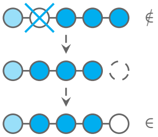
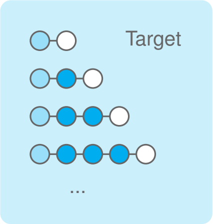
What do you do
if a computational object fails a specification?



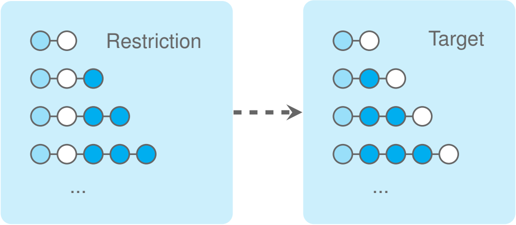


We have studied this problem over words:

1. “Regular repair of specifications”, in LICS 2011.
2. “The cost of traveling between languages”, in ICALP 2011.

We study this problem over XML Documents (trees).

What do you do if a computational object fails a specification?



We have studied this problem over words:

1. “Regular repair of specifications”, in LICS 2011.
2. “The cost of traveling between languages”, in ICALP 2011.

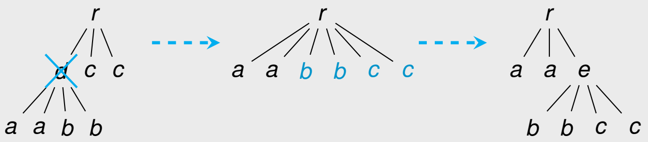
We study this problem over XML Documents (trees).

Can we repair each XML document with an uniformly bounded number of modifications?

Bounded Repair Problem

Example

R:	r → d c*	*T*:	r → a* e
	d → a* b*		e → b* c*
	a → EMPTY		a → EMPTY
	b → EMPTY		b → EMPTY
	c → EMPTY		c → EMPTY



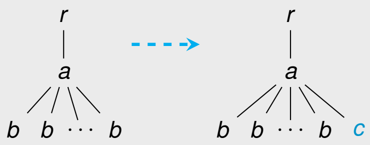
Can we repair each XML document with an uniformly bounded number of modifications?

Bounded Repair Problem

Example

R': r → a
a → b*
b → EMPTY

T': r → a
a → b*, c
b → EMPTY
c → EMPTY



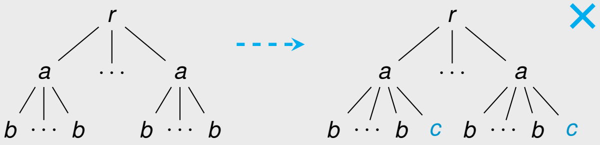
Can we repair each XML document with an uniformly bounded number of modifications?

Bounded Repair Problem

Example

R': r → a*
a → b*
b → EMPTY

T': r → a*
a → b*, c
b → EMPTY
c → EMPTY



Can we repair each XML document with an uniformly bounded number of modifications?

Bounded Repair Problem

Example

R'':

r → a, d
a → a | EMPTY
d → b, c*
b → a
c → EMPTY

T'':

r → d, c*
d → a, a
a → a | b
b → EMPTY
c → EMPTY

? ? ? ?

We give an effective characterization for bounded repairability for every pair of regular tree languages

1. Effective characterization based on:
   - strongly connected components and
   - tree representation for the cyclic behavior of tree automata.
2. Decidability of the bounded repair problem.
   - Between *EXPTIME* and Π_2^{EXP}.
   - Complexity analisys for other subcases.

Bounded repairability for regular tree languages

Cristian Riveros
University of Oxford

Gabriele Puppis
CNRS/LaBRI Bordeaux

Slawek Staworko
University of Lille

ICDT 2012

Outline

Problem definition

Characterization tools

Characterization and proof

Concluding remarks

Outline

Problem definition

Characterization tools

Characterization and proof

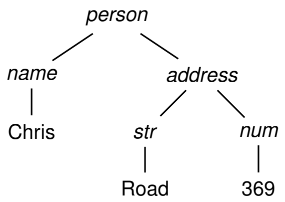
Concluding remarks

Trees and regular tree languages

XML Documents

```
<person>
   <name> Chris </name>
   <address>
      <str> Road </str>
      <num> 369 </num>
   </address>
</person>
```

Unranked trees over Σ



XML Schemas: D

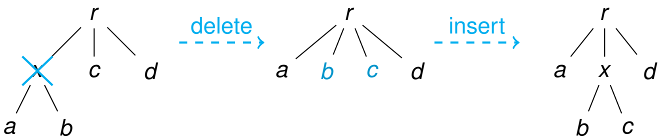
$\mathcal{L}(D) = \{t \in \text{XML} \mid t \models D\}$

Unranked tree automata: T

$\mathcal{L}(T) = \{t \in \text{Trees}(\Sigma) \mid T \text{ accepts } t\}$

Edit operations over trees

Edit operations: deletion, insertion, and relabeling.



All operations have equal cost.

Definition

For trees t, t' and tree language T:

$$\mathrm{dist}(t, t') = \text{shortest sequence of operations that transform } t \text{ into } t'$$

$$\mathrm{dist}(t, T) = \min_{t' \in T} \{ \mathrm{dist}(t, t') \}$$

Bounded repair problem

Definition

Given unranked tree automata $\mathcal{R}$ (restriction) and $\mathcal{T}$ (target), determine if there exists a uniform bound $N \in \mathbb{N}$ such that:

$$\mathrm{dist}(t, L(\mathcal{T})) \leq N \quad \text{for all } t \in L(\mathcal{R})$$

Generalization of language containment.

Outline

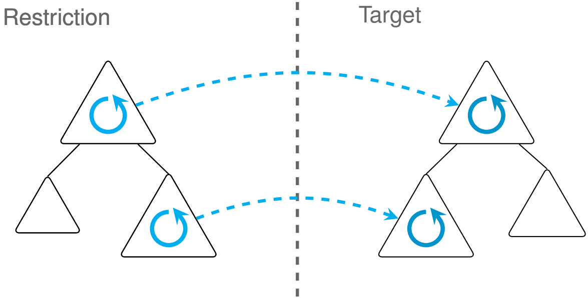
Problem definition

Characterization tools

Characterization and proof

Concluding remarks

How to repair trees? (intuition)



1. Cyclic behavior:
   - Stepwise tree automata over curry encoding of trees.
   - Strongly connected components of stepwise tree automata.
   - Tree representation of cyclic behavior (Synopsis trees).

2. Mapping:
   - Covering relation between synopsis trees.

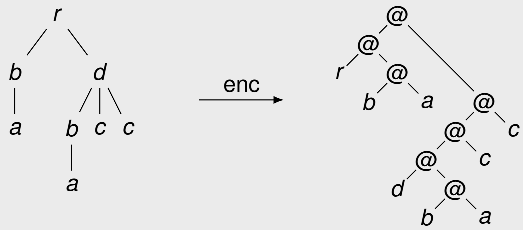
Curry encoding

Definition

The curry encoding of an unranked tree over Σ is a complete binary tree that has two types of nodes:

- Internal nodes: @.
- Leaf nodes: Σ.

Example

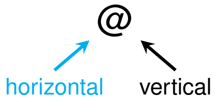


Curry encoding

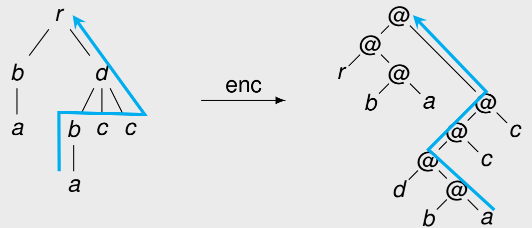
Definition

$$\mathrm{enc}(a) = a$$

$$\mathrm{enc}\left(\begin{array}{ccc} & a & \\ t_1 & \cdots & t_n \end{array} \right) = @\left(\mathrm{enc}\left(\begin{array}{ccc} & a & \\ t_1 & \cdots & t_{n-1} \end{array} \right), \mathrm{enc}(t_n) \right)$$



Example



Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

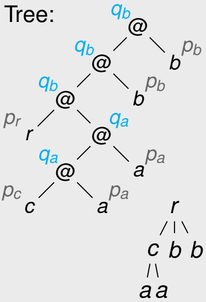
1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

Example

R: r → c b*
c → a^+
a → EMPTY
b → EMPTY

$$\mathcal{R}: \begin{aligned} \delta(p_c, p_a) &\to q_a \\ \delta(q_a, p_a) &\to q_a \\ \delta(p_r, q_a) &\to \underline{q_b} \\ \delta(\underline{q_b}, p_b) &\to \underline{q_b} \\ \delta_0(a) &\to p_a \\ \delta_0(b) &\to p_b \\ \delta_0(c) &\to p_c \\ \delta_0(r) &\to p_r \end{aligned}$$

Tree:



Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

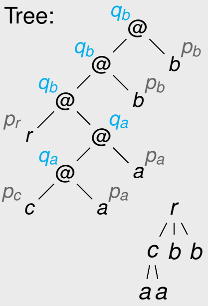
1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

Example

R: r → c b*
c → a^+
a → EMPTY
b → EMPTY

$$\mathcal{R}: \quad \begin{aligned} \delta(c, a) &\to q_a \\ \delta(q_a, a) &\to q_a \\ \delta(r, q_a) &\to \underline{q_b} \\ \delta(\underline{q_b}, b) &\to \underline{q_b} \end{aligned}$$

Tree:



Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

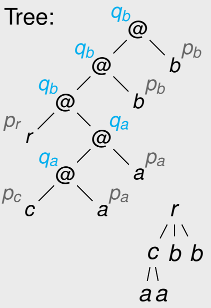
1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

Example

R: r → c b*
c → a^+
a → EMPTY
b → EMPTY

$$\mathcal{R}: \quad \begin{aligned} c\,@\,a &\to q_a \\ q_a\,@\,a &\to q_a \\ r\,@\,q_a &\to \underline{q_b} \\ \underline{q_b}\,@\,b &\to \underline{q_b} \end{aligned}$$

Tree:



Stepwise tree automata

Definition

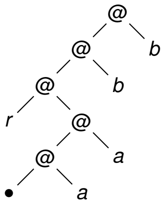
A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

- $L(\mathcal{A}) = \{t \in \text{Trees}(\Sigma) \mid$ $\exists$ an accepting run of $\mathcal{A}$ over $t\}$.
- contexts.
- concatenation between contexts:

$$C_1 \circ C_2.$$

- run of $\mathcal{A}$ on a context C from q.



Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

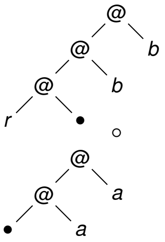
- $L(\mathcal{A}) = \{t \in \mathrm{Trees}(\Sigma) \mid$
  $\exists$ an accepting run of $\mathcal{A}$ over $t\}$.

- contexts.

- concatenation between contexts:

$$C_1 \circ C_2.$$

- run of $\mathcal{A}$ on a context C from q.



Stepwise tree automata

Definition

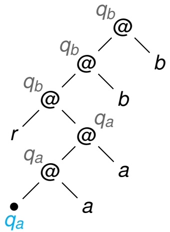
A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

- $L(\mathcal{A}) = \{t \in \text{Trees}(\Sigma) \mid$
  $\exists$ an accepting run of $\mathcal{A}$ over $t\}$.
- contexts.
- concatenation between contexts:

$$C_1 \circ C_2.$$

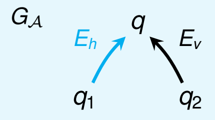
- run of $\mathcal{A}$ on a context C from q.



Cyclic behavior of stepwise automata (components)

Definition

Given $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$, the transition graph of $\mathcal{A}$ is the graph $G_{\mathcal{A}} = (Q, E_h \cup E_v)$ such that for every $q \in \delta(q_1, q_2)$:

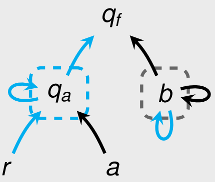


- SCC$(\mathcal{A})$ is the set of strongly connected component X of $G_{\mathcal{A}}$.
- $L(\mathcal{A} \mid X) = \{C \in \text{context}_\Sigma \mid \exists p, q \in X : \ q \in \delta(p, C)\}$

Example

$$\begin{array}{lcl} r & \to & a^* \cdot b \\ a & \to & \text{EMPTY} \\ b & \to & b^* \end{array}$$

$$\begin{array}{rcccl} r & @ & a & \to & q_a \\ q_a & @ & a & \to & q_a \\ q_a & @ & b & \to & q_f \\ b & @ & b & \to & b \end{array}$$

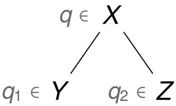


= horizontal, = vertical

Synopsis trees

Definition

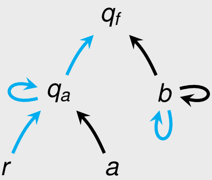
A synopsis tree of $\mathcal{A}$ is a binary tree with labels in $\mathrm{SCC}(\mathcal{A})$ that respect the transition relation of $\mathcal{A}$.



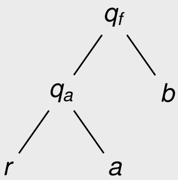
✓ $q \in \delta(q_1, q_2)$

Example

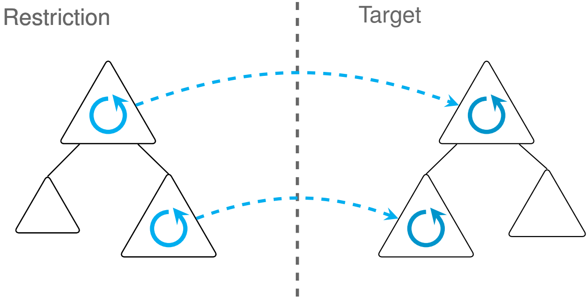
Transition graph of $\mathcal{A}$:



Synopsis tree



How to repair trees? (intuition)



1. Cyclic behavior:
   - Stepwise tree automata over curry encoding of trees.
   - Strongly connected components of stepwise tree automata.
   - Tree representation of cyclic behavior (Synopsis trees).

2. Mapping:
   - Covering relation between synopsis trees.

Coverings

Definition

Given two synopsis trees τ of $\mathcal{R}$ and σ of $\mathcal{T}$, we say that σ covers τ iff there exists a mapping λ from nodes of τ to nodes of σ:

1. λ preserves language containment of components,

$$L(\mathcal{R} \mid \tau(x)) \subseteq L(\mathcal{T} \mid \sigma(\lambda(x)))$$

2. λ preserves the post-order of nodes,

$$x \preccurlyeq_{\tau}^{\text{post}} y \text{ iff } \lambda(x) \preccurlyeq_{\sigma}^{\text{post}} \lambda(y)$$

3. λ preserves the ancestorship of vertical nodes,

$$x \preccurlyeq_{\tau}^{\text{anc}} y \text{ iff } \lambda(x) \preccurlyeq_{\sigma}^{\text{anc}} \lambda(y) \text{ with } x \text{ a vertical node}$$

for every non-trivial nodes x and y of τ.

Coverings

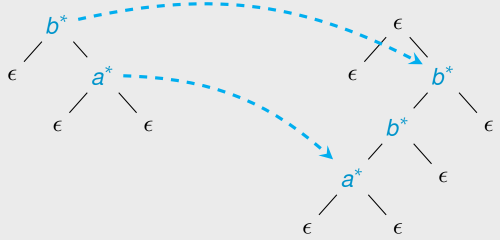
σ covers τ iff there exists a mapping λ from nodes of τ to nodes of σ:

1. λ preserves language containment of components,
2. λ preserves the post-order of nodes, and
3. λ preserves the ancestorship of vertical nodes.

Example

R: r → c b*
c → a*

T: r → d
d → a* b*



Outline

Problem definition

Characterization tools

Characterization and proof

Concluding remarks

Main Characterization

Theorem

$L(\mathcal{R})$ is bounded repairable into $L(\mathcal{T})$ iff every synopsis tree of $\mathcal{R}$ is covered by some synopsis tree of $\mathcal{T}$.

Two directions proof:

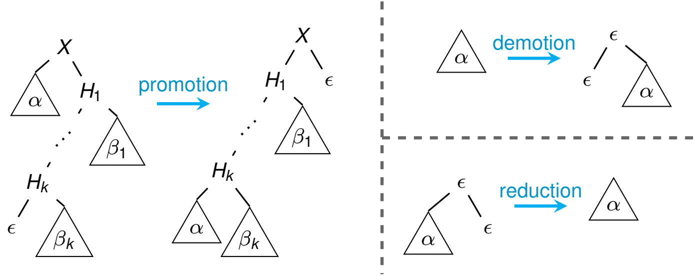
- From repair to covering.
- From covering to repair.

From covering to repair

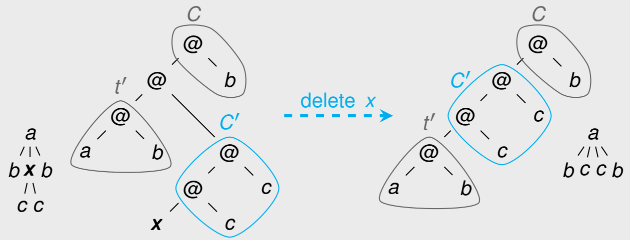
For every tree in $t \in \mathcal{L}(\mathcal{R})$:

1. Run $\mathcal{R}$ and find the synopsis tree τ that represents t.
2. Find a synopsis tree σ in $\mathcal{T}$ that covers τ.
3. Use a set of macro operations over synopsis tree to transform τ into σ.
4. Macro operations over synopsis tree preserves bounded repairability.

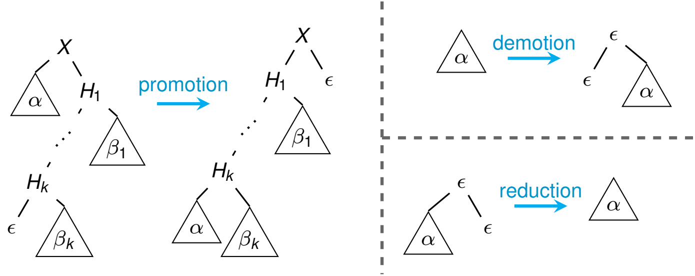
Synopsis tree operations



Remark.



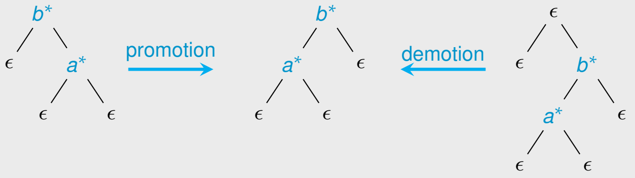
Synopsis tree operations



Example

R: r → c b*
c → a*

T: r → d
d → a* b*



Outline

Problem definition

Characterization tools

Characterization and proof

Concluding remarks

Concluding remarks

- Effective characterization for every pair of regular tree languages.
  - between *EXPTIME* and Π_2^{EXP}
    for stepwise automata.
  - *PSPACE*-hard
    for deterministic DTD.
  - in Π_2^{P}
    for deterministic DTDs with fixed alphabet.

Future work: bounded streaming repair.

Bounded repairability for regular tree languages

Cristian Riveros
University of Oxford

Gabriele Puppis
CNRS/LaBRI Bordeaux

Slawek Staworko
University of Lille

ICDT 2012