Foundations of XML Types: Tree Automata

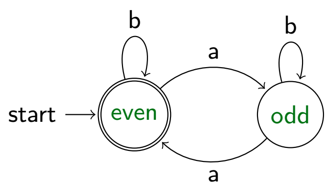
Pierre Genevès
CNRS

M2R – University of Grenoble, 2009–2010

Why Tree Automata?

- Foundations of XML type languages (DTD, XML Schema, Relax NG...)
- Provide a general framework for XML type languages
- A tool to define regular tree languages with an operational semantics
- Provide algorithms for efficient validation
- Basic tool for static analysis (proofs, decision procedures in logic)

Prelude: Word Automata

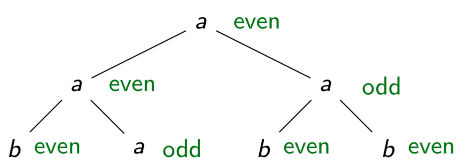


Transitions

even $\xrightarrow{a}$ *odd*
odd $\xrightarrow{a}$ *even*
...

From Words to Trees: Binary Trees

Binary trees with a even number of *a*'s



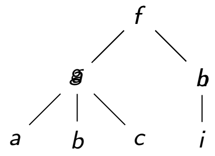
How to write transitions?

$(\text{even}, \text{odd}) \xrightarrow{a} \text{even}$
$(\text{even}, \text{even}) \xrightarrow{a} \text{odd}$
etc.

Ranked Trees?

They come from *parse trees* of data (or programs)...

A function call $f(a,b)$ $f(g(a,b,c),h(i))$ is a ranked tree



Ranked Alphabet

A *ranked alphabet symbol* is:

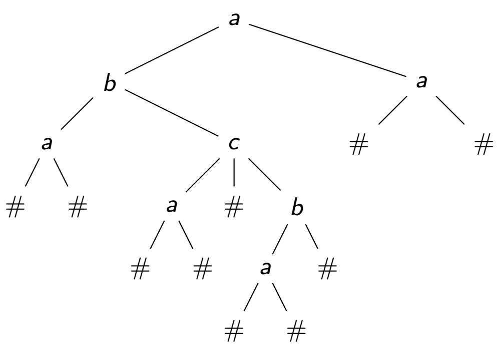
- a formalisation of a function call
- a symbol a with an integer arity(a)
  - arity(a) indicates the number of children of a

Notation

$a^{(k)}$: symbol a avec arity(a) $= k$

Example

Alphabet: $\{a^{(2)}, b^{(2)}, c^{(3)}, \#^{(0)}\}$
Possible tree:



Ranked Tree Automata

A ranked tree automaton A consists in:

Alphabet(A):	finite alphabet of symbols
States(A):	finite set of states
Rules(A):	finite set of transition rules
Final(A):	finite set of final states ($\subseteq$ States(A))

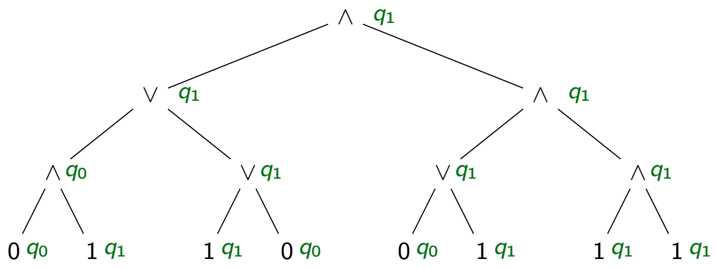
where :

Rules(A) are the form $(q_1, ..., q_k) \xrightarrow{a^{(k)}} q$

if $k = 0$, we will write $\epsilon \xrightarrow{a^{(0)}} q$

How do they work?

Example: Boolean Expressions



Principle

- Alphabet(A) $= \{\wedge, \vee, 0, 1\}$
- States(A) $= \{q_0, q_1\}$
- 1 accepting state at the root: Final(A) $= \{q_1\}$

Rules(A)

$\epsilon \xrightarrow{0} q_0$	$\epsilon \xrightarrow{1} q_1$
$(q_1, q_1) \xrightarrow{\wedge} q_1$	$(q_0, q_1) \xrightarrow{\vee} q_1$
$(q_0, q_1) \xrightarrow{\wedge} q_0$	$(q_1, q_0) \xrightarrow{\vee} q_1$
$(q_1, q_0) \xrightarrow{\wedge} q_0$	$(q_1, q_1) \xrightarrow{\vee} q_1$
$(q_0, q_0) \xrightarrow{\wedge} q_0$	$(q_0, q_0) \xrightarrow{\vee} q_0$

Terminology

- Language(A) : set of trees accepted by A
- For a tree automaton A, Language(A) is a *regular tree language* [Thatcher and Wright, 1968, Doner, 1970]

Example

Tree automaton A over $\{a^{(2)}, b^{(2)}, \#^{(0)}\}$ which recognizes trees with a even number of a's

Alphabet(A) : $\{a, b, \#\}$
States(A) : $\{\text{even}, \text{odd}\}$
Final(A) : $\{\text{even}\}$
Rules(A) :

$(\text{even}, \text{even}) \xrightarrow{a} \text{odd}$	$(\text{even}, \text{even}) \xrightarrow{b} \text{even}$
$(\text{even}, \text{odd}) \xrightarrow{a} \text{even}$	$(\text{even}, \text{odd}) \xrightarrow{b} \text{odd}$
$(\text{odd}, \text{even}) \xrightarrow{a} \text{even}$	$(\text{odd}, \text{even}) \xrightarrow{b} \text{odd}$
$(\text{odd}, \text{odd}) \xrightarrow{a} \text{odd}$	$(\text{odd}, \text{odd}) \xrightarrow{b} \text{even}$

$\epsilon \xrightarrow{\#} even$

Outline

- Can we implement a tree automaton efficiently? (notion of determinism)
- Are tree automata closed under set-theoretic operations?
- Can we check type inclusion?
- Can we build equivalent top-down tree automata?
- Nice theory. But... what should I do with my *unranked* XML trees?
- Can we apply this for XSLT type-checking?

Deterministic Tree Automata

Deterministic

does not have two rules of the form:

$$(q_1, ..., q_k) \xrightarrow{a^{(k)}} q$$
$$(q_1, ..., q_k) \xrightarrow{a^{(k)}} q'$$

for two different states q and q'

Intuition

At most one possible transition at a given node $\rightarrow$ implementation...

Can we Make a Tree Automaton Deterministic?

Theorem (determinisation)

From a given non-deterministic (bottom-up) tree automaton we can build a deterministic tree automaton

Corollary

Non-deterministic and deterministic (bottom-up) tree automata recognize the same languages.

Complexity

EXPTIME ($|\text{States}(A_{\text{det}})| = 2^{|\text{States}(A)|}$)

Implementing Validation

Membership Checking

Given a tree automaton A and a tree t, is $t \in \text{Language}(A)$?

Remark

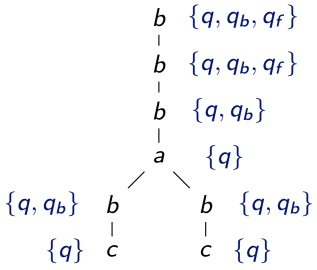
We can implement even if A is non-deterministic...

Example

Automaton with $\text{Final}(A) = \{q_f\}$ and :

$\epsilon \xrightarrow{c} q$ $\quad q \xrightarrow{b} q_b$ $\quad q \xrightarrow{b} q$

$q_b \xrightarrow{b} q_f$ $\quad (q, q) \xrightarrow{a} q$



Complexity

Membership-Checking is in PTIME (time linear in the size of the tree)

Set-Theoretic Operations

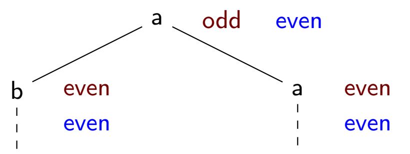
Recall

- We have seen that neither local tree grammars nor single-type tree grammars are closed under boolean operations (e.g. union)
- What about tree automata?

Closure under Union and Intersection...

Example

- Automaton A: even number of a's
  - $(\text{even}, \text{even}) \xrightarrow{a} \text{odd}$
- Automate B: even number of b's
  - $(\text{even}, \text{even}) \xrightarrow{a} \text{even}$



$((\text{even}, \text{even}), (\text{even}, \text{even})) \xrightarrow{a} (\text{odd}, \text{even})$

Product Construction

Given A and B, build $A \times B$

- $\text{Alphabet}(A \times B) = \text{Alphabet}(A) \cup \text{Alphabet}(B)$
- $\text{States}(A \times B) = \text{States}(A) \times \text{States}(B)$
- $\text{Final}(A \times B) = \{(q_a, q_b) \mid q_a \in \text{Final}(A) \wedge q_b \in \text{Final}(B)\}$
- $\text{Rules}(A \times B) =$
$$\left\{ ((q_a^1, q_b^1), \ldots, (q_a^k, q_b^k)) \xrightarrow{a^{(k)}} (q_a, q_b) \;\middle|\; \begin{array}{l} q_a^1, \ldots, q_a^k \xrightarrow{a^{(k)}} q_a \in \text{Rules}(A) \\ q_b^1, \ldots, q_b^k \xrightarrow{a^{(k)}} q_b \in \text{Rules}(B) \end{array} \right\}$$

Closure under Union

Given A and B, build $A \cup B$

- $\text{Alphabet}(A \cup B) = \text{Alphabet}(A) \cup \text{Alphabet}(B)$
- $\text{States}(A \cup B) = \text{States}(A) \times \text{States}(B)$
- $\text{Rules}(A \cup B) =$
$$\left\{ ((q_a^1, q_b^1), \ldots, (q_a^k, q_b^k)) \xrightarrow{a^{(k)}} (q_a, q_b) \;\middle|\; \begin{array}{l} q_a^1, \ldots, q_a^k \xrightarrow{a^{(k)}} q_a \in \text{Rules}(A) \\ q_b^1, \ldots, q_b^k \xrightarrow{a^{(k)}} q_b \in \text{Rules}(B) \end{array} \right\}$$
- $\text{Final}(A \cup B) = \{(q_a, q_b) \mid q_a \in \text{Final}(A) \vee q_b \in \text{Final}(B)\}$

Closure under intersection

Given A and B, build $A \cap B$

- $\text{Alphabet}(A \cap B) = \text{Alphabet}(A) \cup \text{Alphabet}(B)$
- $\text{States}(A \cap B) = \text{States}(A) \times \text{States}(B)$
- $\text{Rules}(A \cap B) =$
$$\left\{ ((q_a^1, q_b^1), \ldots, (q_a^k, q_b^k)) \xrightarrow{a^{(k)}} (q_a, q_b) \;\middle|\; \begin{array}{l} q_a^1, \ldots, q_a^k \xrightarrow{a^{(k)}} q_a \in \text{Rules}(A) \\ q_b^1, \ldots, q_b^k \xrightarrow{a^{(k)}} q_b \in \text{Rules}(B) \end{array} \right\}$$
- $\text{Final}(A \cap B) = \{(q_a, q_b) \mid q_a \in \text{Final}(A) \wedge q_b \in \text{Final}(B)\}$

Complexity of the Product

Size of the Result Automaton

- $|\text{States}(A \times B)| = |\text{States}(A)| \cdot |\text{States}(B)|$
- $|\text{Rules}(A \times B)| \leq |\text{Rules}(A)| \cdot |\text{Rules}(B)|$

Quadratic increase in size

Closure under Negation: Completion

Definition : Complete Tree Automaton

For each $a^{(k)} \in \text{Alphabet}(A)$ et $q_1, ..., q_k \in \text{States}(A)$, there exists a rule

$$(q_1, ..., q_k) \xrightarrow{a} q$$

with some q

Intuition

At least one transition at a given node...

Exemple

Incomplete (deterministic) tree automaton

Tree automaton A for $\{a(b, b)\}$:

$\epsilon \xrightarrow{b} q_b$
$(q_b, q_b) \xrightarrow{a} q_a$
with $\text{Final}(A) = \{q_a\}$

Completion of A, Complementation of A

Add a sink state q_p

$\epsilon \xrightarrow{b} q_b$ $\quad \epsilon \xrightarrow{a} q_p$
$(q_b, q_b) \xrightarrow{a} q_a \quad (q_b, q_a) \xrightarrow{a} q_p \quad (q_a, q_b) \xrightarrow{a} q_p \quad (q_a, q_a) \xrightarrow{a} q_p$
$(q_b, q_b) \xrightarrow{b} q_a \quad (q_b, q_a) \xrightarrow{b} q_p \quad (q_a, q_b) \xrightarrow{b} q_p \quad (q_a, q_a) \xrightarrow{b} q_p$
$(q_p, q) \xrightarrow{\sigma} q_p$ for all $q \in \{q_a, q_b, q_p\}$ et $\sigma \in \{a, b\}$
$(q, q_p) \xrightarrow{\sigma} q_p$ for all $q \in \{q_a, q_b, q_p\}$ et $\sigma \in \{a, b\}$
with $\text{Final}(A) = \{q_a\}$ $\text{Final}(\overline{A}) = \{q_b, q_p\}$

Closure under Negation: Summary

Building the Complement of A

- Make A deterministic
- Complete the result
- Switch final $\leftrightarrow$ non-final states

Complexity

- Determinisation of A : exponential explosion (states: $2^{\text{States}(A)}$)
- Completion of the result: exponential explosion of the number of rules: $|\text{Alphabet}(A)| \cdot \left(2^{|\text{States}(A)|}\right)^k$ where k is the maximal rank
- Switching final $\leftrightarrow$ non-final states : linear

Total: exponential explosion

Emptiness Test

Given a tree automaton A, is Language(A) $\neq \emptyset$?

Principle

Compute the set of reachable states and then see if any of them are in the final set

Complexity

PTIME (time proportional to $|A|$)

Application for Checking Type Inclusion

Type Inclusion

Given two tree automata A_1 and A_2, is Language(A_1) $\subseteq$ Language(A_2) ?

Theorem

Containment for non-deterministic tree automata can be decided in exponential time

Principle

- Language($A_1 \cap \overline{A_2}$) $\stackrel{?}{=} \emptyset$
- For this purpose, we must make A_2 deterministic (size: $O(2^{|A_2|})$)

→ EXPTIME

- Essentially no better solution [Seidl, 1990]

Top-Down Tree Automata

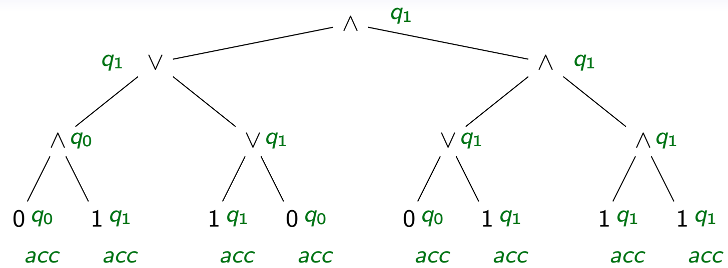
Is that useful?...

Example: Connection with Strings

abcd = a(b(c(d))) = a | b | c | d

Reading strings from left to right = reading trees top-down (→ e.g. streaming validation...)

Top-Down Tree Automata: Example



Principle

- starting from the root, guess correct values
- check at leaves
- 3 states: q_0, q_1, acc
- initial state at the root: q_1
- accepting if all leaves labeled acc

Transitions

$q_1 \xrightarrow{\wedge} (q_1, q_1)$ $q_1 \xrightarrow{\vee} (q_0, q_1)$

$q_0 \xrightarrow{\wedge} (q_0, q_1)$ $q_1 \xrightarrow{\vee} (q_1, q_0)$

$q_0 \xrightarrow{\wedge} (q_1, q_0)$ $q_1 \xrightarrow{\vee} (q_1, q_1)$

$q_0 \xrightarrow{\wedge} (q_0, q_0)$ $q_0 \xrightarrow{\vee} (q_0, q_0)$

$q_1 \xrightarrow{1} acc$ $q_0 \xrightarrow{0} acc$

Top-Down Tree Automata

A top-down tree automaton A consists in:

Alphabet(A):	finite alphabet of symbols
States(A):	finite set of states
Rules(A):	finite set of transition rules
Initial(A):	finite set of initial states ($\subseteq$ States(A))

où :

Rules(A) are of the form $q \xrightarrow{a^{(k)}} (q_1, ..., q_k)$

Top-down tree automata also recognize all regular tree languages

Top-down Determinism

Deterministic Top-Down Tree Automaton

- for each $q \in$ States(A) et $a \in$ Alphabet(A) there is at most one rule

$$q \xrightarrow{a^{(k)}} (q_1, ..., q_k)$$

- there is at most one initial state

Can We Make Top-Down Automata Deterministic?

(a) Yes (b) No (c) Maybe...

Can We Make Top-Down Automata Deterministic?

Maybe !

Deterministic top-down tree automata do not recognize all regular tree languages

Example

Initial(A) $= q_0$
$q_0 \xrightarrow{a} (q, q)$
$q \xrightarrow{b} \epsilon$
$q \xrightarrow{c} \epsilon$
reconnaît aussi...

Expressive Power of Tree Automata: Summary

Theorem

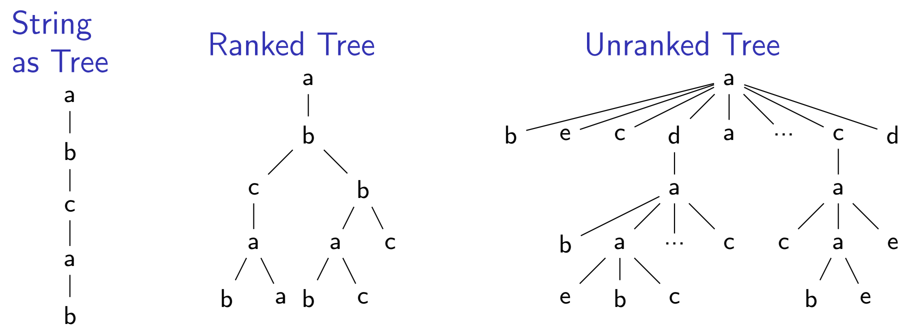
The following properties are equivalent for a tree language L:

(a) L is recognized by a bottom-up non-deterministic tree automaton
(b) $L$is recognized by a bottom-up deterministic tree automaton
(c) L is recognized by a top-down non-deterministic tree automaton
(d) L is generated by a regular tree grammar

Proof Idea

(a) $\Rightarrow$ (b): determinisation (see [Comon et al., 1997])
(a) $\Leftrightarrow$ (c): same thing seen from 2 different ways
(d) $\Leftrightarrow$ (a) : ? (horizontal recursion a^*?)

Unranked Trees



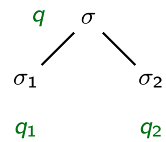
Unranked Tree Automata?

1. either we adapt ranked tree automata
2. or we encode unranked trees are ranked trees...

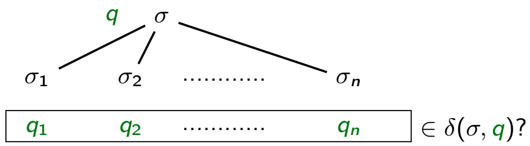
Unranked Tree Automata

Ranked Trees

Transitions can be described by finite sets:
$\delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots\}$



Unranked Trees

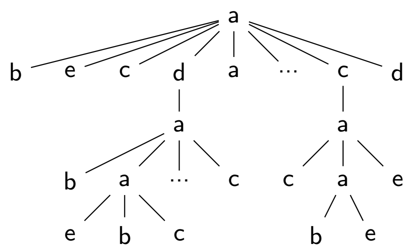


$\delta(\sigma, q)$

- For unranked trees, $\delta(\sigma, q)$ is a regular tree language
- $\delta(\sigma, q)$ may be specified by a regular expression or by a finite word automaton [Murata, 1999]

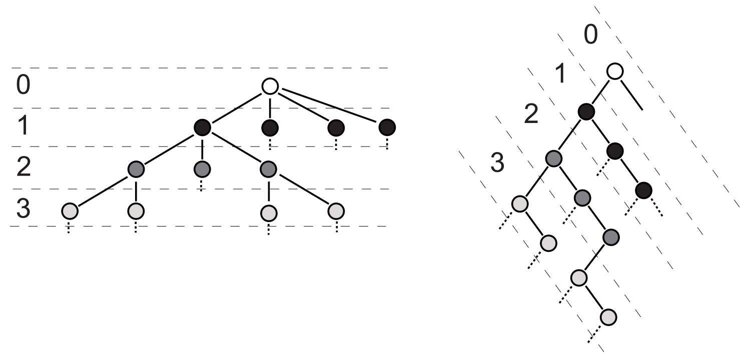
Second Option

Can we encode unranked trees as ranked trees?



?

Encoding Unranked Trees As Binary Trees



Bijective Encoding

- "first child; next sibling" encoding
- Allows to focus on binary trees without loss of generality
- Results for ranked trees hold for unranked trees as well

Tree Automata: Summary

Definition

A tree language is regular iff it is recognized by a non-deterministic tree automaton

Advantages

- Closure, decidable operations
- General tool (theoretical and algorithmic)

Limitations

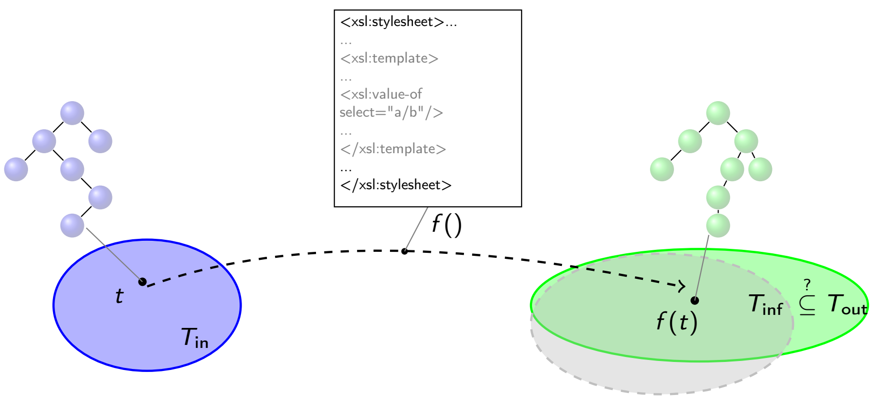
- $a^n b^n$

Application for Type-Checking

The XSLT Type-Checking Problem

Given a type T_{in}, an XSLT stylesheet f and a type T_{out}, does $f(t) \in T_{\text{out}}$ for all $t \in T_{\text{in}}$?

Application for XSLT Type-Checking



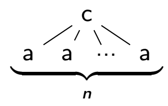
Approach

- Compute $T_{\text{inf}} = \{f(t) | t \in T_{\text{in}}\}$
- Check whether $T_{\text{inf}} \subseteq T_{\text{out}}$ holds
- In case $T_{\text{inf}} \subseteq T_{out}$ holds, then we know that for any $t \in T_{\text{in}}$, $f(t) \in T_{\text{out}}$

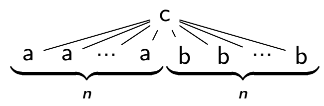
Limitation of the Approach

T_{inf} may not be regular:

Transform



into



Problem

- Approximation is required, e.g.: $a^n b^n$ approximated by $a^* b^*$
- Approximation is not contained in T_{out} (whereas the real type is)
- There is no "good" approximation...
- Consequence: this approach yields *static type-checkers* which are not complete: some correct transformations might be rejected.

Backward Type Inference for XSLT

Modified Approach

- Compute $T_{\mathrm{inf}} = \{f^{-1}(t) | t \in T_{\mathrm{out}})$
- Check whether $T_{\mathrm{in}} \subseteq T_{\mathrm{inf}}$ holds

Theorem and Research Prototype

Static type-checking is decidable for an XSLT fragment: "XSLT0" [Tozawa, 2001]

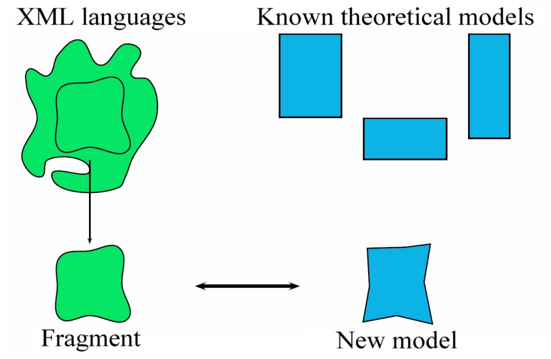
- Inference of the input tree automaton (PTIME)
- Containment of tree automata (EXPTIME)

Limitation

- Only basic transformations are supported (no real XPath)

Concluding Remarks

- Tree automata are part of the theoretical tools that provide the underlying guiding principles for XML (like the relational algebra provide the underlying principles for relational databases)
- Still a lot of research ongoing on the topic, important challenges remain



</session>

A few pointers for the curious who want to learn more...

- Sheaves automata [Dal-Zilio and Lugiez, 2003]
(how to model efficiently unordered content, e.g. XML attributes, or interleaving/shuffle operator)
- Visibly pushdown automata [Alur and Madhusudan, 2004]
(beyond regular tree languages)
- A powerful and efficient modal tree logic [Genevès et al., 2007]
(how to support regular tree languages and XPath too)

Questions / discussions...?

Alur, R. and Madhusudan, P. (2004).
Visibly pushdown languages.
In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 202–211, New York, NY, USA. ACM.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (1997).
Tree automata techniques and applications.
Available on: http://www.grappa.univ-lille3.fr/tata.
release October, 1st 2002.

Dal-Zilio, S. and Lugiez, D. (2003).
XML schema, tree logic and sheaves automata.
In Nieuwenhuis, R., editor, *RTA'03: Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 246–263. Springer.

Doner, J. (1970).
Tree acceptors and some of their applications.
Journal of Computer and System Sciences, 4:406–451.

Genevès, P., Layaïda, N., and Schmitt, A. (2007).
Efficient static analysis of XML paths and types.

In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 342–351, New York, NY, USA. ACM Press.

Murata, M. (1999).
Hedge automata: a formal model for XML schemata.
http://www.xml.gr.jp/relax/hedge_nice.html.

Seidl, H. (1990).
Deciding equivalence of finite tree automata.
SIAM J. Comput., 19(3):424–437.

Thatcher, J. W. and Wright, J. B. (1968).
Generalized finite automata theory with an application to a decision problem of second-order logic.
Mathematical Systems Theory, 2(1):57–81.

Tozawa, A. (2001).
Towards static type checking for XSLT.
In *DocEng '01: Proceedings of the 2001 ACM Symposium on Document Engineering*, pages 18–27, New York, NY, USA. ACM Press.