Introduction to Tree Automata, with an application to XML schemas

Johannes Dellert
Seminar für Sprachwissenschaft, Universität Tübingen
03.12.2009
String languages and tree languages

String languages

- are sets of strings, i.e. linear sequences over an alphabet
- are recognized by string automata such as FSA, PDA, TMs

Tree languages

- are sets of trees, i.e. bracketed structures over an alphabet
- are recognized by tree automata such as RTA and PDTA
- strings can be seen as non-branching trees, so every string language is also a tree language which are thus more general
- many nice formal properties of string languages carry over
Regular tree grammars (RTG)

Definition

A **regular tree grammar (RTG)** is defined by the tuple $G = (N, \Sigma, Z, P)$ where

- $N$ is a set of nonterminal symbols
- $\Sigma$ is a ranked alphabet disjoint from $N$
- $Z \in N$ is the starting nonterminal
- $P$ is a set of productions of the form $A \rightarrow t$ with $A \in N$ and $t \in T_\Sigma(N)$
Regular tree grammars (RTG)

Variant for describing XML Schemata

Definition

A regular tree grammar is defined by the tuple $G = (\Sigma, D, N, P, n_s)$ where

- $\Sigma$ is a finite set of element types
- $D$ is a finite set of data types
- $N$ is a finite set of non-terminals
- $P$ is a finite set of productions of the form $n \rightarrow a(r)$ with $n \in N$, $a \in \Sigma$, and either $r = \omega \in D$ or $r$ is a regular expression over $N$
- $n_s \in N$ is the starting symbol

The grammar allows a document tree $t$ if it can be produced from $n_s$ using $P$ and does not contain any elements of $N$. 
Regular tree grammars (RTG)

Example: Translation of an XML DTD

Example: A DTD for a class of XML documents

```xml
<!ELEMENT body (paper*)>
<!ELEMENT paper (title,author*,journal?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT journal (#PCDATA | EMPTY)>
```

Example: The corresponding RTG

\[ \Sigma := \{ \text{body}, \text{paper}, \text{title}, \text{author}, \text{journal} \}, \quad D := \{ \#\text{PCDATA}, \epsilon \}, \]
\[ N := \{ n_b, n_p, n_a, n_t, n_j \}, \quad n_s := n_b, \text{ and} \]
\[ P := \{ n_b \to \text{body}(n_p^*), n_p \to \text{paper}(n_t n_a^* n_j?), n_t \to \text{title}(\#\text{PCDATA}), n_j \to \text{journal}(\#\text{PCDATA}), n_j \to \text{journal}(\epsilon) \}\]
Definition

A (bottom-up) finite tree automaton is defined by the tuple $M = (\Sigma, D, Q, \delta, F)$ where

- $\Sigma$ and $D$ are finite sets of element types and data types
- $Q$ is a finite set of states, $F \subset Q$ the set of final states
- $\delta$ is a function $\delta : \Sigma \times E \rightarrow Q$, where either $E \in D$ or $E$ is a regular expression over $Q$

Recognition Procedure

- annotate nodes in tree structure with state symbols
- begin by annotating leaves, moving upwards in the structure and making decisions on which rule to apply
- a tree is accepted iff its root can be annotated with one of the final states in this manner
Example: Automaton encoding an RTG

Example: An RTG for a class of XML documents

\( \Sigma := \{ \text{section, paragraph} \}, \ D := \{ \#\text{PCDATA} \}, \)
\( N := \{ n_1, n_2, n_p \}, \ n_s := n_1, \) and
\( P := \{ n_1 \rightarrow \text{section}(n_2 \ast n_p^*), n_2 \rightarrow \text{section}(n_p), n_p \rightarrow \text{paragraph}(\#\text{PCDATA}) \} \)

Example: An RTA recognizing this language

\( \Sigma := \{ \text{section, paragraph} \}, \ D := \{ \#\text{PCDATA} \}, \)
\( Q := \{ q_1, q_2, q_p \}, \ F := \{ q_1 \}, \) and \( \delta \) such that
\( \delta(\text{section}, q_2 \ast q_p^*) = q_1, \)
\( \delta(\text{section}, q_p^*) = q_2, \) and
\( \delta(\text{paragraph}, \#\text{PCDATA}) = q_p \)
the example automaton was a non-deterministic automaton because it allowed a choice of rules at some point

analogously to the set of current states during the run of a non-deterministic FSA, a step of an automaton can be seen as annotating a tree node with a set of state symbols

in the case of bottom-up finite tree automata, the determinisation algorithm for FSAs can easily be generalized

this means that non-deterministic and deterministic bottom-up finite tree automata are equally powerful
Some results: Automata types

Bottom-Up and Top-Down Automata

Top-Down Automata

- it is also possible to define tree automata that process trees starting at the root and moving down
- instead of final states, we define a set $I \subseteq Q$ of initial states
- rules have reverse format of the rules for bottom-up automata
- top-down automata are useful for checking trees while they are being constructed

Determinism and Top-Down Automata

- a deterministic top-down tree automaton has to decide which rule to apply to a parent without inspecting its children
- therefore, deterministic **top-down** finite tree automata are **strictly less powerful** than non-deterministic ones
Some results on Tree Languages

Closure Properties of tree languages

Definition
A tree language is \textbf{recognizable} iff there exists a finite tree automaton accepting that language.

Theorem
The set of recognizable tree languages is \textbf{closed under union}.

Theorem
The recognizable tree languages are \textbf{closed under complement}.

Theorem
The recognizable tree languages are \textbf{closed under intersection}.
Conclusion

- Tree automata are a generalization of string automata.
- They are useful in defining and efficiently checking membership in classes of tree structures (XML schemata, grammars).
- They have become very popular for implementations of query languages on tree-structured data (e.g., XML documents).

Outlook

- **Weighted tree automata** are used e.g. in grammar induction from tree banks as an alternative to PCFGs.
- **Tree transducers** can be used to define and calculate with changes on trees, e.g. in document standardization or MT.
- **Pushdown tree automata** are even more powerful than finite tree automata by employing a stack of subtree structures.
Further reading

My source for the examples

Boris Chidlovskii (1999): Using Regular Tree Automata as XML schemas

The most popular reference work, with all the proofs

Thank you!