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 τ.

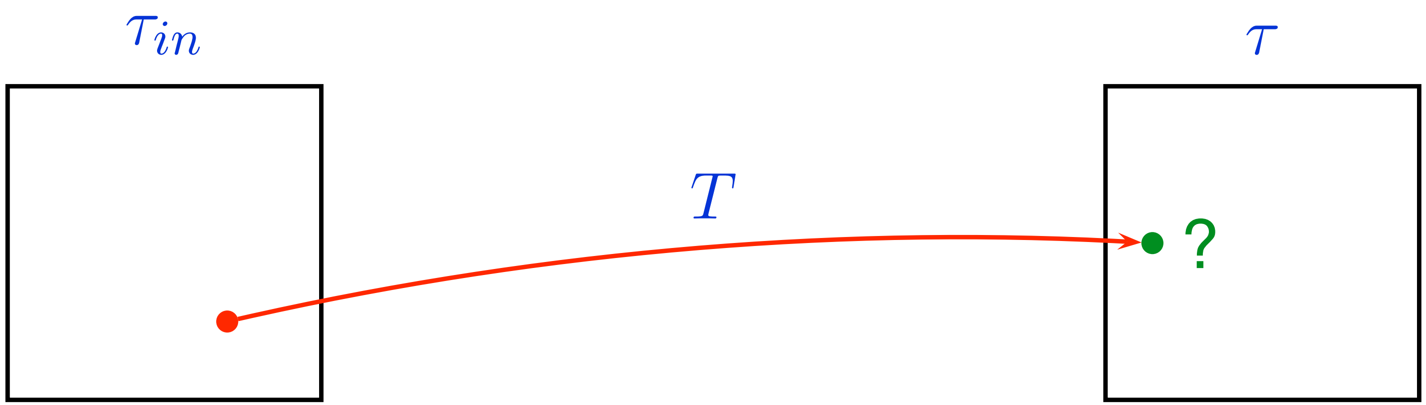
For a user, who executes an XML to XML transformation T, an input tree type τ_{in} is available.

Importance of Typechecking

An example:

Suppose that a certain user community agrees to produce documents satisfying a common tree type τ.

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



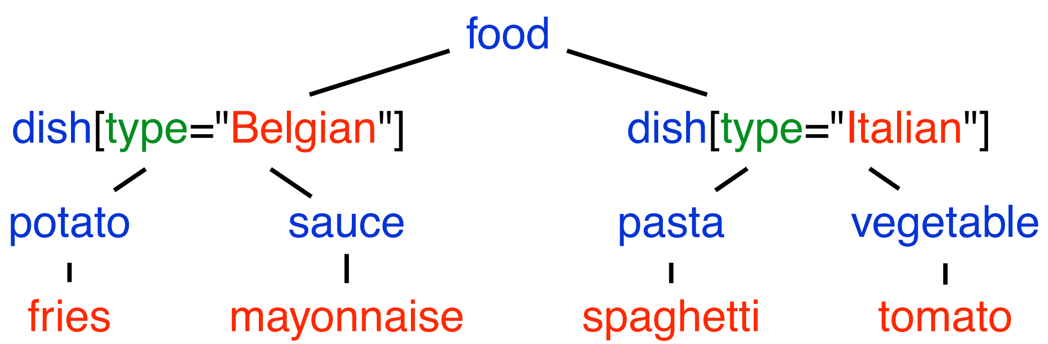
This is called the *Typechecking Problem*.

XML

```
<food>
    <dish type="Belgian">
        <potato> fries </potato>
        <sauce> mayonnaise </sauce>
    </dish>
    <dish type="Italian">
        <pasta> spaghetti </pasta>
        <vegetable> tomato </vegetable>
    </dish>
</food>
```

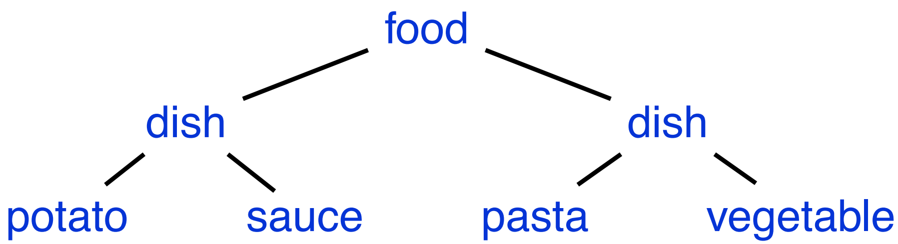
XML

```
<food>
    <dish type="Belgian">
        <potato> fries </potato>
        <sauce> mayonnaise </sauce>
    </dish>
    <dish type="Italian">
        <pasta> spaghetti </pasta>
        <vegetable> tomato </vegetable>
    </dish>
</food>
```



XML

```xml
<food>
    <dish type="Belgian">
        <potato> fries </potato>
        <sauce> mayonnaise </sauce>
    </dish>
    <dish type="Italian">
        <pasta> spaghetti </pasta>
        <vegetable> tomato </vegetable>
    </dish>
</food>
```



Previous Work

- Alon et al (2001) :
  - 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 → (dish)*

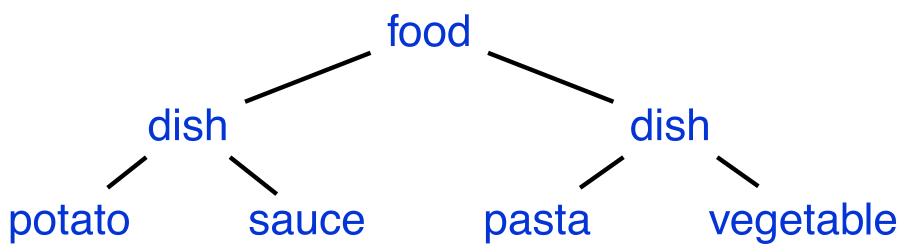
  dish → (potato | pasta) (vegetable)* (sauce)?

Tree languages

- DTDs:

  food → (dish)*

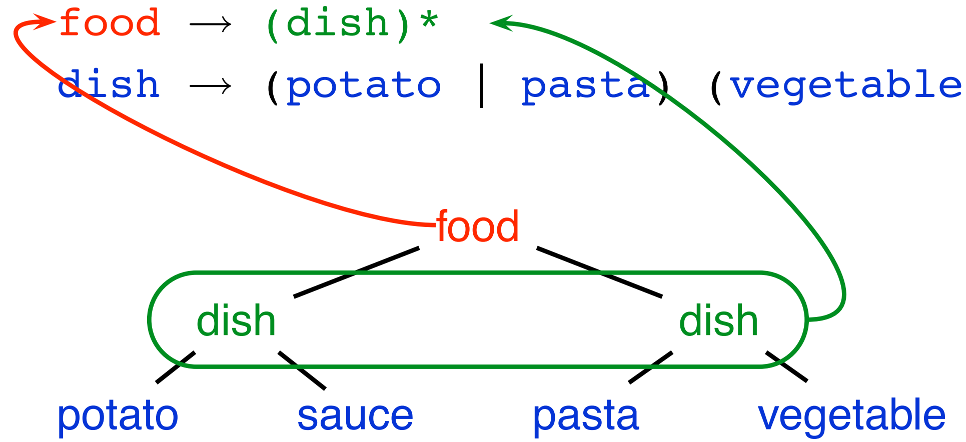
  dish → (potato | pasta) (vegetable)* (sauce)?



Tree languages

- DTDs:

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

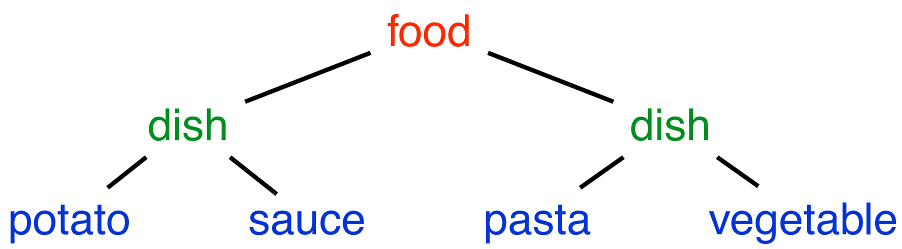


Tree languages

- DTDs:

  $\texttt{food} \longrightarrow (\texttt{dish})^*$

  $\texttt{dish} \longrightarrow (\texttt{potato} \mid \texttt{pasta})\ (\texttt{vegetable})^*\ (\texttt{sauce})?$



- Tree Automata:

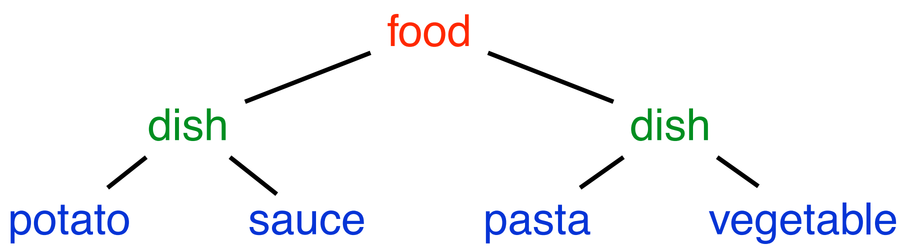
  unranked tree automata

Tree languages

- DTDs:

  food → (dish)*

  dish → (potato | pasta) (vegetable)* (sauce)?



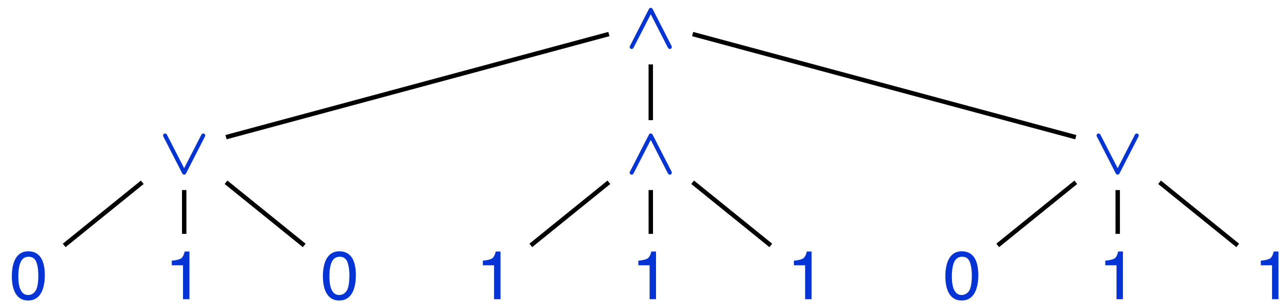
- Tree Automata:

  unranked tree automata

  Specify transition function δ by regular string languages over states.

Tree Automata - Example

Evaluate Boolean expressions:

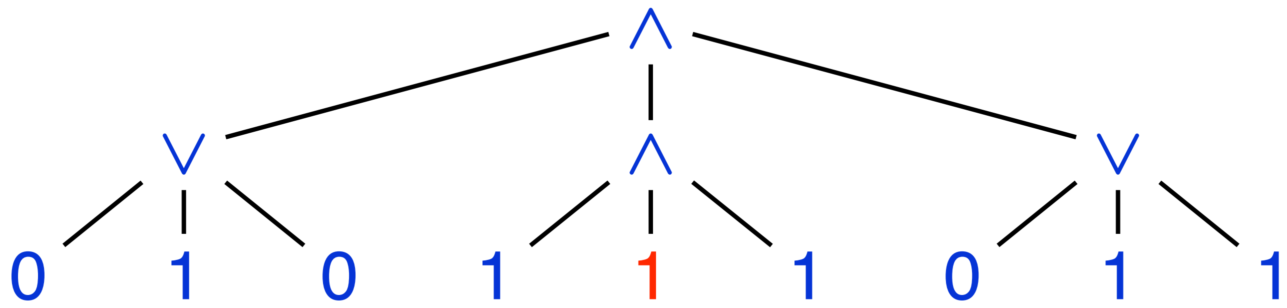


States: $\{t, f\}$

	label		state		language
$\delta($	1	,	t	$) =$	ε
$\delta($	0	,	f	$) =$	ε
$\delta($	$\wedge$	,	t	$) =$	tt^*
$\delta($	$\wedge$	,	f	$) =$	$(f\|t)^*f(f\|t)^*$
$\delta($	$\vee$	,	t	$) =$	$(f\|t)^*t(f\|t)^*$
$\delta($	$\vee$	,	f	$) =$	ff^*

Tree Automata - Example

Evaluate Boolean expressions:

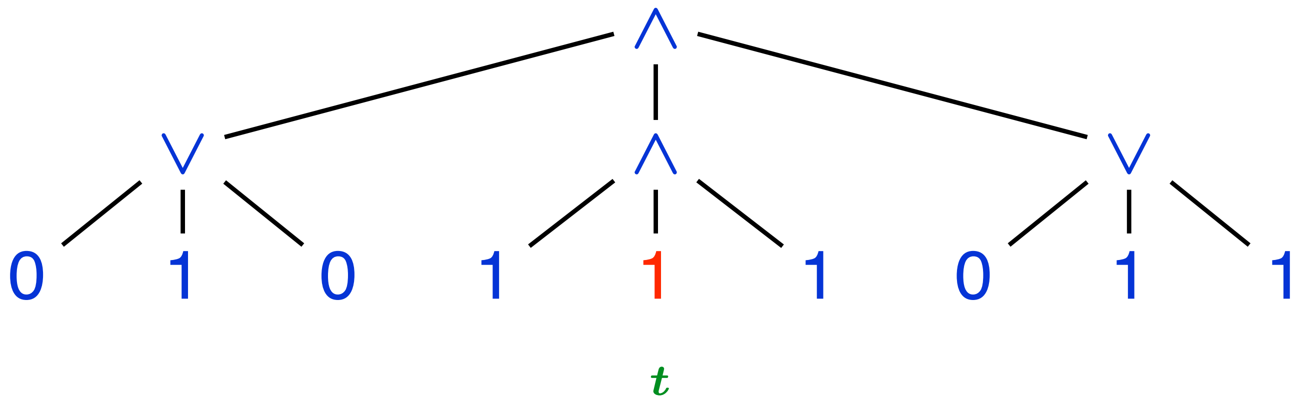


States: $\{t, f\}$

	label	state	language
$\delta($	1	$, \; t \;) =$	ε
$\delta($	0	$, \; f \;) =$	ε
$\delta($	$\wedge$	$, \; t \;) =$	tt^*
$\delta($	$\wedge$	$, \; f \;) =$	$(f\|t)^* f (f\|t)^*$
$\delta($	$\vee$	$, \; t \;) =$	$(f\|t)^* t (f\|t)^*$
$\delta($	$\vee$	$, \; f \;) =$	ff^*

Tree Automata - Example

Evaluate Boolean expressions:

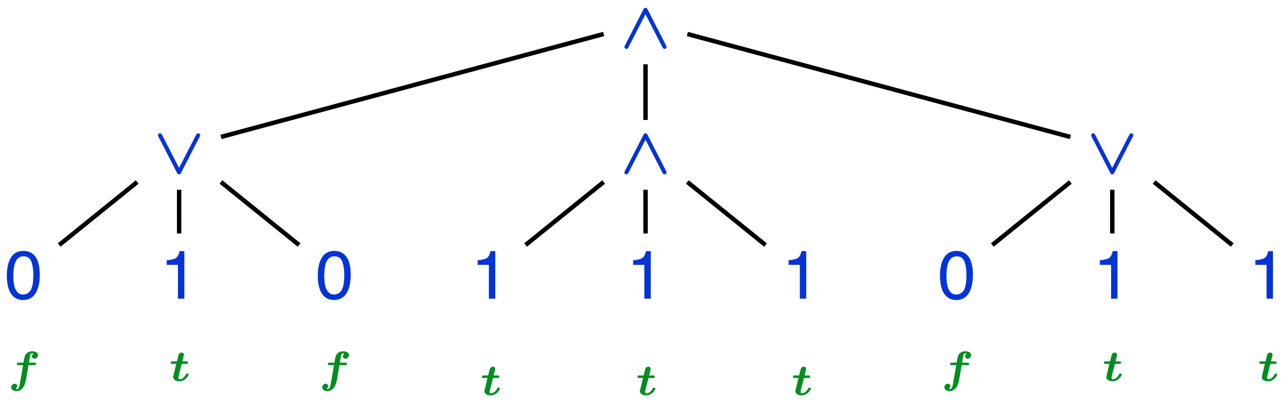


States: $\{t, f\}$

	label	state		language
$\delta($	1	$, \; t$	$) =$	ε
$\delta($	0	$, \; f$	$) =$	ε
$\delta($	$\wedge$	$, \; t$	$) =$	tt^*
$\delta($	$\wedge$	$, \; f$	$) =$	$(f\|t)^* f (f\|t)^*$
$\delta($	$\vee$	$, \; t$	$) =$	$(f\|t)^* t (f\|t)^*$
$\delta($	$\vee$	$, \; f$	$) =$	ff^*

Tree Automata - Example

Evaluate Boolean expressions:

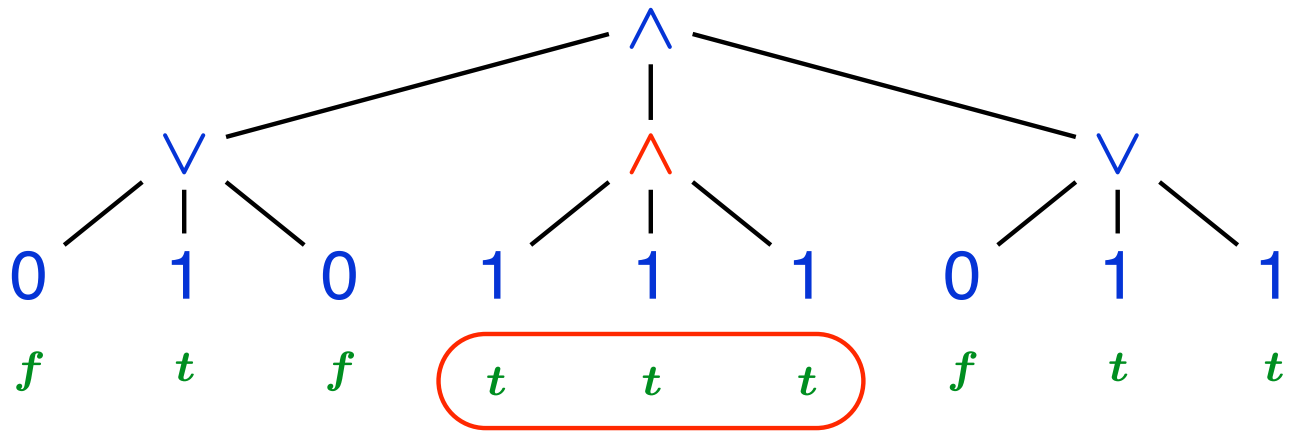


States: $\{t, f\}$

	label	state	language
$\delta($	1	$, \; t \;) =$	ε
$\delta($	0	$, \; f \;) =$	ε
$\delta($	$\wedge$	$, \; t \;) =$	tt^*
$\delta($	$\wedge$	$, \; f \;) =$	$(f\|t)^*f(f\|t)^*$
$\delta($	$\vee$	$, \; t \;) =$	$(f\|t)^*t(f\|t)^*$
$\delta($	$\vee$	$, \; f \;) =$	ff^*

Tree Automata - Example

Evaluate Boolean expressions:

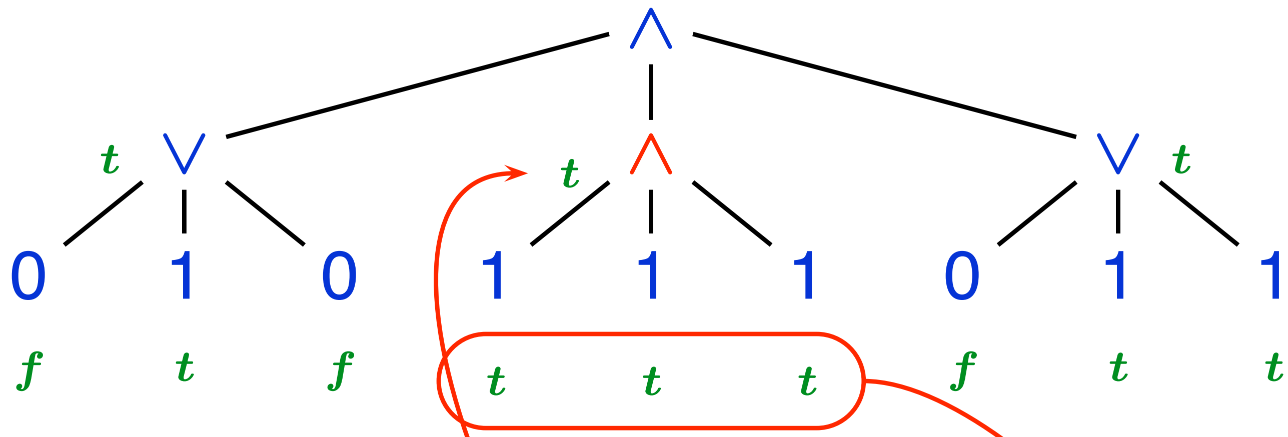


States: $\{t, f\}$

	label	state		language
$\delta($	1	$, t$	$) =$	ε
$\delta($	0	$, f$	$) =$	ε
$\delta($	$\wedge$	$, t$	$) =$	tt^*
$\delta($	$\wedge$	$, f$	$) =$	$(f\|t)^* f (f\|t)^*$
$\delta($	$\vee$	$, t$	$) =$	$(f\|t)^* t (f\|t)^*$
$\delta($	$\vee$	$, f$	$) =$	$f f^*$

Tree Automata - Example

Evaluate Boolean expressions:

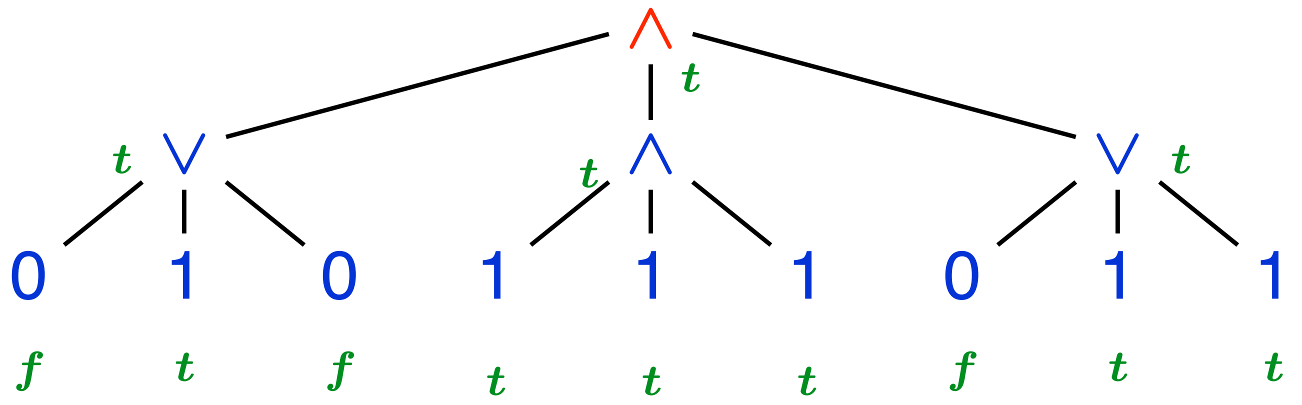


States: $\{t, f\}$

	label	state		language
$\delta($	1	$, t$	$) =$	ε
$\delta($	0	$, f$	$) =$	ε
$\delta($	$\wedge$	$, t$	$) =$	tt^*
$\delta($	$\wedge$	$, f$	$) =$	$(f\|t)^* f (f\|t)^*$
$\delta($	$\vee$	$, t$	$) =$	$(f\|t)^* t (f\|t)^*$
$\delta($	$\vee$	$, f$	$) =$	$f f^*$

Tree Automata - Example

Evaluate Boolean expressions:



States: $\{t, f\}$

	label	state	language
$\delta($	1	, t	$) = \varepsilon$
$\delta($	0	, f	$) = \varepsilon$
$\delta($	$\wedge$	, t	$) = tt^*$
$\delta($	$\wedge$	, f	$) = (f\|t)^* f (f\|t)^*$
$\delta($	$\vee$	, t	$) = (f\|t)^* t (f\|t)^*$
$\delta($	$\vee$	, f	$) = ff^*$

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

$(a, \text{simple}) \longrightarrow \begin{array}{c} b \\ | \\ \text{simple} \end{array}$

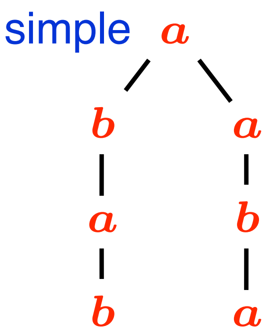
$(b, \text{simple}) \longrightarrow \begin{array}{c} a \\ | \\ \text{simple} \end{array}$

XSLT - simple case

Example : 1 mode : simple

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

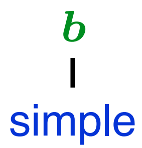
$(b,\text{simple}) \rightarrow$ a
|
simple



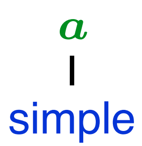
XSLT - simple case

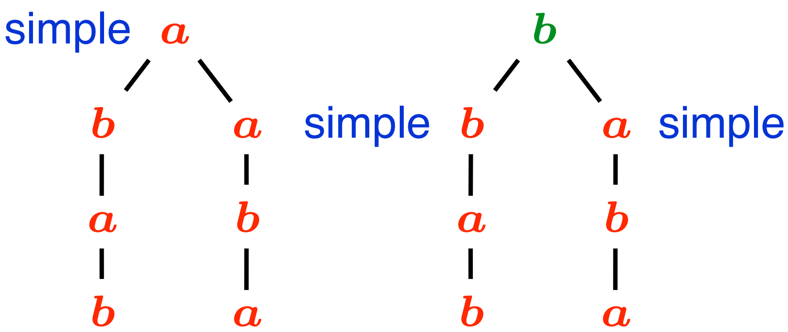
Example : 1 mode : simple

$(a,$simple$) \rightarrow$



$(b,$simple$) \rightarrow$

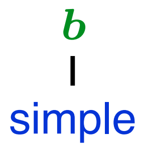




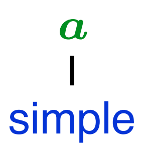
XSLT - simple case

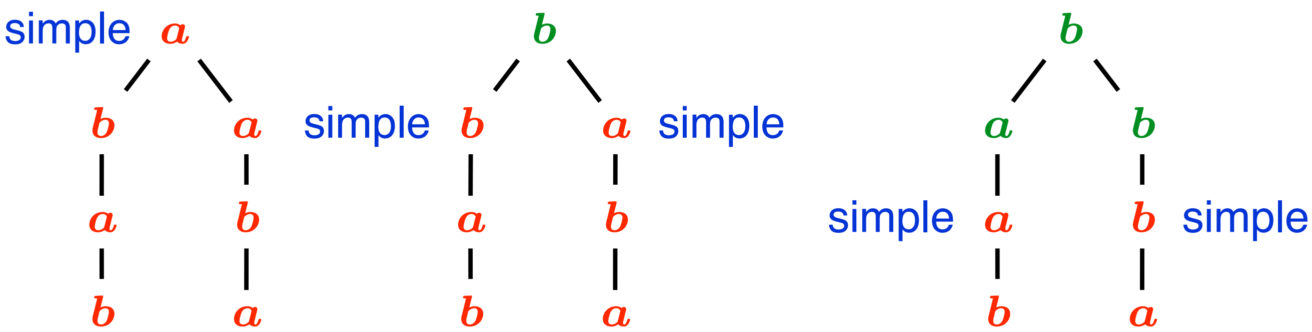
Example : 1 mode : simple

$(a,\text{simple}) \rightarrow$



$(b,\text{simple}) \rightarrow$

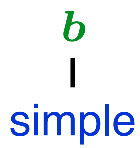




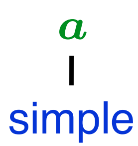
XSLT - simple case

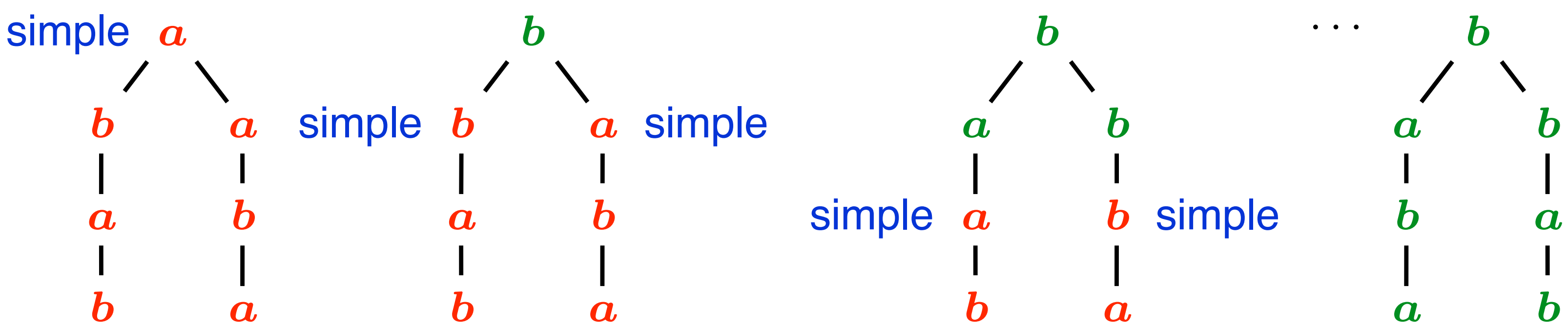
Example : 1 mode : simple

(*a*,simple) →



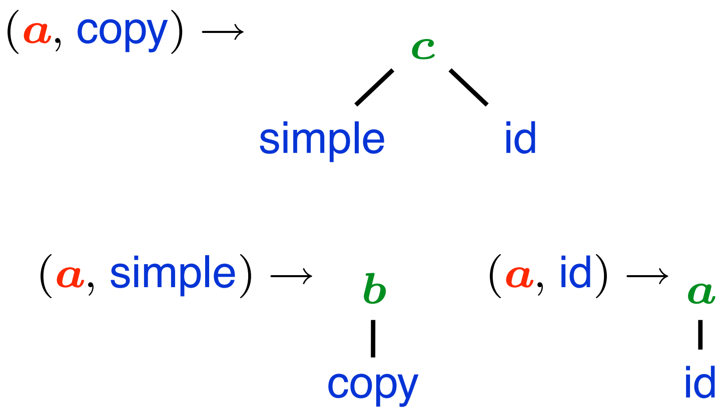
(*b*,simple) →





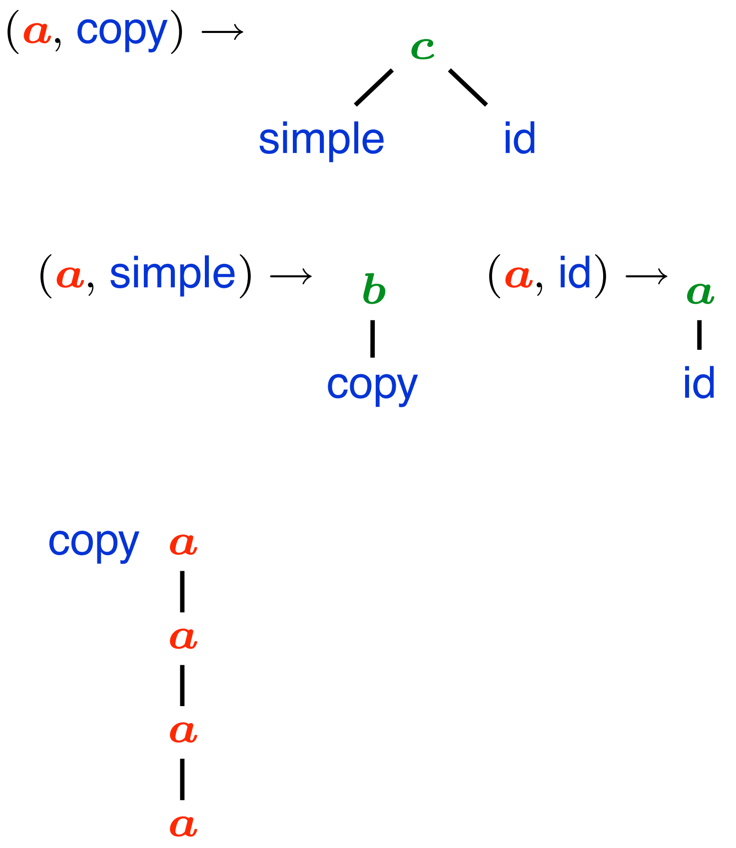
XSLT - copying

Example: 3 modes: simple, copy, id



XSLT - copying

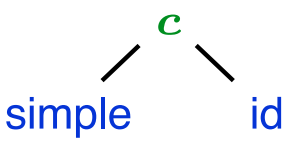
Example: 3 modes: simple, copy, id



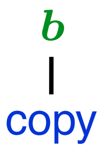
XSLT - copying

Example: 3 modes: simple, copy, id

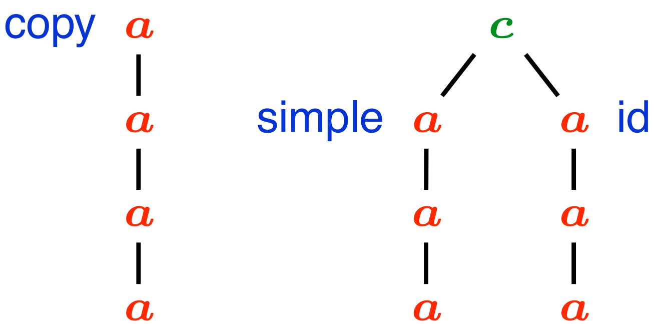
$(a, \text{copy}) \rightarrow$



$(a, \text{simple}) \rightarrow$



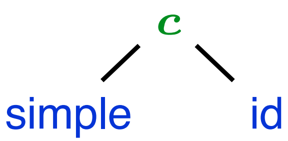
$(a, \text{id}) \rightarrow$



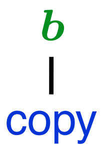
XSLT - copying

Example: 3 modes: simple, copy, id

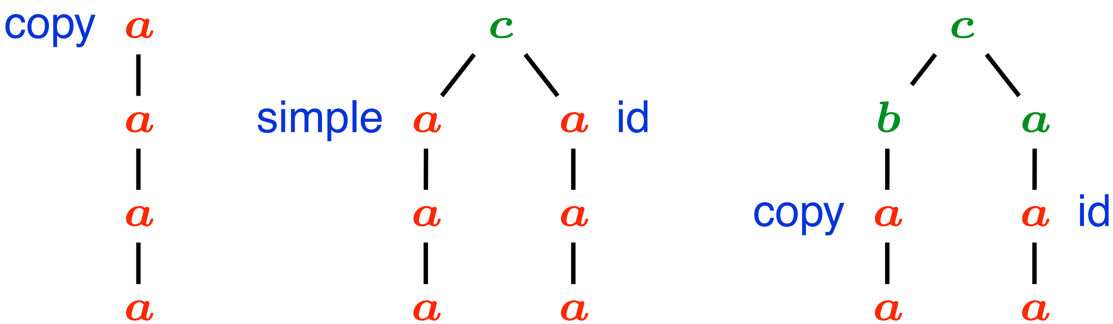
$(a, \text{copy}) \rightarrow$



$(a, \text{simple}) \rightarrow$

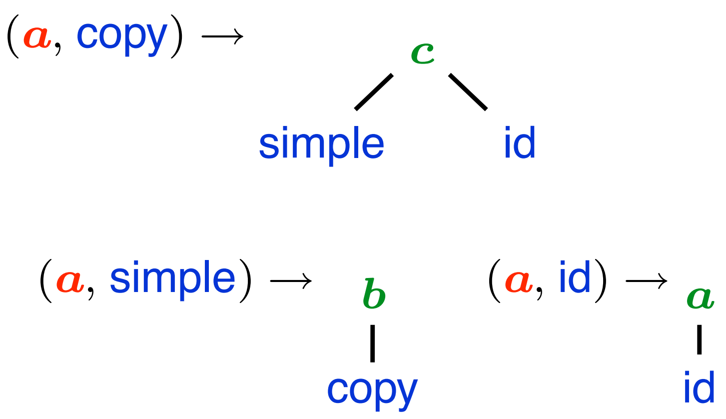


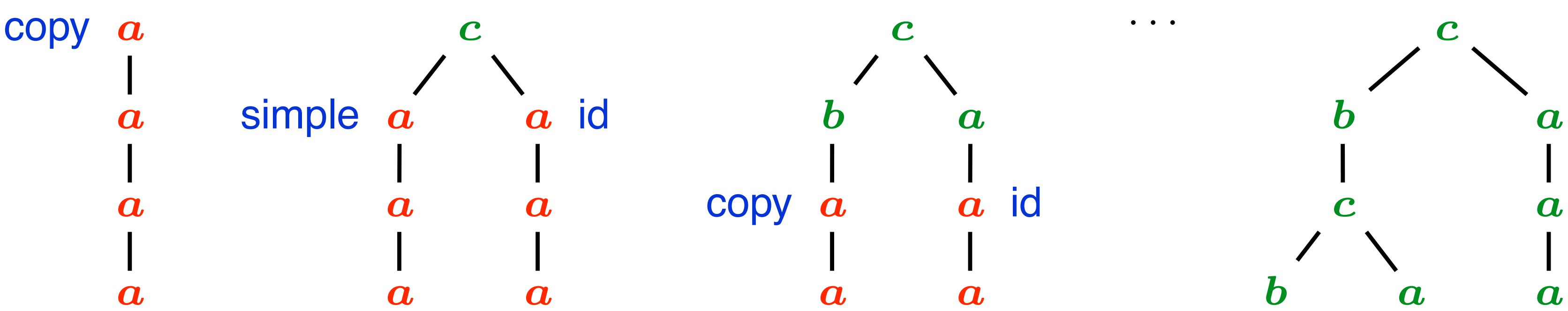
$(a, \text{id}) \rightarrow$



XSLT - copying

Example: 3 modes: simple, copy, id

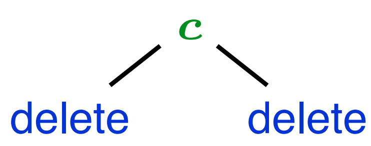




XSLT - deleting

Example: 2 modes: delete, copy

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

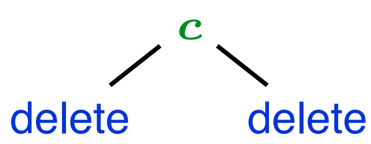


$(a, \text{delete}) \rightarrow \text{copy}$

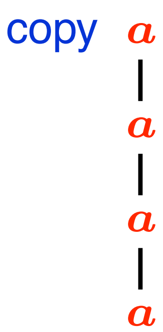
XSLT - deleting

Example: 2 modes: delete, copy

$(a,$ copy$) \to$



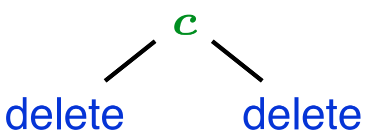
$(a,$ delete $) \to$ copy



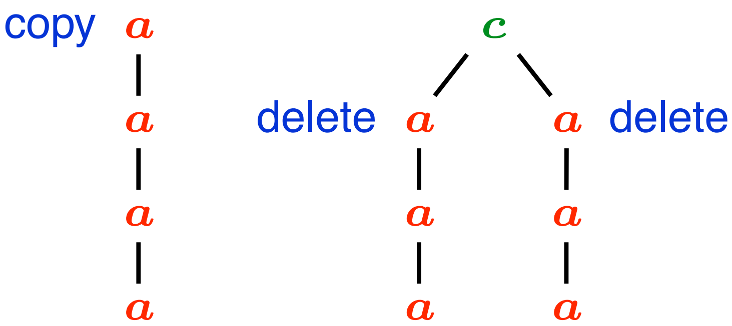
XSLT - deleting

Example: 2 modes: delete, copy

$(a,$ copy$) \rightarrow$



$(a,$ delete $) \rightarrow$ copy

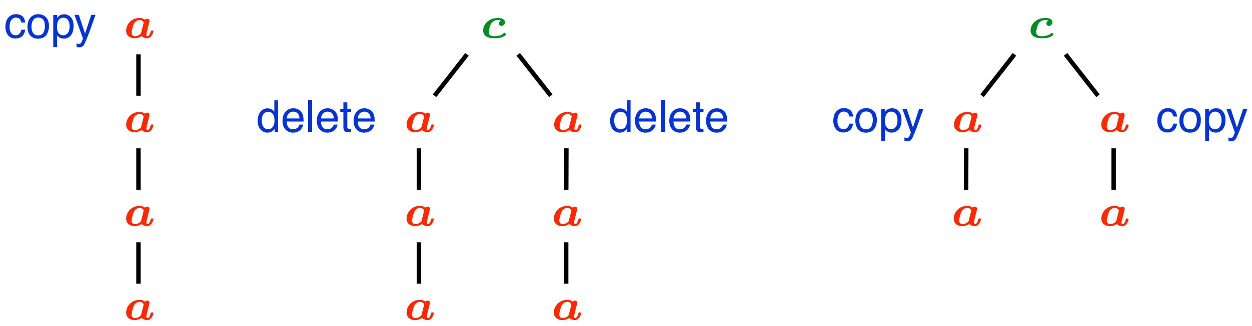


XSLT - deleting

Example: 2 modes: delete, copy

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

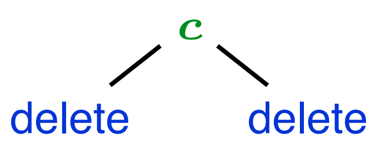
$(a, \text{delete}) \rightarrow \text{copy}$



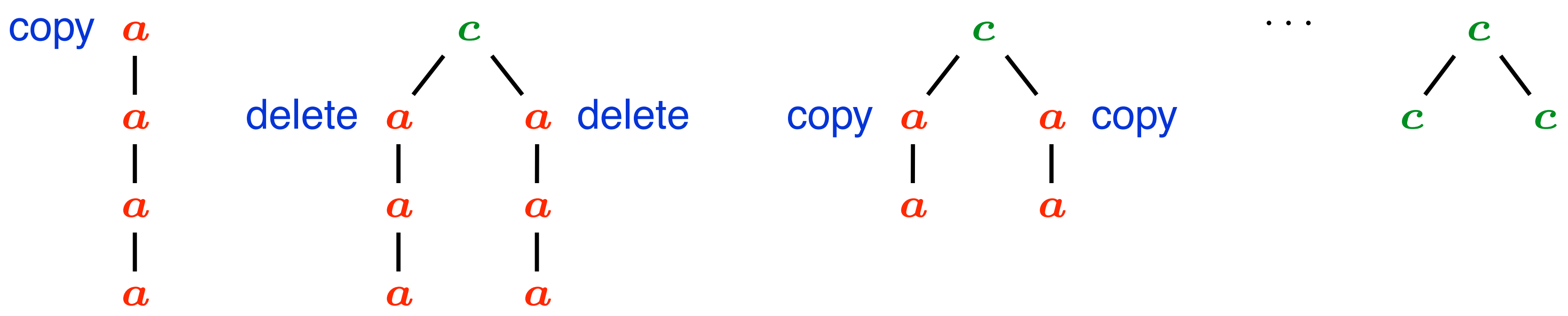
XSLT - deleting

Example: 2 modes: delete, copy

$(a,$ copy$) \rightarrow$



$(a,$ delete $) \rightarrow$ copy



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 τ_{in} DTD, TA
- output tree language τ_{out} DTD, TA
- XML-transformation T XSLT

The Typechecking Problem

Given:

- input tree language τ_{in} DTD, TA
- output tree language τ_{out} DTD, TA
- XML-transformation T XSLT

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?

	Tree Automata(NFA)	DTD(RE)	DTD(DFA)
deleting + copying	EXPTIME	EXPTIME	EXPTIME
no deleting + copying	EXPTIME	PSPACE	PSPACE
no deleting + bounded copying	EXPTIME	PSPACE	PTIME

Main Results

What is the complexity of the typechecking problem?

	Tree Automata(NFA)	DTD(RE)	DTD(DFA)
deleting + copying	EXPTIME	EXPTIME	EXPTIME
no deleting + copying	EXPTIME	PSPACE	PSPACE
no deleting + bounded copying	EXPTIME	PSPACE	PTIME

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 TA(2AFA) is in **PSPACE**.
2. Emptiness of TA(NFA) is in **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?

	Tree Automata(NFA)	DTD(RE)	DTD(DFA)
deleting + copying	EXPTIME	EXPTIME	EXPTIME
no deleting + copying	EXPTIME	PSPACE	PSPACE
no deleting + bounded copying	EXPTIME	PSPACE	PTIME

Copying, DTD(RE) $\rightarrow$ in PSPACE

Reduction to emptiness of TA(2AFA).

Copying, DTD(RE) $\rightarrow$ in PSPACE

Reduction to emptiness of TA(2AFA).

Construct a TA(2AFA) B such that

$$L(B) = \{t \in \tau_{in} \mid T(t) \notin \tau_{out}\}.$$

Copying, DTD(RE) $\rightarrow$ in PSPACE

Reduction to emptiness of TA(2AFA).

Construct a TA(2AFA) B such that

$$L(B) = \{t \in \tau_{in} \mid T(t) \notin \tau_{out}\}.$$

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

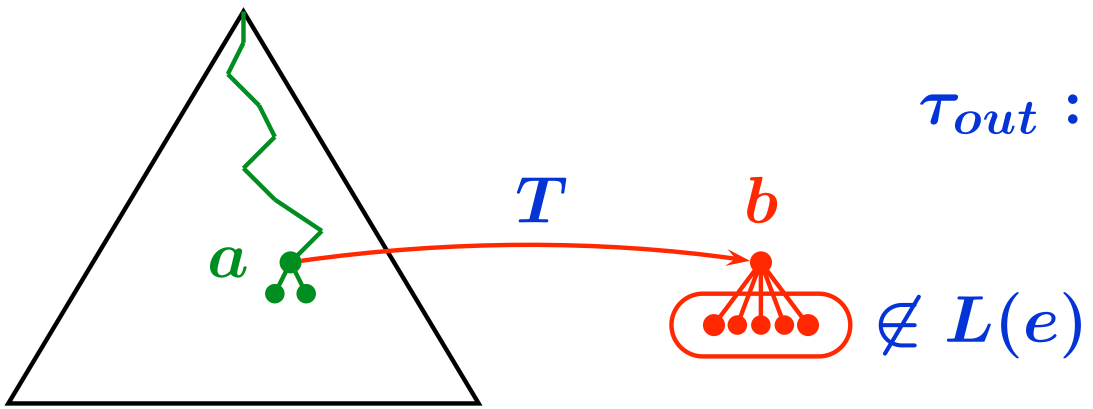
Copying, DTD(RE) $\rightarrow$ in PSPACE

Reduction to emptiness of TA(2AFA).

Construct a TA(2AFA) B such that

$$L(B) = \{t \in \tau_{in} \mid T(t) \notin \tau_{out}\}.$$

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



Main Results

What is the complexity of the typechecking problem?

	Tree Automata(NFA)	DTD(RE)	DTD(DFA)
deleting + copying	EXPTIME	EXPTIME	EXPTIME
no deleting + copying	EXPTIME	PSPACE	PSPACE
no deleting + bounded copying	EXPTIME	PSPACE	PTIME

Bound. Copying, DTD(DFA) $\rightarrow$ 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) $\longrightarrow$ 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 **PTIME**-complete.

In the future, we will try to expand the **PTIME**-result.