Finite State Tree Automata and Transformation-Based Parsing[1]

Giorgio Satta
University of Padua
Italy

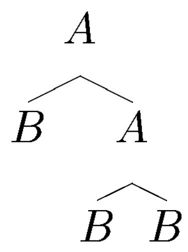
[1]Work done in collaboration with Eric Brill, Johns Hopkins University, Baltimore, MD

Outline

1. finite state tree automata
2. tree automata and tree pattern matching
3. transformation-based parsing
4. parsing using tree automata

Trees

We consider ordered, labeled trees



Standard terminology: root, leaf, internal node, father, child, sibling, yield

Rank of a node: number of its children (leaves have rank zero)

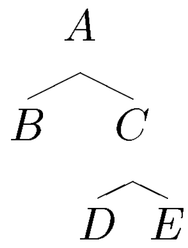
Binary trees: trees whose nodes have rank equal to zero or two

Walks

A **walk** on a tree is the process of visiting each of its nodes

In a **top-down** walk each node is visited before its childern;
in a **bottom-up** walk each node is visited after its childern

Example:



The sequence $BDECA$ corresponds to a bottom-up walk

Finite State Tree Automata

Generalization of FSA to trees first introduced in (Thatcher 1967); a good overview of known results is presented in (Gecseg and Steinby 1984)

Four types of finite state tree automata, depending on two parameters:

- state transition (deterministic/nondeterministic)
- tree walk (bottom-up/top-down)

	det	ndet
bu	DFR	NFR
td	DRF	NRF

Example

We trace a bottom-up deterministic automaton reading input tree

$$\begin{array}{c} A \\ \begin{array}{cc} B & C \end{array} \end{array}$$

When reading a leaf the automaton is always in the initial state

$$\begin{array}{c} A \\ \begin{array}{cc} B : q_1 = \delta(q_0, B) & C : q_2 = \delta(q_0, B) \end{array} \end{array}$$

When reading an internal node, the automaton is in all the states reached after having read each child node

$$\begin{array}{c} A : q_3 = \delta(q_1 q_2, A) \\ \begin{array}{cc} B : q_1 = \delta(q_0, B) & C : q_2 = \delta(q_0, B) \end{array} \end{array}$$

DFR Automata

Definition A **deterministic frontier-to-root** (DFR) automaton is a 5-tuple

$$A = (Q, \Sigma, \delta, q_0, F)$$

where

(i) Q is a finite set of states

(ii) Σ is a finite alphabet

(iii) $q_0 \in Q$ is the initial state

(iv) $F \subseteq Q$ is the set of final states

(v) δ is a finite transition function from $Q^* \times \Sigma$ to Q

Example

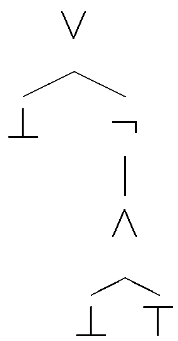
The propositional formulae with no variables are

(i) $\top$, $\bot$

(ii) $(F \vee G)$, $(F \wedge G)$ and $\neg F$,
where F and G are propositional formulae with no variables

The above formulae can be represented as trees:

$$(\bot \vee \neg(\bot \wedge \top)) \quad =$$



The truth values assigned to these formulae are

(i) $\top = \mathbf{true}$, $\bot = \mathbf{false}$

(ii) $F \vee G = \mathbf{false}$ iff $F = G = \mathbf{false}$

(iii) $F \wedge G = \mathbf{true}$ iff $F = G = \mathbf{true}$

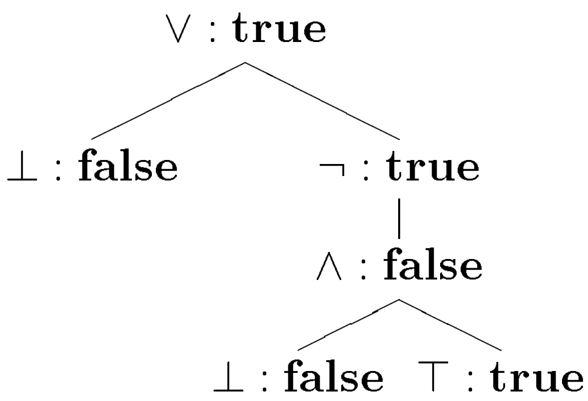
(iv) $\neg F = \mathbf{true}$ iff $F = \mathbf{false}$

Example (cont'd)

The DRF automaton below accepts all propositional formulae with no variables, evaluating to true (δ partially specified)

$$\Sigma = \{\vee, \wedge, \neg, \top, \bot\},\ Q = \{q_0, \mathbf{false}, \mathbf{true}\},\ F = \{\mathbf{true}\}$$

1. $\delta(q_0, \top) = \mathbf{true}$
   $\delta(q_0, \bot) = \mathbf{false}$
2. $\delta(\mathbf{false}\ \mathbf{false}, \vee) = \mathbf{false}$
   $\delta(\mathbf{false}\ \mathbf{true}, \vee) = \mathbf{true}$
   $\vdots$
3. $\delta(\mathbf{true}\ \mathbf{true}, \wedge) = \mathbf{true}$
   $\delta(\mathbf{false}\ \mathbf{true}, \vee) = \mathbf{true}$
   $\vdots$
4. $\delta(\mathbf{false}, \neg) = \mathbf{true}$
   $\delta(\mathbf{true}, \neg) = \mathbf{false}$

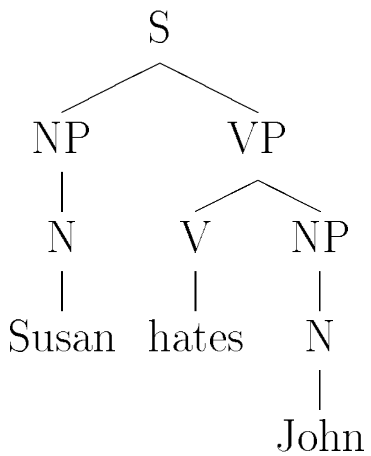


Example

Consider the context-free grammar specified by the productions:

1. S $\rightarrow$ NP VP
2. VP $\rightarrow$ V NP
3. NP $\rightarrow$ N
4. N $\rightarrow$ Susan, N $\rightarrow$ Mary, N $\rightarrow$ John
5. V $\rightarrow$ hates, V $\rightarrow$ loves

The above grammar derives trees of the kind:



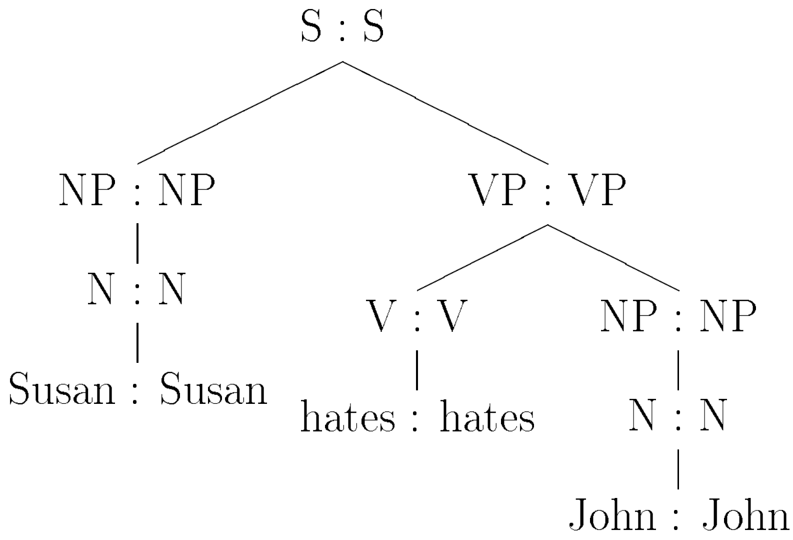
Example (cont'd)

The trees generated by the previous grammar are accepted by the following DFR automaton

$$\Sigma = \{\text{Susan}, \text{hates}, ...\},\ Q = \{\text{V}, \text{VP}, ...\},\ F = \{\text{S}\}$$

1. for every lexical element: Susan, ...
   $\delta(q_0, \text{Susan}) = \text{Susan},$
   $\vdots$

2. for every production: N $\rightarrow$ Susan, S $\rightarrow$ NP VP, ...
   $\delta(\text{Susan}, \text{N}) = \text{N},$
   $\delta(\text{NP VP}, \text{S}) = \text{S},$
   $\vdots$



Accepted Forest

$\hat{\delta}(n)$ is the state **reached** by the automaton upon reading node n

We specify $\hat{\delta}$ as:

(i) if n is a leaf labeled by B
$\hat{\delta}(n) = \delta(q_0, B)$

(ii) if n is labeled by B and has children $n_1, \ldots, n_m$
$\hat{\delta}(n) = \delta(\hat{\delta}(n_1) \cdots \hat{\delta}(n_m), B)$

The **tree forest** accepted by the automaton is the set

$$F(A) \;=\; \{T \mid T \text{ has root } n,\; \hat{\delta}(n) \in F\}$$

The Class REC

REC is the class of all forests accepted by some DFR automaton.

A forest is in REC if and only if it is the set of all derivation trees of some context-free grammar

REC is closed under the following operations

- union
- intersection
- set-difference (and complementation)

NFR Automata

Definition A **nondeterministic frontier-to-root** (NFR) automaton is a 5-tuple

$$A = (Q, \Sigma, \delta, q_0, F)$$

where

(i) Q, Σ, q_0 and F are as for a DFR automaton

(ii) δ is a finite transition function from $Q^* \times \Sigma$ to 2^Q

At each state transition an NFR automaton might have several choices

Accepted Forest

$\hat{\delta}(n)$ is the set of states **reached** by the automaton upon reading node n

We specify $\hat{\delta}$ as:

$$q \in \hat{\delta}(n)$$

if one of the following conditions holds

(i) n is a leaf labeled by B and $q \in \delta(q_0, B)$

(ii) n is labeled by B and has children $n_1, \ldots, n_m$, and there exists $q_i \in \hat{\delta}(n_i)$, $1 \leq i \leq m$ such that $q \in \delta(q_1 \cdots q_m, B)$

The **tree forest** accepted by the automaton is the set

$$F(A) \ = \ \{T \mid T \text{ has root } n,\ \hat{\delta}(n) \cap F \neq \emptyset\}$$

DFR and NFR Automata

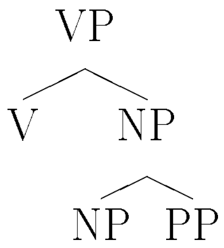
A forest is accepted by a NFR automaton if and only if it is accepted by some DFR automaton

To convert an NFR automaton into an equivalent DFR automaton we use the standard power set construction. States of the resulting DFR automaton are therefore sets of states of the source NFR automaton

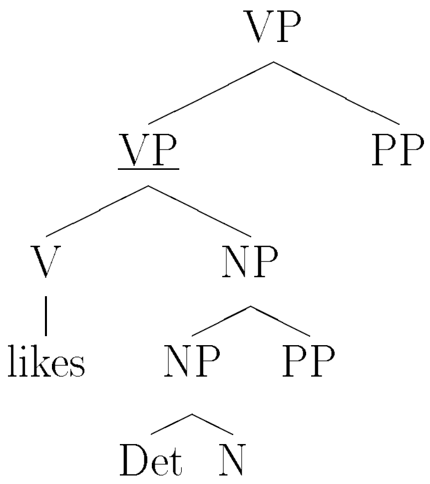
Tree Matching

Tree P matches tree T at the node with the underlined label

P:



T:



Tree Matching (cont'd)

Definition A tree P **matches** a tree T at node n if there exists a mapping from nodes of P to nodes of T such that:

(i) if n' maps to n'', n' and n'' have the same label

(ii) the root of P maps to n

(iii) if n' maps to n'' and n' is not a leaf in P, then n' and n'' have the same rank and the i-th child of n' maps to the i-th child of n''

P is called **pattern**, T is called **text** and n is called **matching node**

Tree Pattern Matching

Multi-pattern problem: Let $\mathcal{P}$ be a set of pattern trees.

- input: a text tree T
- output: for each node n of T, indicate all trees in $\mathcal{P}$ that match at n

A bottom-up algorithm for the multi-pattern problem has been presented in (Hoffmann and O'Donnell 1982); we reformulate the algorithm using tree automata

Matching Automata

A **matching automaton** is a tree automaton that "finds" all matches of given pattern trees in a text tree

We construct a matching NFR automaton for $\mathcal{P} = \{P_1, \ldots, P_q\}$ (assume the P_i's are binary trees)

States of the automaton are q_0 and the nodes of the P_i's; δ is specified as

(i) $n \in \delta(q_0, B)$
if n is a leaf of some P_i labeled by B

(ii) $n \in \delta(n'n'', B)$
if n is labeled by B and

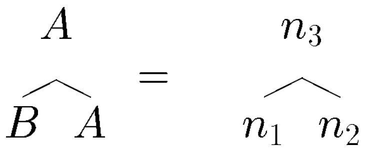
(a) n is the father of n' and n'', or

(b) n is a leaf of some P_i

Example

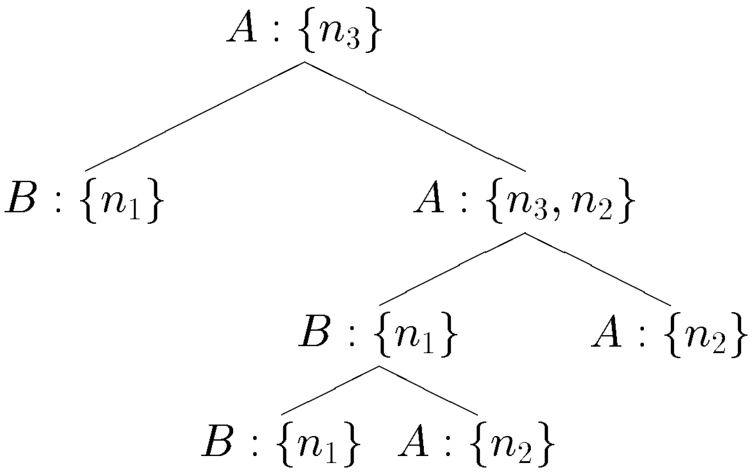
$\mathcal{P} = \{P_1\}$

P_1:



Tree T with states reached by matching automaton

T:



Matching Automata (cont'd)

The NFR matching automaton can be transformed into a DFR matching automaton using the power set construction; the resulting automaton has states that are sets of nodes of the P_i's

Proposition 1 [Match] Let q be the state reached by the DFR matching automaton upon reading node n of a text tree T. P_i matches T at n if and only if the root node of P_i belongs to q

Transformation-Based Parsing (TBP)

(Brill 1993)

The grammar can be trained automaticaly from tree data banks

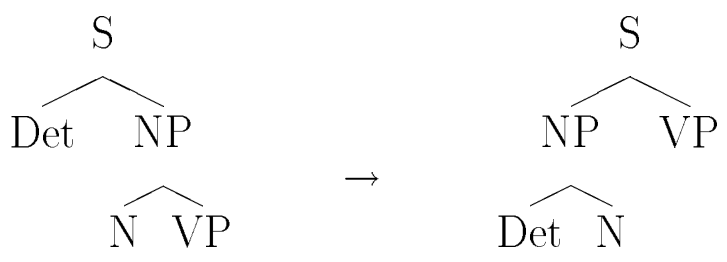
The parser always assigns a unique parse tree to the imput string

Competitive parsing accuracy on naturally occurring text

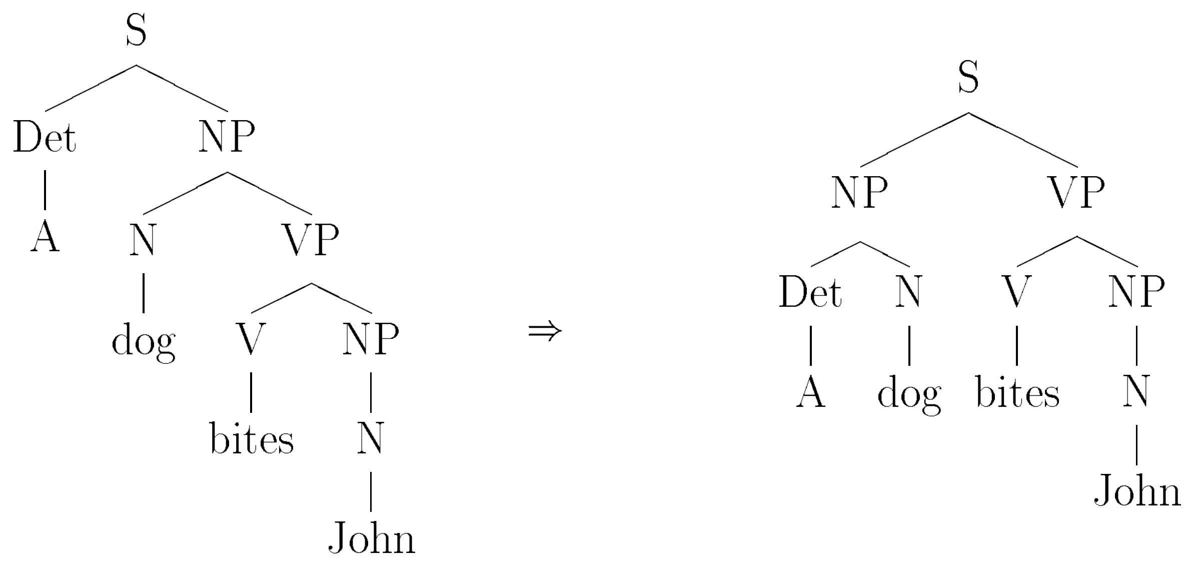
Parsing takes an amount of time proportional to the length of the input string

Transformations

Transformations are tree rewriting rules $Q \to Q'$, where Q, Q' are trees with equal number of leaves



Transformations are applied at matching nodes for the left-hand side



TBP (cont'd)

The **grammar** consists of a finite ordered sequence of transformations

Parsing prescription:

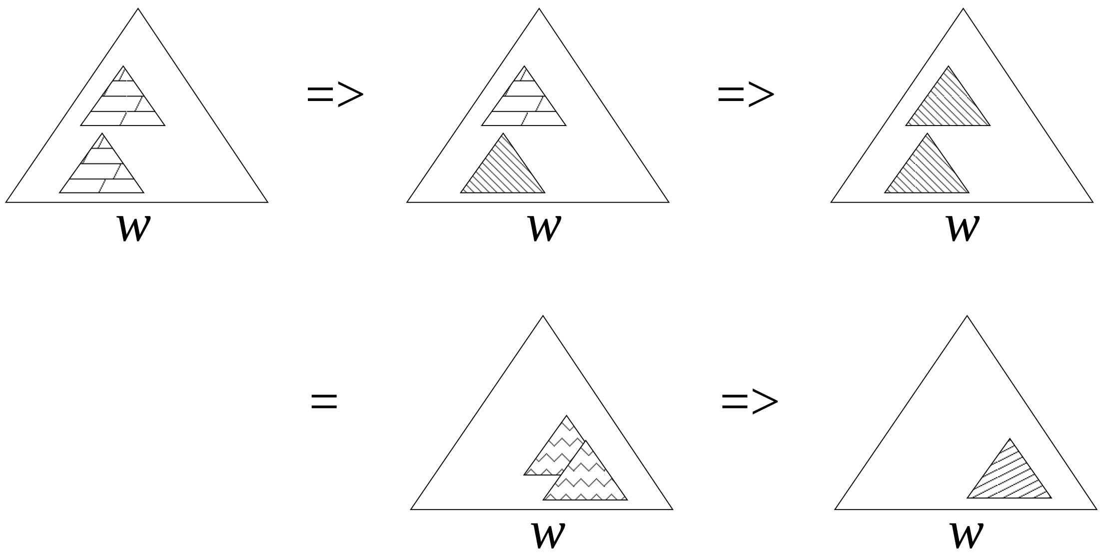
- assign a default parse to input sentence
- rewrite parse tree applying the transformations in the given order (no recursion)
- each transformation is applied to the matching nodes in a bottom-up fashion (no autofeed)

Example

Grammar:

1. 2.

Tree rewriting:



Problem

In principle, every transformation could be applied at every node of the tree. If

$$\begin{aligned} \pi &= \text{number of transformations} \\ w &= \text{input string} \end{aligned}$$

then a naive algorithm has to check a number of matchings proportional to $\pi|w|$

In practical applications

- less than $|w|$ matchings are successful
- π is several hundreds large

Then the naive algorithm wastes a lot of time with unsuccessful matchings

Efficient Parsing

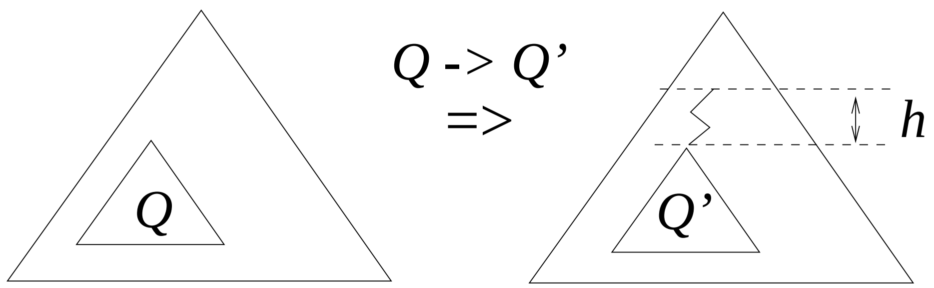
We develop an efficient parsing algorithm that behaves as if the successful matchings were known in advance

The basic idea is to

- construct a DFR matching automaton for the set of left-hand sides of the transformations
- associate with each node of the tree to be rewritten the state reached by the automaton

Update

The **height** of a tree is the length (# of arcs) of the longest path from the root to some leaf. Let h be the maximum height of the transformation left-hand sides



Proposition 2 [Update] After Q has been replaced with Q', only Q' and the closest h ancestors of its root need to be reread by the automaton

Proof outline:
$\hat{\delta}(n)$ does not change if

- n does not dominate the root of Q
- n is the i-th ancestor of the root of Q, with $i > h$

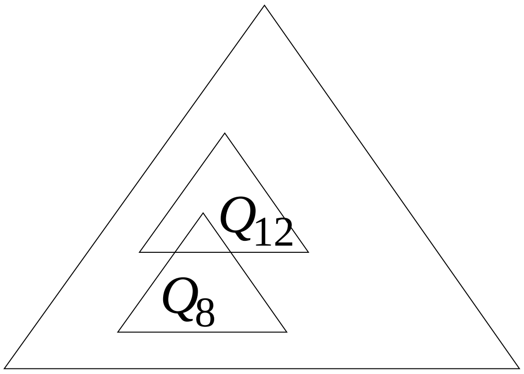
Algorithm

We use an agenda to schedule matching transformations according to their application priority

Read tree and update Agenda
while(Agenda $\neq \emptyset$) **do**
    $(T, n) \leftarrow$ Agenda
    if T matches at n **then do**
        apply T
        local update
    od
od

False Matchings

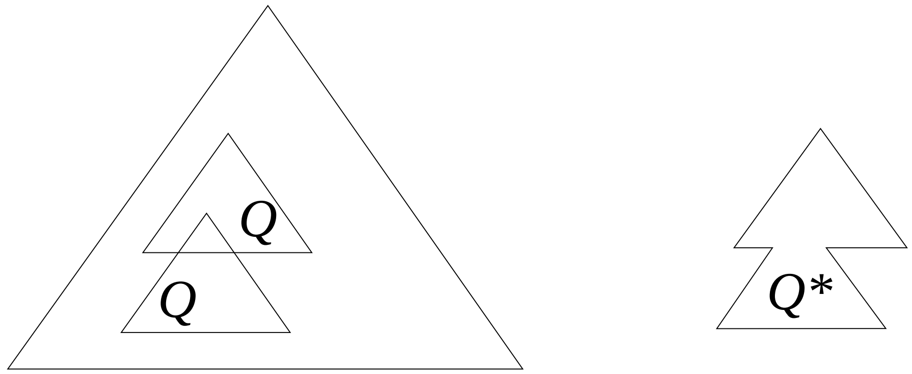
Transformations already scheduled in the Agenda might no longer match:



The cost of "killing" the false matchings is amortized by the successful matchings

Bottom-Up requirement

The bottom-up requirement on transformation application must be observed in cases of overlapping left-hand sides



The requirement can be enforced by grammar preprocessing

Remarks

The algorithm can easily be extended to transformations with don't care symbols

The algorithm can also be applied to strings for the solution of part-of-speech tagging. In this case the compilation phase corresponds to the construction of the DFA used in (Aho and Corasick 1975)

DFR Automata (cont'd)

We can use a generalization of DFR automata that reduces size of δ

We define δ as a transition function from $Q^2 \times \Sigma$ to Q

Current state at node n is computed on the basis of:

- immediate left sibling
- rightmost child