Node Query Preservation for Deterministic Linear Top-Down Tree Transducers

Kazuki Miyahara\textsuperscript{1} Kenji Hashimoto\textsuperscript{2} Hiroyuki Seki\textsuperscript{1,2}

\textsuperscript{1}Nara Institute of Science and Technology, Japan
\textsuperscript{2}Nagoya University, Japan

TTATT, Hanoi, October 19, 2013
Query Preservation

Lossless data transformation

- Information in the data source is preserved through data transformation.

Query preservation

\[ \exists Q'. \forall t. \; Q(t) = Q'(Tr(t)) \]

- We focus on the query preservation for data tree transformations.
Data Tree Transformation

Data tree is a data whose nodes have a data value.

Example: Mapping noun/word/jp to word/eword/noun/jp
Node Query

Mapping data tree to tuples of data values

Example:

```
Data Tree

en-jp-dic
   |
   noun
      |
      word
         |
         (tree)
         |
         (ki)
   |
jp
   |
(trend)
   |
(keikou)
   |

binary
(x, y)

Query
for each word:
x ← value of [word]
y ← value of [jp]

Results
{(tree, ki),
(trend, keikou),
.
.
}
(Strong) Query Preservation

Let $\mathcal{L}_Q$ be a class of queries, $\mathcal{L}_T$ be a class of transformations. Given a query $Q \in \mathcal{L}_Q$ and a transformation $Tr \in \mathcal{L}_T$, we say $Tr$ preserves $Q$, if there exists $Q' \in \mathcal{L}_Q$, such that for any tree $t \in \text{dom}(Tr)$, results of $Q'$ coincide with results of $Q$.

\[
Q(t) = Q'(Tr(t))
\]
Weak Query Preservation

Let $\mathcal{L}_Q$ be a class of queries, $\mathcal{L}_T$ be a class of transformations. Given a query $Q \in \mathcal{L}_Q$ and a transformation $Tr \in \mathcal{L}_T$, we say $Tr$ weakly preserves $Q$, if there exists $Q' \in \mathcal{L}_Q$, such that for any tree $t \in \text{dom}(Tr)$, results of $Q'$ coincide with the union of the results of $Q$.

$$\bigcup_{t \in \text{dom}(Tr)} Q(t) = Q'(Tr(t))$$
Results

\{(\text{tree, ki}),
(\text{trend, keikou}),
\ldots\}\n
\textbf{Tr}_1 \text{ does not preserve } Q

\textbf{Q'} \text{ does not exist}

\textbf{for each word}
\begin{align*}
x &\leftarrow \text{data value of [word]} \\
y &\leftarrow \text{data value of [jp]}
\end{align*}

\text{en-jp-dic} \quad \text{noun}

\text{word} \quad \text{word} \quad \ldots

\begin{align*}
\text{(tree)} & \quad \text{jp} & \quad \text{(trend)} & \quad \text{jp} \\
\text{(ki)} & \quad \text{jp} & \quad \text{(keikou)}
\end{align*}

\text{en-jp-dic} \quad \text{noun}

\text{word} \quad \text{word} \quad \ldots

\begin{align*}
\text{(tree)} & \quad \text{jp} & \quad \text{(trend)} & \quad \text{jp} \\
\text{(ki)} & \quad \text{jp} & \quad \text{(keikou)}
\end{align*}
# Table of Contents

1. Our results and related work

2. Preliminaries
   - Deterministic Linear Top-down Data Tree Transducer
   - Run-based $n$-ary Query

3. Our algorithms for query preservation

4. Conclusion and future work
Our Results

We obtain the results 1. and 2. for the following class:

**Trans.** Deterministic Linear Top-down Data Tree Transducer \((\text{DLT}^V)\)

**Query** Run-based \(n\)-ary Query \((n\text{-RQ})\)

1. Query preservation problem is **decidable**
2. Query on transformed tree can be **constructed**
Our Results and Related Work

Decidability of Query Preservation

Expressiveness

Query class

Trans. class

1-RQ  n-RQ

Our results

Decidable + Constructible

DLT^V

XSLT, XQuery

Undecidable

[Bohannon+2005, Fan+2008]
Our Results and Related Work

[Our results]
- DLTT\(^V\) as transformations
- Run-based \(n\)-ary (= MSO \(n\)-ary) queries (node queries)

[Benedikt+13]
- (fu-ELT\(^R\))\(^*\) as transformations
- DMSOTT (tree queries)

[Groz+13]
- Selection of subtrees using Regular XPath filterings as transformations and queries on unranked trees
1. Our results and related work

2. Preliminaries
   - Deterministic Linear Top-down Data Tree Transducer
   - Run-based $n$-ary Query

3. Our algorithms for query preservation

4. Conclusion and future work
Data Tree

**Data Tree:** Labeled ordered ranked tree such that each node can have a data value

**Data Value:** Nonnegative integer

```
0 = 0
1 = 1
2 = tree
3 = ki
4 = trend
5 = keikou
```

```
Label                      Data Value

en-jp-dic\(^{(0)}\)        noun\(^{(1)}\)

  |                                  |

word\(^{(2)}\)             word\(^{(4)}\)

  |  |                                  |

jp\(^{(3)}\)             jp\(^{(5)}\)
```
Deterministic Linear Top-down Data Tree Transducer

Definition: \( \text{DLT}^V \ Tr = (P, \Sigma, \Delta, P_I, \delta) \)

- \( P \): Finite set of states,
- \( \Sigma \): Input alphabet,
- \( \Delta \): Output alphabet,
- \( P_I \subseteq P \): Finite set of initial states,
- \( \delta \): Finite set of transduction rules.

Transduction rules have the form:

\[
p(\sigma^{(z)}(x_1, \ldots, x_n)) \rightarrow C^{(j \leftarrow z)}[p_1(x_1), \ldots, p_n(x_n)]
\]

where \( \sigma \in \Sigma, \ p, p_1, \ldots, p_n \in P, \ C \in C(\Delta, X_n), \ t' \in T_\Delta, \)

- \( z \) is a variable for data value,
- \( j \) is a position of the context \( C \) to which the value \( z \) is transferred, and \((j \leftarrow z)\) stands for copying the data value of \( \sigma \) to a node on position \( j \) of the context \( C \).
Example of $\text{DLT}^Y$

Let $p, p_1, p_2 \in P$, $p \in P_I$, $a, m, n, \# \in \Sigma$, $b, c, \# \in \Delta$, Context $C^{(\varepsilon \leftarrow z)} = b^{(z)}(c(y_1, y_2), \#)$.

$p(a^{(z)}(x_1, x_2)) \rightarrow b^{(z)}(c(p_1(x_1), p_2(x_2)), \#) \in \delta$
Run-based $n$-ary Query

$n$-RQ: Run-based $n$-ary query [Niehren+05]

Definition: $n$-RQ $Q = (A, S)$

$A = (P, \Sigma, P_I, \delta)$ is a TA, $S$ is a set of $n$-tuple of states ($S \subseteq P^n$).

Result of accepting run by $A$

Unary

$S = \{ p_3 \}$

Result

$\{ 3, 5 \}$
Examples of Run-based Query

**Unary**

\[ S = \{ p_3, p_4 \} \]

Results

\[ \{ 3, 4, 5 \} \]

**Binary**

\[ S = \{ (p_3, p_4) \} \]

\[ \{ (3, 4), (5, 4) \} \]

\[ \{ 3,5 \} \times \{ 4 \} \]
1. Our results and related work

2. Preliminaries
   - Deterministic Linear Top-down Data Tree Transducer
   - Run-based $n$-ary Query

3. Our algorithms for query preservation

4. Conclusion and future work
Our algorithms for query preservation

Strong and Weak Query Preservation

$Tr$ preserves $Q$

$\Leftrightarrow$

$Tr$ weakly preserves $Q \land \text{"Join" does not exist}$

Join:

$\exists t_1, t_2 \in \text{dom}(Tr) \text{ s.t. } Tr(t_1) = Tr(t_2) \land \text{val}(Q(t_1)) \neq \text{val}(Q(t_2))$
Example of “Join”

Join:
\[ \exists t_1, t_2 \in \text{dom}(Tr) \text{ s.t. } Tr(t_1) = Tr(t_2) \land \text{val}(Q(t_1)) \neq \text{val}(Q(t_2)) \]

Consider the relabeling \( Tr: f, g, a \rightarrow h, h, a \)
\( Q = (A, \{p_1\}) \) where \( A = (\{p_0, p_1, p_2\}, \{f, g, a\}, \{p_0\}, \delta) \).

\[ \text{val}(Q(t_1)) = \{4\} \]

\[ \text{val}(Q(t_2)) = \{5\} \]

\[ \sigma = \{ \]
\[ p_0 \rightarrow f(p_1, p_2), \]
\[ p_0 \rightarrow g(p_2, p_1), \]
\[ p_1 \rightarrow a, \]
\[ p_2 \rightarrow a \]
\[ \} \]
Our algorithms for query preservation

Example of “Join”

Join:
\( \exists t_1, t_2 \in \text{dom}(Tr) \text{ s.t. } Tr(t_1) = Tr(t_2) \land \text{val}(Q(t_1)) \neq \text{val}(Q(t_2)) \)

Consider the relabeling \( Tr: f, g, a \rightarrow h, h, a \)

\( Q = (A, \{p_1\}) \) where \( A = (\{p_0, p_1, p_2\}, \{f, g, a\}, \{p_0\}, \delta) \).

\[ \text{val}(Q(t_1)) = \{4\} \]

\[ \text{val}(Q(t_2)) = \{5\} \]

\( \sigma = \{ \)
\( p_0 \rightarrow h(p_1, p_2), \)
\( p_0 \rightarrow h(p_2, p_1), \)
\( p_1 \rightarrow a, \)
\( p_2 \rightarrow a \}

\( Tr \text{ weakly preserves } Q \)

\( Tr \text{ doesn't preserve } Q \)

\( Tr(t_1) = Tr(t_2) \)

\[ \text{val}(Q'(Tr(t))) = \{4, 5\} \]
Our Algorithm for Query Preservation

Input: $\text{DLT}^V Tr$ and $n$-$\text{RQ} Q$
Output: If $Tr$ preserves $Q$, output “Yes,” otherwise “No.”

1. Decide whether $Tr$ weakly preserves $Q$
   - Yes — Go to 2.
   - No — Output “No”

2. Does “Join” exist?
   - Yes — Output “No”
   - No — Output “Yes” ($Tr$ preserves $Q$)
Our Algorithm for Query Preservation

Input: $DLT^V Tr$ and $n$-RQ $Q$
Output: If $Tr$ preserves $Q$, output “Yes,” otherwise “No.”

1. Decide whether $Tr$ weakly preserves $Q$
   - Yes — Go to 2.
   - No — Output “No”

2. Does “Join” exist?
   - Yes — Output “No”
   - No — Output “Yes” ($Tr$ preserves $Q$)
Weak Query Preservation: Unary Case (1-RQ)

\[ Q = (A, S) \]
\[ S = \{ \text{select, …} \} \]

**Tr** doesn’t weakly preserve \( Q \)

\[ \Updownarrow \]

There exists a tree such that a node of the tree is deleted by **Tr** and is selected by \( Q \)
Weak Query Preservation: Unary Case (1-RQ)

Input

Query 1-RQ \( Q = (A, S) \)
Trans. DLT^V \( Tr \)

1. Construct TA \( A_T \), that assigns a “delete” state to nodes that will be deleted by \( Tr \).

2. Construct TA \( A' \) such that
   \[ L(A') = L(A) \cap L(A_T). \]

   States of \( A' \) are the pairs such that
   (state of \( A \), state of \( A_T \)).

3. Check whether the set
   \( \{ (s, \text{“delete”}) \mid s \in S \} \) is empty.
Weak Query Preservation: \textit{n-ary} Case (\textit{n-RQ})

\textbf{Input:} \textit{n-RQ} \( Q = (A, S) \) and \( DLT^V \) \( Tr \)

1. Divide given query \( Q \) into \( |S| \) queries.
   
   \textbf{Example:}
   
   \[ Q = (A, \{(p_1, p_2), (p_1, p_3)\}) \rightarrow Q_1 = (A, \{(p_1, p_2)\}), \]
   
   \[ Q_2 = (A, \{(p_1, p_3)\}) \]

2. Flatten queries: \textit{n-ary} \( \rightarrow \) \textit{unary} queries.
   
   \textbf{Flattening:} \( \{(p_1, p_2, \ldots, p_n)\} \rightarrow \{p_1, p_2, \ldots, p_n\} \)

3. Input each flattened query to the algorithm for unary case.
Our Algorithm for Query Preservation

Input: \( \text{DLT}^V \ Tr \) and \( n\text{-RQ} \ Q \)

Output: If \( Tr \) preserves \( Q \), output “Yes,” otherwise “No.”

1. Decide whether \( Tr \) weakly preserves \( Q \)
   - Yes — Go to 2.
   - No — Output “No”

2. Does “Join” exist?
   - Yes — Output “No”
   - No — Output “Yes” (\( Tr \) preserves \( Q \))
How to Detect “Join”

1. Construct $\text{TA} \ A_{mk}$ from $A$ and $\text{DLT} \ T_{mk}$ from $Tr$:
   - $A_{mk}$ accepts “marked” trees.
   - $T_{mk}$ transforms “marked” trees.

2. Construct $\text{TA} \ A''_{mk}$ such that
   \[ L(A''_{mk}) = T_{mk}^{-1}(T_{mk}(L(A_{mk}))). \]

3. Construct $\text{TA} \ A_{idx}$ from $A$.
   - $A_{idx}$ accepts all variations of marked trees.
   - Thus $L(A_{idx}) \supseteq L(A_{mk})$.

4. If $L(A_{mk}) = L(A_{idx}) \cap L(A''_{mk})$, then output Yes, otherwise No.
Marked Tree

A tree into which run of query automaton is embedded
If $L(A_{mk}) = L(A_{idx}) \cap L(A''_{mk})$, then output **Yes**, otherwise **No**.
Our algorithms for query preservation

If $L(A_{mk}) = L(A_{idx}) \cap L(A''_{mk})$, then output **Yes**, otherwise **No**.
If $L(A_{mk}) = L(A_{idx}) \cap L(A_{mk}'')$, then output **Yes**, otherwise **No**.
Our algorithms for query preservation

If \( L(A_{mk}) = L(A_{idx}) \cap L(A''_{mk}) \), then output **Yes**, otherwise **No**.
Complexity

Time complexities of $\text{DLT}^V$ and $n$-$\text{RQ}$.

Weak Query Preservation \quad \text{coNP-complete}

Query Preservation \quad \text{in Double-EXPTIME}
1. Our results and related work

2. Preliminaries
   - Deterministic Linear Top-down Data Tree Transducer
   - Run-based $n$-ary Query

3. Our algorithms for query preservation

4. Conclusion and future work
Conclusion and Future Work

Decidability of Query Preservation

<table>
<thead>
<tr>
<th>Trans. class</th>
<th>Query class</th>
<th>Expressiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-RQ</td>
<td>DLT$^V$</td>
<td>Decidable + Constructible</td>
</tr>
<tr>
<td>n-RQ</td>
<td>DLT$^{VR}$</td>
<td>Future Work</td>
</tr>
<tr>
<td></td>
<td>XSLT, XQuery</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>

[Bohannon2005, Fan2008]


Appendix
Data Tree

Tree-structured data whose nodes have a data value.

Example: An XML document can be represented by a data tree

```
<en-jp-dic value="">
  <noun value="">
    <word value="tree">
      <jp value="ki" />
    </word>
  </noun>
  <word value="trend">
    <jp value="keikou" />
  </word>
  ...
</en-jp-dic>
```
Results

\{ ( \text{tree}, \text{ki} ),
(trend, \text{keikou}),
\ldots \}\n
\text{for each word}
\begin{align*}
x &\leftarrow \text{data value of [word]} \\
y &\leftarrow \text{data value of [jp]}
\end{align*}

\text{en-ja-dic}
\begin{array}{c}
\text{noun} \\
\text{word} \\
(tree) \\
(ki) \\
\text{word} \\
(trend) \\
\text{jp} \\
\text{jp} \\
\text{...}
\end{array}

\xrightarrow{T_2}\n
\text{en-ja-dic}
\begin{array}{c}
\text{noun} \\
\text{word} \\
(tree) \\
(ki) \\
\text{word} \\
(trend) \\
\text{jp} \\
\text{jp} \\
\text{...}
\end{array}

\text{eword}
\begin{array}{c}
\text{noun} \\
\text{noun} \\
(trend) \\
\text{jp} \\
\text{jp} \\
\text{keikou}
\end{array}

\text{eword}
\begin{array}{c}
\text{noun} \\
\text{noun} \\
(trend) \\
\text{jp} \\
\text{jp} \\
\text{keikou}
\end{array}

\text{Q}
\text{Q'}
\begin{align*}
\{ (\text{tree}, \text{ki} ), \\
(\text{trend}, \text{keikou}) \\
\ldots \\
\} \\
\text{Tr}_2 \text{ preserves } Q
\end{align*}

for each \textbf{word}

\begin{align*}
x & \leftarrow \text{data value of } [\text{word}] \\
y & \leftarrow \text{data value of } [\text{jp}]
\end{align*}

\begin{align*}
\text{en-jp-dic} & \quad \text{en-jp-dic} \\
| \\
\text{noun} \\
\text{noun}
\end{align*}

\begin{align*}
\text{word} & \quad \text{word} \\
\text{word} & \quad \text{word} \\
(\text{tree}) & \quad (\text{trend}) \\
(\text{ki}) & \quad (\text{keikou}) \\
\text{e-word} & \quad \text{e-word} \\
\text{e-word} & \quad \text{e-word} \\
(\text{tree}) & \quad (\text{trend}) \\
\text{noun} & \quad \text{noun} \\
(\text{ki}) & \quad (\text{keikou}) \\
\text{jp} & \quad \text{jp} \\
(\text{ki}) & \quad (\text{keikou})
\end{align*}
Weak Query Preservation: \( n \)-ary Case (\( n \)-RQ)

\[ 3\text{-RQ} (A,S) = (p_1, p_2, p_3), (p_2, p_4, p_5), (p_3, p_4, p_6) \]

\[ S_1 = \{(p_1, p_2, p_3)\} \]

\[ S_2 = \{(p_2, p_4, p_5)\} \]

\[ S_3 = \{(p_3, p_4, p_6)\} \]

\[ \text{preprocessing} \]

\[ 1\text{-RQ} (A^F, S_1^F) \]

\[ \text{Algorithm for 1-RQ} \]

\[ \text{decide} \]

\[ \text{preprocessing} \]

\[ 1\text{-RQ} (A^F, S_2^F) \]

\[ \text{Algorithm for 1-RQ} \]

\[ \text{preprocessing} \]

\[ 1\text{-RQ} (A^F, S_3^F) \]

\[ \text{Algorithm for 1-RQ} \]
Weak Query Preservation: \( n \)-ary Case (\( n \-RQ \))

3-RQ \((A,S)\), \( S = \{ (p_1,p_2,p_3), (p_2,p_4,p_5), (p_3,p_4,p_6) \} \)

3-RQ \((A,S_1)\), \( S_1 = \{ (p_1,p_2,p_3) \} \)

3-RQ \((A,S_2)\), \( S_2 = \{ (p_2,p_4,p_5) \} \)

3-RQ \((A,S_3)\), \( S_3 = \{ (p_3,p_4,p_6) \} \)

1-RQ \((A^F,S_1^F)\)

1-RQ \((A^F,S_2^F)\)

1-RQ \((A^F,S_3^F)\)

Algorithm for 1-RQ

Yes

Yes

Yes

Weakly Preserve
Weak Query Preservation: \( n \)-ary Case (\( n \)-RQ)

3-RQ (A,S), \( S = \{ (p_1, p_2, p_3), (p_2, p_4, p_5), (p_3, p_4, p_6) \} \)

3-RQ (A,S₁) \( S_1 = \{ (p_1, p_2, p_3) \} \)
3-RQ (A,S₂) \( S_2 = \{ (p_2, p_4, p_5) \} \)
3-RQ (A,S₃) \( S_3 = \{ (p_3, p_4, p_6) \} \)

preprocessing

1-RQ (A^F, S^F₁)
Algorithm for 1-RQ: No

1-RQ (A^F, S^F₂)
Algorithm for 1-RQ: Yes

1-RQ (A^F, S^F₃)
Algorithm for 1-RQ: Yes

Weak Query Preservation: X

39
Why Preprocessing Is Needed

Co-occurrence constraint may be lost by simple flattening.

Flattening \( \{ (p_1, p_2, \ldots, p_n) \} \rightarrow \{ p_1, p_2, \ldots, p_n \} \)

States are assigned by \( A \)

\[
\begin{align*}
p_1 & \quad \text{A(1)} \\
p_2 & \quad \text{B(2) #} \\
p_# & \quad # \quad p_# \\
p_# & \quad # \quad #
\end{align*}
\]

Results

3-RQ \((A,S)\)
\(S = \{ (p_1,p_2,p_3) \}\)

empty set \(\Phi\)
\(= \{1\} \times \{2\} \times \Phi\)

1-RQ \((A,S')\)
\(S' = \{ p_1, p_2, p_3 \}\)

\(\{ 1, 2 \}\)
\(= \{1\} \cup \{2\} \cup \Phi\)
Why Preprocessing Is Needed

Co-occurrence constraint may be lost by simple flattening.

Flattening \( \{ (p_1, p_2, \ldots, p_n) \} \rightarrow \{ p_1, p_2, \ldots, p_n \} \)

States are assigned by A

\[
\begin{align*}
p_1 & \quad A(1) \\
p_2 & \quad B(2) \\
p_# & \quad p_# \quad B(2) \\
p_# & \quad # \\
p_# & \quad #
\end{align*}
\]

3-RQ \((A,S)\)
\[
S = \{ (p_1,p_2,p_3) \}
\]

empty set \(\Phi\)
\[
= \{1\} \times \{2\} \times \Phi
\]

1-RQ \((A,S')\)
\[
S' = \{ p_1, p_2, p_3 \}
\]

\[
\{1,2\}
\]
\[
= \{1\} \cup \{2\} \cup \Phi
\]
Preprocessing

Construct $A^F$ and $S^F$ from flattened 1-RQ $(A, S')$. $A^F$ accepts tree $t$ if all states in $S$ are assigned to nodes of $t$ by $A$.

$(A, S)$ is preserved $\iff (A^F, S^F)$ is preserved

3-RQ $(A, S)$,
$S = \{ (p_1, p_2, p_3) \}$

$\downarrow$ flattening

1-RQ $(A, S')$,
$S' = \{ p_1, p_2, p_3 \}$

$\downarrow$ construct

1-RQ $(A^F, S^F)$
Tree automaton $A_{\text{idx}}$

$A_{\text{idx}}$ accepts all variations of marked trees.

Example: There are $3^3 = 27$ variations of marked trees.