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

We have studied this problem over words:


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:


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

\[ R: \quad r \rightarrow d \, c^* \]
\[ d \rightarrow a^* \, b^* \]
\[ a \rightarrow \text{EMPTY} \]
\[ b \rightarrow \text{EMPTY} \]
\[ c \rightarrow \text{EMPTY} \]

\[ T: \quad r \rightarrow a^* \, e \]
\[ e \rightarrow b^* \, c^* \]
\[ a \rightarrow \text{EMPTY} \]
\[ b \rightarrow \text{EMPTY} \]
\[ c \rightarrow \text{EMPTY} \]
Can we repair each XML document with an uniformly bounded number of modifications?

Bounded Repair Problem

Example

\[ R' : \begin{align*}
  r & \rightarrow a \\
  a & \rightarrow b^* \\
  b & \rightarrow \text{EMPTY}
\end{align*} \quad T' : \begin{align*}
  r & \rightarrow a \\
  a & \rightarrow b^*, c \\
  b & \rightarrow \text{EMPTY} \\
  c & \rightarrow \text{EMPTY}
\end{align*} \]
Can we repair each XML document with an uniformly bounded number of modifications?

**Bounded Repair Problem**

**Example**

$$R': \quad r \rightarrow a^*$$
$$a \rightarrow b^*$$
$$b \rightarrow \text{EMPTY}$$

$$T': \quad r \rightarrow a^*$$
$$a \rightarrow b^*, c$$
$$b \rightarrow \text{EMPTY}$$
$$c \rightarrow \text{EMPTY}$$

---

```
      r
     / \  \
    a   ...  a
   /     /     /
  b     b     b
```

```
      r
     /     \
    a   ...  a
   /       /     /
  b     c   b     b
```

---

```
      r
     /     \
    a   ...  a
   /       /     /
  b     b   b     b
```

---

```
      r
     /     \
    a   ...  a
   /       /     /
  b     c   b     b
```
Can we repair each XML document with an uniformly bounded number of modifications?

**Bounded Repair Problem**

**Example**

$R''$:  
- $r \rightarrow a, d$
- $a \rightarrow a$ | EMPTY
- $d \rightarrow b, c^*$
- $b \rightarrow a$
- $c \rightarrow$ EMPTY

$T'':$
- $r \rightarrow d, c^*$
- $d \rightarrow a, a$
- $a \rightarrow a$ | $b$
- $b \rightarrow$ EMPTY
- $c \rightarrow$ 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 $\Pi_2^{EXP}$.
   - Complexity analysis 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 $\Sigma$

```
person
  name
  |   address
  |     str
  |       num
  |         Road
  |           369
```

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 \( T \):

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

\[
\text{dist}(t, T) = \min_{t' \in T} \{ \text{dist}(t, t') \}
\]
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:

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

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 $\Sigma$ is a complete binary tree that has two types of nodes:

- Internal nodes: @.
- Leaf nodes: $\Sigma$.

Example
Curry encoding

Definition

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

$$\text{enc}( \langle t_1 \cdots t_n \rangle ) = (@(\text{enc}( \langle t_1 \cdots t_{n-1} \rangle ), \text{enc}(t_n)))$$

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

\[
\begin{align*}
R: & & r \to c b^* & R: & & \delta(p_c, p_a) \to q_a \\
& & c \to a^+ & & \delta(q_a, p_a) \to q_a \\
& & a \to \text{EMPTY} & & \delta(p_r, q_a) \to q_b \\
& & b \to \text{EMPTY} & & \delta(q_b, p_b) \to q_b \\
& & \delta_0(a) \to p_a \\
& & \delta_0(b) \to p_b \\
& & \delta_0(c) \to p_c \\
& & \delta_0(r) \to p_r \\
\end{align*}
\]

Tree:

```
@ -- r  \\
|   \  \\
|    @  \\
|    |  \\
|    qa @  \\
|    |   \\
|    qa qb b  \\
|    |   \\
|    qa qa  \\
|    |   \\
|    qa qa p_a  \\
|    |   \\
|    qa qa qa  \\
|    |   \\
|    qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa qa qa qa qa qa \\
|    |   \\
|    qa qa qa qa qa qa qa qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa qa qa qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa qa qa qa qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa qa qa qa qa qa qa qa qa qa  \\
|    |   \\
|    qa qa qa qa qa qa qa qa qa qa qa qa qa qa qa qa 
```

[@]
Stepwise tree automata

Definition

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

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

Example

$R: \begin{align*}
  r &\rightarrow c \ b^* \\
c &\rightarrow a^+ \\
a &\rightarrow \text{EMPTY} \\
b &\rightarrow \text{EMPTY}
\end{align*}$

\[ \mathcal{R}: \begin{align*}
  \delta(c, a) &\rightarrow q_a \\
  \delta(q_a, a) &\rightarrow q_a \\
  \delta(r, q_a) &\rightarrow q_b \\
  \delta(q_b, b) &\rightarrow q_b
\end{align*} \]

Tree:
Stepwise tree automata

Definition

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

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

Example

\[
R: \begin{align*}
r &\rightarrow c b^* \\
c &\rightarrow a^+ \\
a &\rightarrow \text{EMPTY} \\
b &\rightarrow \text{EMPTY}
\end{align*}
\]
\[
R: \begin{align*}
c @ a &\rightarrow q_a \\
a @ a &\rightarrow q_a \\
r @ q_a &\rightarrow q_b \\
q_b @ b &\rightarrow q_b
\end{align*}
\]

Tree:
Stepwise tree automata

Definition

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

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

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

contexts.

concatenation between contexts:

$C_1 \circ C_2$.

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

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

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. \( \delta : Q \times Q \rightarrow 2^Q \) is the transition function,
2. \( \delta_0 : \Sigma \rightarrow 2^Q \) is the initial function,
3. \( F \subseteq Q \) is the final set of states.

\[ L(\mathcal{A}) = \{ t \in \text{Trees}(\Sigma) \mid \exists \text{ an accepting run of } \mathcal{A} \text{ over } t \} \]

- **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 $q \in \delta(q_1, q_2)$:

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

Example

$$
\begin{align*}
& r \rightarrow a^* \cdot b & r @ a \rightarrow q_a \\
& a \rightarrow \text{EMPTY} & q_a @ a \rightarrow q_a \\
& b \rightarrow b^* & q_a @ b \rightarrow q_f \\
& & b @ b \rightarrow b
\end{align*}
$$

= horizontal,  = vertical
Synopsis trees

Definition

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

Example

Transition graph of $A$:  

Synopsis tree:

$q \in X \checkmark q \in \delta(q_1, q_2)$

$q_1 \in Y \quad q_2 \in Z$
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 $\tau$ of $\mathcal{R}$ and $\sigma$ of $\mathcal{T}$, we say that $\sigma$ covers $\tau$ iff there exists a mapping $\lambda$ from nodes of $\tau$ to nodes of $\sigma$:

1. $\lambda$ preserves language containment of components,

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

2. $\lambda$ preserves the post-order of nodes,

$$x \preceq_{\tau} y \text{ iff } \lambda(x) \preceq_{\sigma} \lambda(y)$$

3. $\lambda$ preserves the ancestorship of vertical nodes,

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

for every non-trivial nodes $x$ and $y$ of $\tau$. 
Coverings

\(\sigma\) covers \(\tau\) iff there exists a mapping \(\lambda\) from nodes of \(\tau\) to nodes of \(\sigma\):

1. \(\lambda\) preserves language containment of components,
2. \(\lambda\) preserves the post-order of nodes, and
3. \(\lambda\) preserves the ancestorship of vertical nodes.

Example

\[
\begin{align*}
R: & \quad r \rightarrow c \ b^* \\
& \quad c \rightarrow a^* \\
T: & \quad r \rightarrow d \\
& \quad d \rightarrow a^* \ b^*
\end{align*}
\]
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 $\tau$ that represents $t$.

2. Find a synopsis tree $\sigma$ in $\mathcal{T}$ that covers $\tau$.

3. Use a set of macro operations over synopsis tree to transform $\tau$ into $\sigma$.

4. Macro operations over synopsis tree preserves bounded repairability.
Synopsis tree operations

Remark.
Synopsis tree operations

Example

\[ R: \quad r \rightarrow c \, b^* \]
\[ c \rightarrow a^* \]
\[ b^* \quad \text{promotion} \]
\[ a^* \quad \text{promotion} \]
\[ a^* \quad \text{promotion} \]

\[ T: \quad r \rightarrow d \]
\[ d \rightarrow a^* \, b^* \]
\[ a^* \quad \text{promotion} \]
\[ b^* \quad \text{promotion} \]
\[ a^* \quad \text{promotion} \]
Outline

Problem definition

Characterization tools

Characterization and proof

Concluding remarks
Concluding remarks

- Effective characterization for every pair of regular tree languages.
  - between $\text{EXPTIME}$ and $\Pi^\text{EXP}_2$
    for stepwise automata.
  - $\text{PSPACE}$-hard
    for deterministic DTD.
  - in $\Pi^P_2$
    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