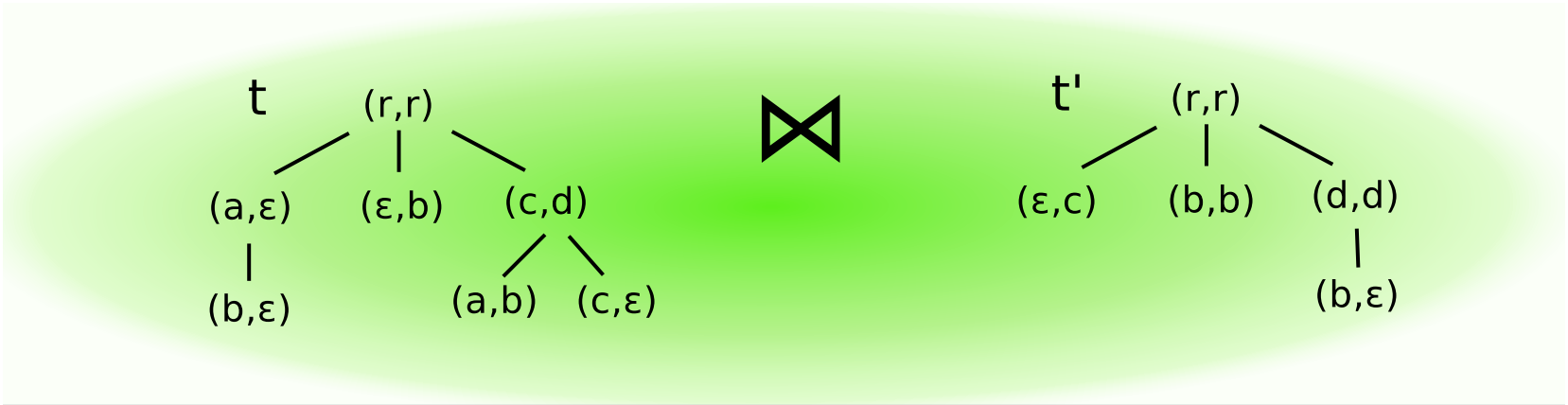
for $L = ((a, \varepsilon)(\varepsilon, b))^* ((c, \varepsilon)(\varepsilon, d))^*$, $[L]_{eq}$ is not even context-free.

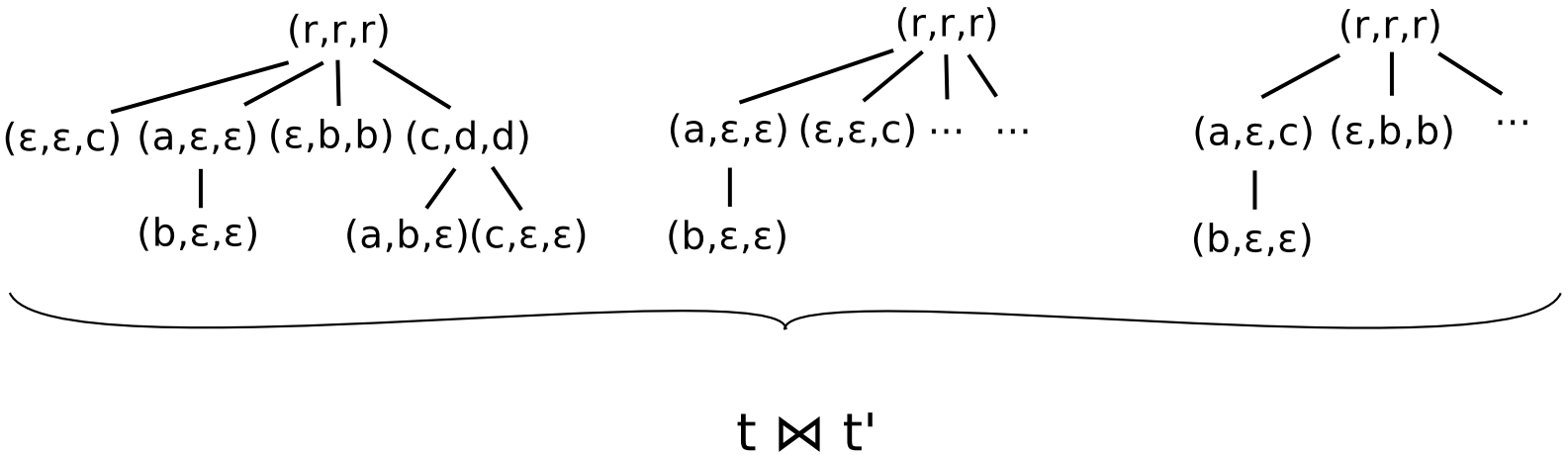
Synchronization

Definition: synchronization

L_1, L_2 sets of editing scripts.

$L_1 \bowtie L_2 = \{t \mid \pi_{1,2}(t) \in L_1 \wedge \pi_{2,3}(t) \in L_2\}$





t ⋈ t'

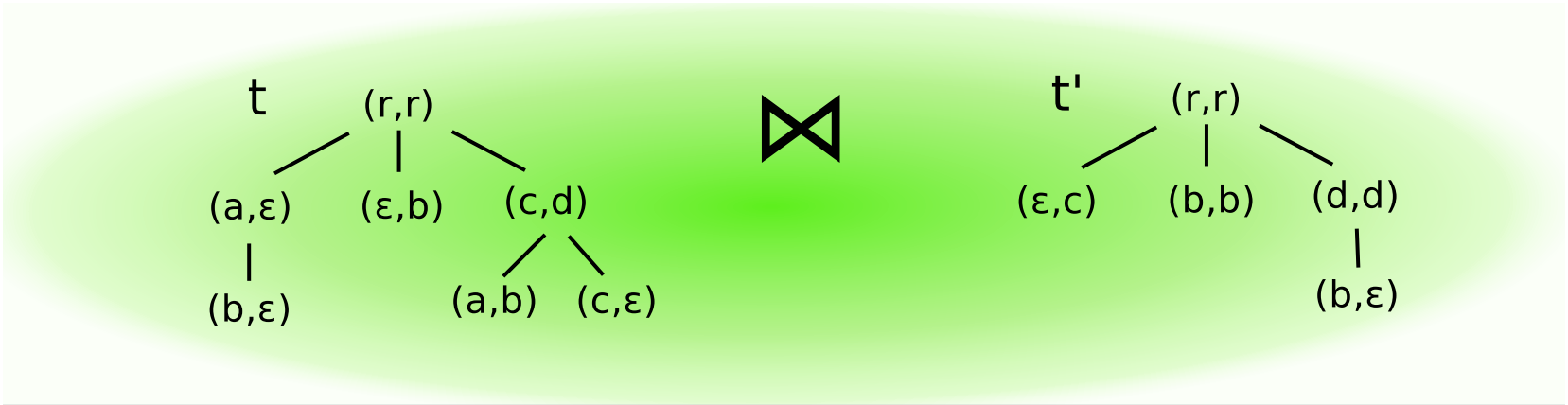
Synchronization

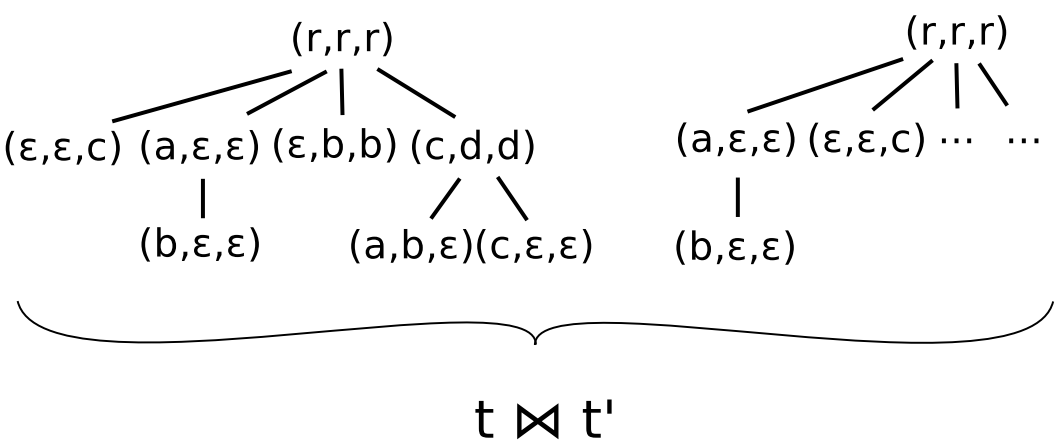
Definition: synchronization

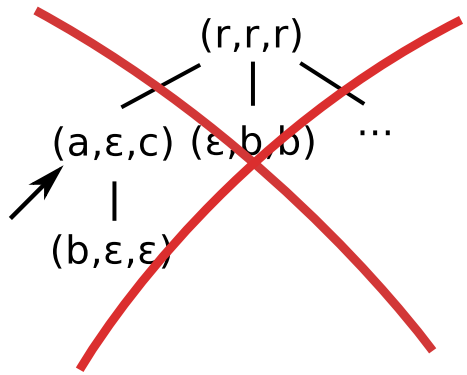
L_1, L_2 sets of editing scripts.

$L_1 \bowtie L_2 = \{t \mid \pi_{1,2}(t) \in L_1 \wedge \pi_{2,3}(t) \in L_2\} \cap L_{\cancel{\Sigma \times \{\varepsilon\} \times \Sigma}}$

... "deletions are forever" !



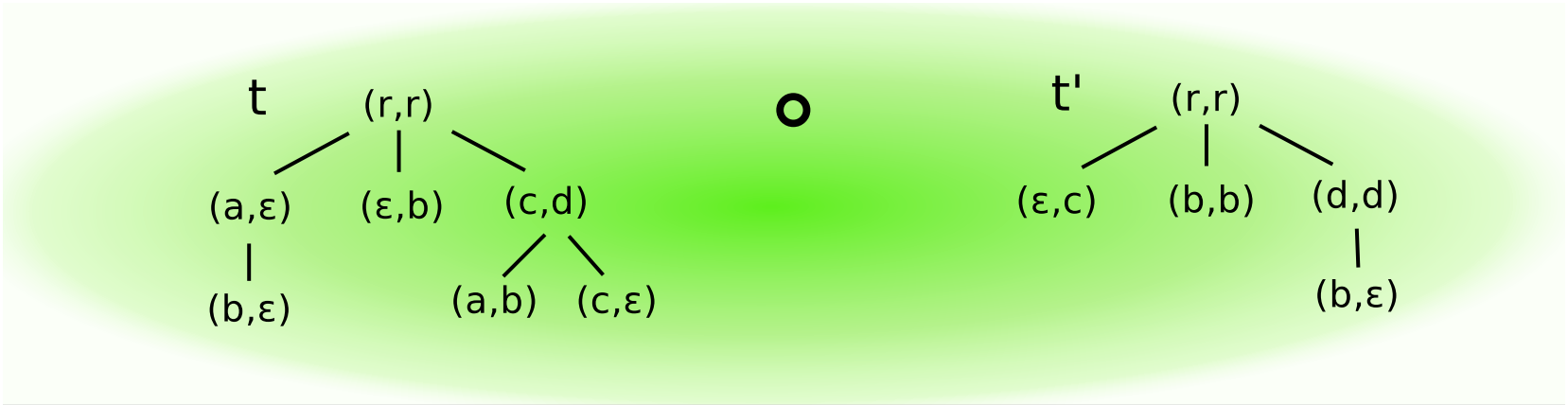


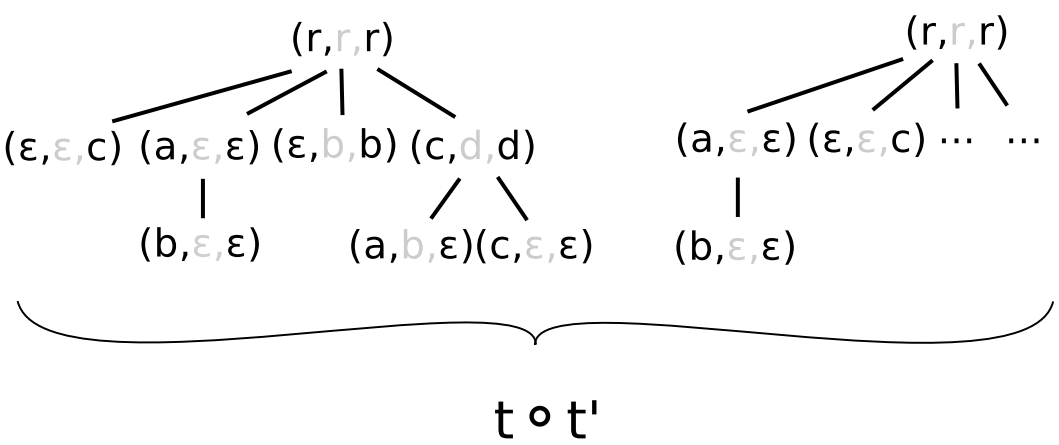


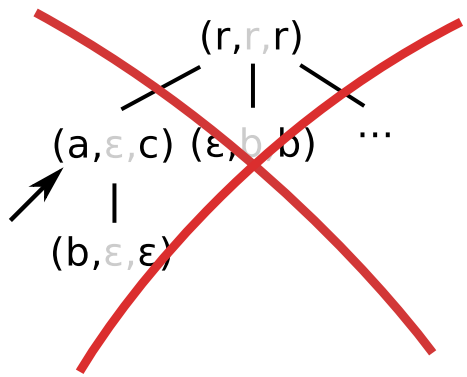
Synchronization

Definition: composition

L_1, L_2 sets of editing scripts.
$L_1 \circ L_2 = \pi_{1,3}(L_1 \bowtie L_2)$
... "deletions are forever" !







Properties of synchronizations

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,
- synchronizations $L_1 \bowtie L_2 \bowtie \ldots \bowtie L_k$: editing scripts $\mapsto$ set of alignments.
  (and composition $L_1 \circ L_2 \circ \ldots \circ L_k$: editing scripts $\mapsto$ sets 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,
- synchronizations $L_1 \bowtie L_2 \bowtie \ldots \bowtie L_k$: editing scripts $\mapsto$ set of alignments.
  (and composition $L_1 \circ L_2 \circ \ldots \circ L_k$: editing scripts $\mapsto$ sets of editing scripts)

Those operations preserve regularity.

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

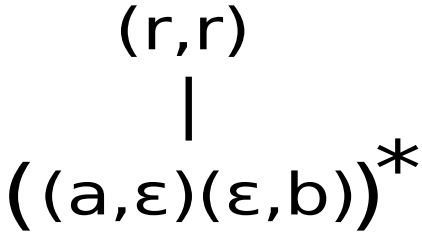
The closure under equivalence does not preserve regularity.

Update function

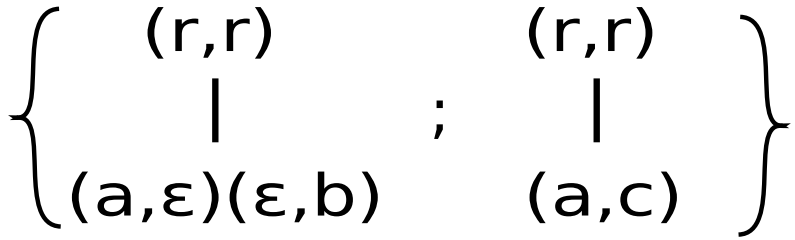
Definition

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

Testing functionality

Proposition

Given a regular set L of editing scripts, we can test in polynomial time whether L is *functional*, i.e. whether it is an update function

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 ($\pi_1(L) = \pi_1(f_L)$).

Outline

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?*

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.

Defining properly translations

Definition

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

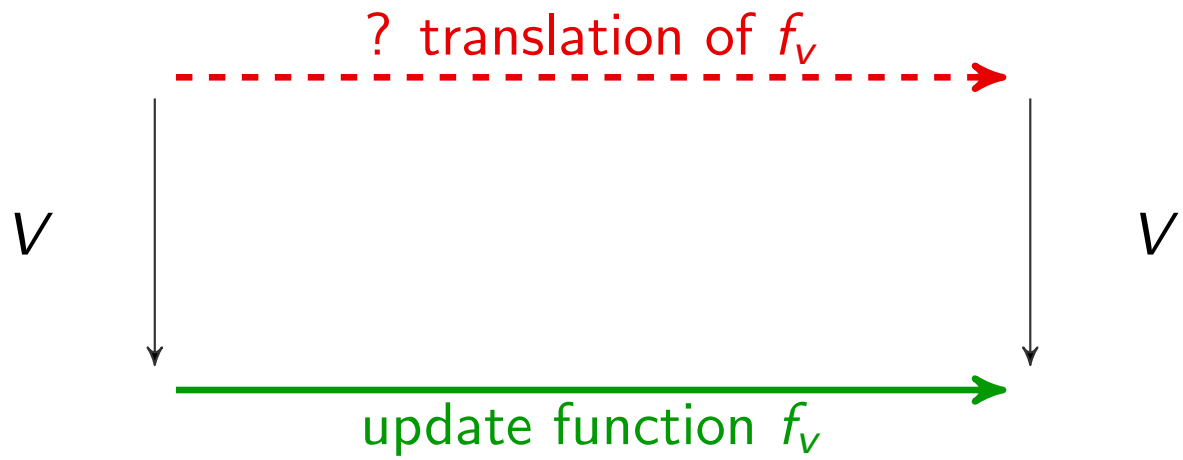


Figure: View update propagation: a synopsis.

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.

Corollary

In the unconstrained setting, an update function f_v has a translation iff $\pi_2(f_v) \subseteq \pi_2(V)$.

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.

Corollary

In the unconstrained setting, an update function f_v has a translation iff $\pi_2(f_v) \subseteq \pi_2(V)$.

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 must check that $L_s \circ V \cup V \circ f_v$ is an update function, which can be achieved in PTIME once equality of the domains is checked.

Outline

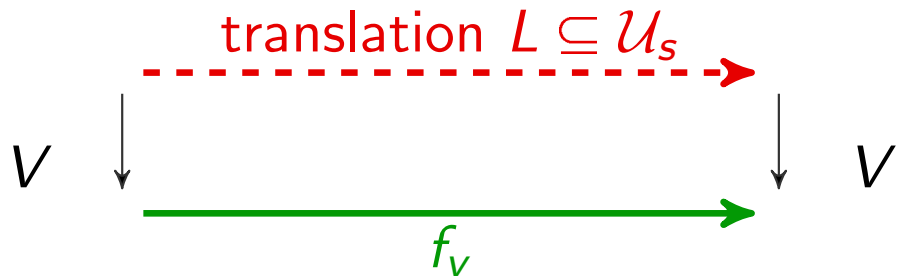
Uniform translatability

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

What changes from the unconstrained setting is: 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

- u_v is *uniformly translatable* (w.r.t. V and $\mathcal{U}_s$) iff $\exists L \subseteq \mathcal{U}_s . L \circ V \sim_{eq} V \circ u_v$.
- f_v is *uniformly translatable* iff it consists in uniform view updates (or equivalently, iff $\exists L \subseteq \mathcal{U}_s . L \circ V \sim_{eq} V \circ f_v$).



The problems we want to solve

Problem: [Test unif. translatability]

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

Problem: [Compute uniform updates]

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

The problems we want to solve

Problem: [Test unif. translatability]

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

Proposition[Testing uniform translatability]

In general, testing uniform translatability of f_v is undecidable.

Problem: [Compute uniform updates]

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

Proposition[Computing the uniform updates]

In general, emptiness for the set of uniform 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:

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

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

Proposition

We can test in polynomial time whether $\exists k.L$ is k-synchronised 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 uniform translatability]

Testing uniform 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 uniform 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 uniform view updates.

Conclusion

	No constraints	General $\mathcal{U}_s$	$\mathcal{U}_s$ & k-synchro. restrictions
Compute translation	PTIME	NO	YES if translatable
Testing translation	PTIME	undecidable	decidable
Test uniform. transl.	~~><~~	undecidable	decidable
Compute unif. 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.