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 (RTGs)

Definition

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

- N is a set of nonterminal symbols
- Σ 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 $t \in T_\Sigma(N)$

Variant for describing XML Schemata

Definition

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

- Σ 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 $r = w \in D$ 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.

Example: Translation of an XML DTD

Example: A DTD for a class of XML documents

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

Example: The corresponding RTG

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

Regular tree automata (RTAs)

Definition

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

- Σ 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
- δ is a function $\delta : \Sigma \times E \longrightarrow Q$, 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

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

Example: An RTA recognizing this language

$\Sigma := \{section, paragraph\}$, $D := \{\#PCDATA\}$,
$Q := \{q_1, q_2, q_p\}$, $F := \{q_1\}$, and δ such that
$\delta(section, q_2 * q_p*) = q_1$,
$\delta(section, q_p*) = q_2$, and
$\delta(paragraph, \#PCDATA) = q_p$

Determinism and Non-Determinism

Determinism and Non-Determinism

- 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**

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 $I \subset Q$ 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

Closure Properties of tree languages

Definition

A tree language is **recognizable** iff there exists a finite tree automaton accepting that language.

Theorem

The set of recognizable tree languages is **closed under union**.

Theorem

The recognizable tree languages are **closed under complement**.

Theorem

The recognizable tree languages are **closed under intersection**.

Conclusion and Outlook

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)
- 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

H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi (2007): **Tree Automata Techniques and Applications** (common abbreviation: TATA)

The End

Thank you!