Typechecking Top-Down XML Transformations

Wim Martens, Frank Neven

University of Limburg
Overview

- Introduction
- Schema : Tree Languages
- Tree Transformations : XSLT
- The Typechecking Problem
- Main Results
- Proof Ideas
- Conclusion and Future Work
Importance of Typechecking

An example:

Suppose that a certain user community agrees to produce documents satisfying a common tree type $\tau$.

For a user, who executes an XML to XML transformation $T$, an input tree type $\tau_{in}$ is available.
Importance of Typechecking

An example:

Suppose that a certain user community agrees to produce documents satisfying a common tree type $\tau$.

For a user, who executes an XML to XML transformation $T$, an input tree type $\tau_{in}$ is available.

This is called the *Typechecking Problem*. 
<food>
  <dish type="Belgian">
    <potato> fries </potato>
    <sauce> mayonnaise </sauce>
  </dish>
  <dish type="Italian">
    <pasta> spaghetti </pasta>
    <vegetable> tomato </vegetable>
  </dish>
</food>
<food>
  <dish type="Belgian">
    <potato> fries </potato>
    <sauce> mayonnaise </sauce>
  </dish>
  <dish type="Italian">
    <pasta> spaghetti </pasta>
    <vegetable> tomato </vegetable>
  </dish>
</food>
<food>
  <dish type="Belgian">
    <potato> fries </potato>
    <sauce> mayonnaise </sauce>
  </dish>
  <dish type="Italian">
    <pasta> spaghetti </pasta>
    <vegetable> tomato </vegetable>
  </dish>
</food>
Previous Work

  - Typechecking quickly turns undecidable when data or attribute values are incorporated.
Previous Work

- Alon et al (2001): Typechecking quickly turns *undecidable* when *data* or *attribute values* are incorporated.

- Milo, Suciu, Vianu (2000): When only looking at *structural properties* of trees, typechecking is *decidable* for a *large fragment* of tree transformations (formalized by $k$-pebble transducers).
  
  Complexity is high (non-elementary).
Previous Work

Alon et al (2001) :
- Typechecking quickly turns **undecidable** when data or attribute values are incorporated.

Milo, Suciu, Vianu (2000) :
- When only looking at **structural properties** of trees, typechecking is **decidable** for a **large fragment** of tree transformations (formalized by $k$-pebble transducers).
- Complexity is high (non-elementary).

We try to lower complexity by simplifying the schema languages and the tree transformations.
Overview

- Introduction
- **Schema : Tree Languages**
- Tree Transformations : XSLT
- The Typechecking Problem
- Main Results
- Proof Ideas
- Conclusion and Future Work
Tree languages

DTDs:

food \rightarrow (dish)^*  
dish \rightarrow (potato | pasta) (vegetable)^* (sauce)^?
Tree languages

**DTDs:**

\[
\begin{align*}
\text{food} & \rightarrow (\text{dish})^* \\
\text{dish} & \rightarrow (\text{potato} \mid \text{pasta}) (\text{vegetable})^* (\text{sauce})? 
\end{align*}
\]
Tree languages

DTDs:

food → (dish)*
dish → (potato | pasta) (vegetable)* (sauce)?
Tree languages

**DTDs:**

- \( \text{food} \rightarrow (\text{dish})^* \)
- \( \text{dish} \rightarrow (\text{potato} \mid \text{pasta}) \ (\text{vegetable})^* \ (\text{sauce})? \)

**Tree Automata:**

*unranked* tree automata
Tree languages

**DTDs:**

\[
\text{food} \rightarrow (\text{dish})* \\
\text{dish} \rightarrow (\text{potato} \mid \text{pasta}) (\text{vegetable})* (\text{sauce})? \]

**Tree Automata:**

*unranked* tree automata

Specify transition function \( \delta \) by *regular string languages* over *states.*
Tree Automata - Example

Evaluate Boolean expressions:

\[
\begin{array}{c}
\wedge \\
\vee \\
\wedge \\
0 \\
1 \\
0 \\
1 \\
1 \\
1 \\
0 \\
1 \\
1 \\
\end{array}
\]

States: \{t, f\}

<table>
<thead>
<tr>
<th>label</th>
<th>state</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>\wedge</td>
<td>t</td>
<td>tt*</td>
</tr>
<tr>
<td>\wedge</td>
<td>f</td>
<td>\varepsilon</td>
</tr>
<tr>
<td>\vee</td>
<td>t</td>
<td>(f</td>
</tr>
<tr>
<td>\vee</td>
<td>f</td>
<td>(f</td>
</tr>
<tr>
<td>\vee</td>
<td>t</td>
<td>ff*</td>
</tr>
</tbody>
</table>
Tree Automata - Example

Evaluate Boolean expressions:

States: \{t, f\}

<table>
<thead>
<tr>
<th>label</th>
<th>state</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>\delta ( 1 , t ) = \varepsilon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\delta ( 0 , f ) = \varepsilon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\delta ( \wedge , t ) = tt^*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\delta ( \wedge , f ) = (f</td>
<td>t)^* f (f</td>
<td>t)^*</td>
</tr>
<tr>
<td>\delta ( \vee , t ) = (f</td>
<td>t)^* t (f</td>
<td>t)^*</td>
</tr>
<tr>
<td>\delta ( \vee , f ) = ff^*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tree Automata - Example

Evaluate Boolean expressions:

States: \{t, f\}

<table>
<thead>
<tr>
<th>label</th>
<th>state</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>\δ(1, t) = ε</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\δ(0, f) = ε</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\δ(\∧, t) = tt*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\δ(\∧, f) = (f</td>
<td>t)* f (f</td>
<td>t)*</td>
</tr>
<tr>
<td>\δ(\∨, t) = (f</td>
<td>t)* t (f</td>
<td>t)*</td>
</tr>
<tr>
<td>\δ(\∨, f) = ff*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tree Automata - Example

Evaluate Boolean expressions:

States: \{t, f\}

<table>
<thead>
<tr>
<th>label</th>
<th>state</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\land)</td>
<td>t</td>
<td>(\varepsilon)</td>
</tr>
<tr>
<td>(\lor)</td>
<td>f</td>
<td>(f</td>
</tr>
<tr>
<td>(\land)</td>
<td>f</td>
<td>(tt^*)</td>
</tr>
<tr>
<td>(\land)</td>
<td>t</td>
<td>((f</td>
</tr>
<tr>
<td>(\lor)</td>
<td>t</td>
<td>((f</td>
</tr>
<tr>
<td>(\lor)</td>
<td>f</td>
<td>(ff^*)</td>
</tr>
</tbody>
</table>
Evaluate Boolean expressions:

States: \{t, f\}

<table>
<thead>
<tr>
<th>label</th>
<th>state</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\wedge)</td>
<td>t</td>
<td>(tt^*)</td>
</tr>
<tr>
<td>(\wedge)</td>
<td>f</td>
<td>((f</td>
</tr>
<tr>
<td>(\vee)</td>
<td>t</td>
<td>((f</td>
</tr>
<tr>
<td>(\vee)</td>
<td>f</td>
<td>(ff^*)</td>
</tr>
</tbody>
</table>

\(\delta(1, t) = \varepsilon\)
\(\delta(0, f) = \varepsilon\)
Tree Automata - Example

Evaluate Boolean expressions:

States: \{t, f\}

<table>
<thead>
<tr>
<th>label</th>
<th>state</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\wedge)</td>
<td>(t)</td>
<td>(\varepsilon)</td>
</tr>
<tr>
<td>(\wedge)</td>
<td>(f)</td>
<td>((f</td>
</tr>
<tr>
<td>(\vee)</td>
<td>(t)</td>
<td>((f</td>
</tr>
<tr>
<td>(\vee)</td>
<td>(f)</td>
<td>(ff^*)</td>
</tr>
</tbody>
</table>
Tree Automata - Example

Evaluate Boolean expressions:

States: \{t, f\}

<table>
<thead>
<tr>
<th>label</th>
<th>state</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\land)</td>
<td>1, t</td>
<td>(\varepsilon)</td>
</tr>
<tr>
<td>(\land)</td>
<td>0, f</td>
<td>(\varepsilon)</td>
</tr>
<tr>
<td>(\land)</td>
<td>(\land), t</td>
<td>(tt^*)</td>
</tr>
<tr>
<td>(\land)</td>
<td>(\land), f</td>
<td>((f</td>
</tr>
<tr>
<td>(\lor)</td>
<td>1, t</td>
<td>((f</td>
</tr>
<tr>
<td>(\lor)</td>
<td>1, f</td>
<td>(ff^*)</td>
</tr>
</tbody>
</table>
Overview

- Introduction
- Schema: Tree Languages
- Tree Transformations: XSLT
- The Typechecking Problem
- Main Results
- Proof Ideas
- Conclusion and Future Work
XSLT - simple case

Example: 1 mode: simple

\[(a, \text{simple}) \rightarrow b \]
\[
  \quad \downarrow \text{simple}
\]

\[(b, \text{simple}) \rightarrow a \]
\[
  \quad \downarrow \text{simple}
\]
XSLT - simple case

Example: 1 mode: simple

\[(a, \text{simple}) \rightarrow b \rightarrow \text{simple} \]

\[(b, \text{simple}) \rightarrow a \rightarrow \text{simple} \]

Diagram:
- simple
  - a
    - b
    - a
  - b
    - a
XSLT - simple case

Example: 1 mode: simple

\[(a, \text{simple}) \rightarrow b \text{ simple} \]

\[(b, \text{simple}) \rightarrow a \text{ simple} \]
XSLT - simple case

Example: 1 mode: simple

Example diagram showing the transformation rules for XSLT in the simple case.
XSLT - simple case

Example: 1 mode: simple

(a, simple) → \( b \)
\( \quad \text{simple} \)

(b, simple) → \( a \)
\( \quad \text{simple} \)

simple \( a \) simple \( b \) simple \( a \) simple \( b \) simple
\| \| \| \| \| \|
\( a \) \( \quad b \) \( a \) \( \quad b \) \( a \) \( \quad b \) \( a \) \( \quad b \)
\| \| \| \| \| \| \| \| \|
\( a \) \( \quad b \) \( a \) \( \quad b \) \( a \) \( \quad b \) \( a \) \( \quad b \)
\| \| \| \| \| \| \| \| \|
\( a \) \( \quad b \) \( a \) \( \quad b \) \( a \) \( \quad b \) \( a \) \( \quad b \)
\| \| \| \| \| \| \| \| \|
\( a \) \( \quad b \) \( a \) \( \quad b \) \( a \) \( \quad b \) \( a \) \( \quad b \)
XSLT - copying

Example: 3 modes: simple, copy, id

$((a, \text{copy}) \rightarrow c \rightarrow \text{simple} \rightarrow \text{id})$

$((a, \text{simple}) \rightarrow b \rightarrow \text{copy} \rightarrow \text{id})$

$((a, \text{id}) \rightarrow a \rightarrow \text{id})$
XSLT - copying

Example: 3 modes: simple, copy, id

(a, copy) →

\[ \begin{array}{c}
\text{c} \\
\text{simple} & \text{id}
\end{array} \]

(a, simple) →

\[ \begin{array}{c}
\text{b} \\
\text{copy} & \text{id}
\end{array} \]

(a, id) →

\[ \begin{array}{c}
\text{a} \\
\text{a} \\
\text{a} \\
\text{a} \\
\text{a}
\end{array} \]
XSLT - copying

Example: 3 modes: simple, copy, id

(a, copy) →

\[ \begin{align*}
\text{c} & \quad \text{simple} \quad \text{id} \\
\end{align*} \]

(a, simple) → b

\[ \begin{align*}
\text{copy} & \quad \text{id} \\
\end{align*} \]

(a, id) → a

\[ \begin{align*}
\text{copy} & \quad \text{a} \\
\text{a} & \quad \text{simple} \quad \text{a} \quad \text{id} \\
\text{a} & \quad \text{a} \quad \text{a} \\
\text{a} & \quad \text{a} \quad \text{a} \\
\text{a} & \quad \text{a} \quad \text{a} \\
\end{align*} \]
Example: 3 modes: simple, copy, id

(a, copy) →

         c
        / \
      simple id

(a, simple) →

     b
    / \
   copy id

(a, id) →

       a
      / \
     copy id

Typechecking Top-Down XML Transformations – p.11/24
XSLT - copying

Example: 3 modes: simple, copy, id

(a, copy) →

(c) └──
    └──
        simple    id

(a, simple) →

(b) └──
    └──
        copy    id

(a, id) →

(a) └──
    └──
        copy    id

...
Example: 2 modes: delete, copy

\[(a, \text{copy}) \rightarrow c\]

\[c \rightarrow \text{delete} \quad \text{delete}\]

\[(a, \text{delete}) \rightarrow \text{copy}\]
XSLT - deleting

Example: 2 modes: delete, copy

\((a, \text{copy}) \rightarrow \)

\(c \quad \text{delete} \quad \text{delete} \)

\((a, \text{delete}) \rightarrow \text{copy} \)

\(\text{copy} \quad a \quad \)

\(\quad a \quad \)

\(\quad a \quad \)

\(\quad a \quad \)

\(\quad a \quad \)
Example: 2 modes: delete, copy

\[(a, \text{copy}) \rightarrow \]
\[
\begin{array}{c}
c \\
\text{delete} & \text{delete}
\end{array}
\]

\[(a, \text{delete}) \rightarrow \text{copy}\]
XSLT - deleting

Example: 2 modes: delete, copy

(a, copy) →

\[ \text{copy } \underbrace{\text{a}}_a \]
\[ \underbrace{\text{delete } \underbrace{\text{a}}_a \underbrace{\text{delete } \underbrace{\text{a}}_a \text{delete}}_c}_c \]

(a, delete ) → copy

\[ \text{copy } \underbrace{\text{a}}_a \]
\[ \underbrace{\text{delete } \underbrace{\text{a}}_a \underbrace{\text{delete } \underbrace{\text{a}}_a \text{delete}}_c}_c \]
\[ \underbrace{\text{copy } \underbrace{\text{a}}_a \underbrace{\text{copy}}_c}_c \]
Example: 2 modes: delete, copy

\((a, \text{copy}) \rightarrow c\)

\(\begin{array}{c}
\text{delete} \\
\text{delete}
\end{array}\)

\((a, \text{delete}) \rightarrow \text{copy}\)

\(\begin{array}{c}
\text{copy} \\
\text{copy}
\end{array}\)

\(\begin{array}{c}
\text{delete} \\
\text{delete}
\end{array}\)

\(\begin{array}{c}
\text{copy} \\
\text{copy}
\end{array}\)

\(\ldots\)

\(\begin{array}{c}
\text{c} \\
\text{c}
\end{array}\)
Overview

- Introduction
- Schema : Tree Languages
- Tree Transformations : XSLT
- The Typechecking Problem
- Main Results
- Proof Ideas
- Conclusion and Future Work
The Typechecking Problem

Given:

- input tree language $\tau_{in}$
- output tree language $\tau_{out}$
- XML-transformation $T$

DTD, TA  
DTD, TA  
XSLT
The Typechecking Problem

Given:

- input tree language $\tau_{in}$
- output tree language $\tau_{out}$
- XML-transformation $T$

Is it true that,

$$\forall t \in \tau_{in} \Rightarrow T(t) \in \tau_{out}?$$
Overview

- Introduction
- Schema : Tree Languages
- Tree Transformations : XSLT
- The Typechecking Problem
- Main Results
- Proof Ideas
- Conclusion and Future Work
Main Results

What is the complexity of the typechecking problem?
## Main Results

What is the complexity of the typechecking problem?

<table>
<thead>
<tr>
<th></th>
<th>Tree Automata (NFA)</th>
<th>DTD (RE)</th>
<th>DTD (DFA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleting + copying</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>no deleting + copying</td>
<td>EXPTIME</td>
<td>PSPACE</td>
<td>PSPACE</td>
</tr>
<tr>
<td>no deleting + bounded copying</td>
<td>EXPTIME</td>
<td>PSPACE</td>
<td>PTIME</td>
</tr>
</tbody>
</table>
Main Results

What is the complexity of the typechecking problem?

<table>
<thead>
<tr>
<th></th>
<th>Tree Automata (NFA)</th>
<th>DTD (RE)</th>
<th>DTD (DFA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleting + copying</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>no deleting + copying</td>
<td>EXPTIME</td>
<td>PSPACE</td>
<td>PSPACE</td>
</tr>
<tr>
<td>no deleting + bounded copying</td>
<td>EXPTIME</td>
<td>PSPACE</td>
<td>PTIME</td>
</tr>
</tbody>
</table>

Typechecking is complete for the complexity classes shown here.
Tree Automata: Toolbox

We can use different formalisms to represent the regular languages in tree automata.

Then, we can use the tree automata as a toolbox:

1. Emptiness of $\text{TA}(\text{2AFA})$ is in $\text{PSPACE}$.
2. Emptiness of $\text{TA}(\text{NFA})$ is in $\text{PTIME}$.
Overview

- Introduction
- Schema : Tree Languages
- Tree Transformations : XSLT
- The Typechecking Problem
- Main Results
- Proof Ideas
- Conclusion and Future Work
Main Results

What is the complexity of the typechecking problem?

<table>
<thead>
<tr>
<th></th>
<th>Tree Automata (NFA)</th>
<th>DTD (RE)</th>
<th>DTD (DFA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleting + copying</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>no deleting + copying</td>
<td>EXPTIME</td>
<td>PSPACE</td>
<td>PSPACE</td>
</tr>
<tr>
<td>no deleting + bounded copying</td>
<td>EXPTIME</td>
<td>PSPACE</td>
<td>PTIME</td>
</tr>
</tbody>
</table>
Copying, DTD(RE) → in PSPACE

Reduction to emptiness of TA(2AFA).
Reduction to emptiness of TA(2AFA).

Construct a TA(2AFA) $B$ such that

$$L(B) = \{ t \in \tau_{\text{in}} \mid T(t) \not\in \tau_{\text{out}} \}.$$
Reduction to emptiness of TA(2AFA).

Construct a TA(2AFA) $B$ such that
\[
L(B) = \{ t \in \tau_{in} \mid T(t) \not\in \tau_{out} \}.
\]

$B$ checks that $t \in \tau_{in}$
Copying, DTD(RE) $\rightarrow$ in PSPACE

Reduction to emptiness of TA(2AFA).

Construct a TA(2AFA) $B$ such that

$$L(B) = \{ t \in \tau_{in} \mid T(t) \not\in \tau_{out} \}.$$ 

$B$ checks that $t \in \tau_{in}$

![Diagram showing the process with labels $a$, $b$, and $T$ connections indicating type checking and input-output relations.]

$\tau_{out}: b \rightarrow e$

$\not\in L(e)$
Main Results

What is the complexity of the typechecking problem?

<table>
<thead>
<tr>
<th></th>
<th>Tree Automata (NFA)</th>
<th>DTD (RE)</th>
<th>DTD (DFA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleting + copying</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>no deleting + copying</td>
<td>EXPTIME</td>
<td>PSPACE</td>
<td>PSPACE</td>
</tr>
<tr>
<td>no deleting + bounded copying</td>
<td>EXPTIME</td>
<td>PSPACE</td>
<td>PTIME</td>
</tr>
</tbody>
</table>
Bound. Copying, DTD(DFA) → in PTIME

Look at previous reduction to emptiness TA(2AFA).
Bound. Copying, DTD(DFA) $\rightarrow$ in PTIME

Look at previous reduction to emptiness TA(2AFA).

Now:

- transformations can make only a bounded (fixed) number of copies
- DFAs are used
Bound. Copying, DTD(DFA) $\rightarrow$ in PTIME

Look at previous reduction to emptiness TA(2AFA).

Now:

- transformations can make only a bounded (fixed) number of copies
- DFAs are used

So...

- 2-way no longer needed
- alternation no longer needed

And emptiness of TA(NFA) $\in$ PTIME.
Overview

- Introduction
- Schema : Tree Languages
- Tree Transformations : XSLT
- The Typechecking Problem
- Main Results
- Proof Ideas
- Conclusion and Future Work
Conclusion and Future Work

If we eliminate

- unbounded copying and deleting in the tree transformation
- non-determinism in the schema languages

the typechecking problem becomes \texttt{PTIME}-complete.

In the future, we will try to expand the \texttt{PTIME}-result.