# Preserving Trees in Minimal Automata

**Jan Daciuk**
Gdańsk University of Technology
jandac@eti.pg.gda.pl

## Abstract

We present a method to store additional information in a minimal automaton so that it is possible to compute a corresponding tree node number for a state. The number can then be used to retrieve additional information. The method works for minimal (and any other) deterministic acyclic finite state automata (DFAs). We also show how to compute the inverse mapping.

## 1 Introduction

Deterministic finite state automata and transducers are widely used in natural language processing and computational linguistics. The most frequent uses include dictionaries (Daciuk et al., 2010), acoustic and language models (the latter indirectly, using perfect hashing to number words), as well as hidden Markov models used for tagging and chunking. Acyclic minimal automata recognize the same languages as automata in form of trees, but they take considerably less space. Therefore, minimization is an obligatory step in most applications. In LZ-style compression of automata (Ristov and Korencic, 2015), it is a trie (a letter-tree) that is compressed without going through minimization. However, the compression finds identical sequences of transitions, so it also finds isomorphic trees, which is what minimization does.

In speech recognition, trees are often used (Ortmanns et al., 1997) instead of minimal automata, because lookahead probabilities are computed for individual nodes of the trees. At a given moment in time represented by a node (a state) in a tree-like automaton, one wants to know the probability of the most probable word that is recognized by going through that state. Minimal perfect hashing (Roche, 1995) can deliver a range of numbers for all words going through that state. It is then possible to use them to access probabilities of each word, and find the maximal one. This can be time-consuming, especially close to the root of the tree.

What is needed for lookahead probabilities is a storage for the probability of the most probable word recognized by going through that state, i.e. one probability for each state. This is trivial to implement in a tree — one simply stores the probability in the state. In a minimal DFA, a state can represent many sets of words as it may be equivalent to several nodes in a tree. Therefore in a state of a minimal DFA, one must refer somehow to the corresponding node in the tree, or at least to a place associated with that node. The solution is to provide a dynamically computed mapping from prefixes of words to node numbers in the tree.

## 2 Definitions

A *deterministic finite state automaton* (DFA) is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where $Q$ is a finite set of states, $\Sigma$ is a finite set of symbols called an *alphabet*, $\delta : Q \times \Sigma \to Q$ is a *transition function*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final states*. The *size* of an automaton $|M|$ is the number $|Q|$ of its states. We use *incomplete* automata, so our transition function is *partial*. When $\delta(q, \sigma) \notin Q$, we write $\delta(q, \sigma) = \perp$.

The transition function $\delta$ can be extended in the usual way so that $\delta^* : Q \times \Sigma^* \to Q$:

$$\begin{array}{lcl} \delta^*(q, \varepsilon) & = & q \\ \delta^*(q, u\sigma) & = & \delta(\delta^*(q, u), \sigma) \end{array} \quad (1)$$

The *language* $\mathcal{L}(M)$ of an automaton $M$ is defined as all words that are recognized along paths from the initial state to any of the final states:

$$\mathcal{L}(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\} \quad (2)$$

Among all automata recognizing the same language there is one (up to isomorphism) that has the smallest number of states. It is called the *minimal automaton* $M_{min}$.

The automata described throughout the paper are *acyclic*. This means there is no pair $q \in Q$ and $w \in \Sigma^+$ such that $\delta^*(q, w) = q$. The language of an acyclic DFA is always a finite set of finite-length words.

Among all acyclic DFAs that recognize the same language, there is one in form of a *tree*. All states of such automaton, except for the initial state, have exactly one incoming transition, i.e. they are the target of exactly one transition. The initial state has no incoming transitions. We will use the name nodes for states of a tree, and we will use the name edges for its transitions.

## 3 Counting Tree Nodes

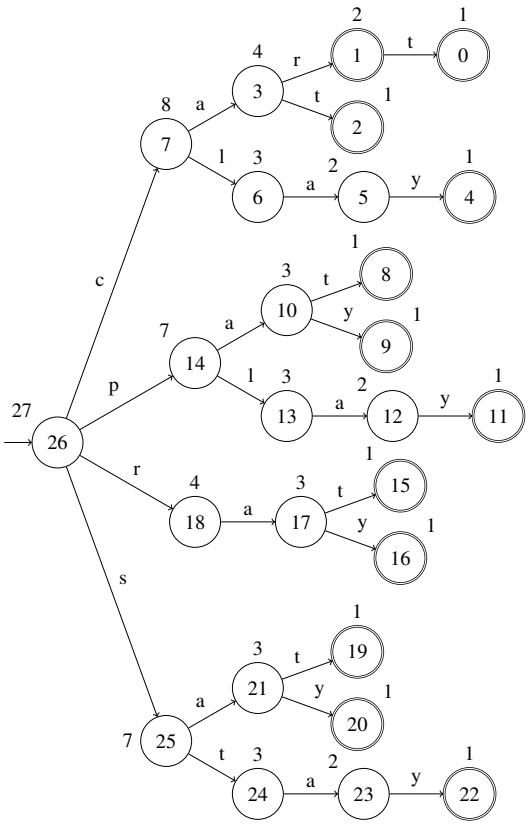


Figure 1: A DFA as a tree recognizing words: *car, cart, cat, clay, pat, pay, play, rat, ray, sat, say*, and *stay*.

A dynamically computed mapping from prefixes of words to node numbers in the tree can be achieved by counting states (nodes) in the tree. Let $c(q)$ be the number of states reachable from state $q$ including state $q$. For a given state $q$, $c(q)$ can be computed as:

$$c(q) = 1 + \sum_{\sigma:\delta(q,\sigma)\neq\perp} c(\delta(q, \sigma)) \qquad (3)$$

If the automaton has a tree shape, and $q$ is the root of a subtree, then $c(q)$ is the number of nodes in that subtree. In a minimal acyclic DFA, the state $q$ can represent several subtrees. However, those trees are isomorphic, and they have exactly the same number of nodes. By induction, equation (3) correctly counts the number of nodes in the corresponding subtree for a minimal acyclic DFA. Actually, it counts that number for any acyclic DFA.

A state in a minimal DFA does not have a node number, as it can correspond to several nodes in the tree. Let $uv = w \in \mathcal{L}(M)$. The prefix $u$ denotes a single node in the tree; it also denotes a state $q = \delta^*(q_0, u)$ corresponding to that node (and perhaps to some other nodes). Nodes are numbered in postorder. Counting nodes from 0, the node number $\mu(u)$ for a prefix $u$ can be calculated as the number of nodes with smaller node numbers. Those are $(c(\delta^*(q_0, u))-1)$ nodes reachable from $\delta^*(q_0, u)$ (excluding $\delta^*(q_0, u)$), and all $\eta(u)$ nodes visited while recognizing all words $w' \prec w = uv$ that are not visited while recognizing the prefix $u$:

$$\mu(u) = c(\delta^*(q_0, u)) - 1 + \eta(u) \qquad (4)$$

where:

$$\eta(u) = \begin{cases} 0 & \text{if } u = \varepsilon \\ \eta(x) + \kappa(\delta^*(q_0, x), a) & \text{if } u = xa, a \in \Sigma \end{cases} \qquad (5)$$

and for a state $q$ and a label $\sigma \in \Sigma : \delta(q, \sigma) \neq \perp$, let $\kappa(q, \sigma)$ be:

$$\kappa(q, \sigma) = \sum_{\sigma' \prec \sigma : \delta(q,\sigma') \neq \perp} c(\delta(q, \sigma')) \qquad (6)$$

The inverse mapping is also easy to calculate. Let $\sigma_{q,n}$ be a label of a transition going out from state $q$ such that $\kappa(q, \sigma) < n \leq \kappa(q, \sigma) + c(\delta(q, \sigma))$. Let $\lambda(q, n)$ be the prefix corresponding to node number $n$ in a subtree with a root $q$.

$$\lambda(q, n) = \begin{cases} \varepsilon & \text{if } n = c(q) - 1 \\ \sigma_{q,n}\lambda(\delta(q, \sigma_{q,n}), z) & \text{if } 0 < n < c(q) \\ \emptyset & \text{if } n > c(q) \end{cases} \qquad (7)$$

where:

$$z = n - \kappa(q, \sigma_{q,n}) \qquad (8)$$

To find a prefix $u$ corresponding to node number $n$, one computes $\lambda(q_0, n)$.

Let us see an example. Figure 1 shows a tree recognizing words *car, cart, cat, clay, pat, pay,*

*play, rat, ray, sat, say*, and *stay*. The number beside node $q$ represents $c(q)$. Let us calculate the node number for a prefix *pl*. As $\delta^*(q_0, pl) = 13$, and $c(13) = 3$, $\mu(pl) = 3 - 1 + \eta(pl)$. We have $\eta(pl) = \eta(p) + \kappa(14, l)$, and $\kappa(14, l) = 3$. As $\eta(p) = \eta(\varepsilon) + \kappa(26, p)$, $\kappa(26, p) = 8$, and $\eta(\varepsilon) = 0$, we have $\mu(pl) = 3 - 1 + ((0 + 8) + 3) = 13$. In the other direction, $\lambda(26, 13) = p\lambda(14, 13 - \kappa(26, p) - 1) = p\lambda(14, 13 - 8 - 1) = p\lambda(14, 4) = pl\lambda(13, 4 - 3 - 1) = pl\lambda(13, 0) = pl\varepsilon = pl$.

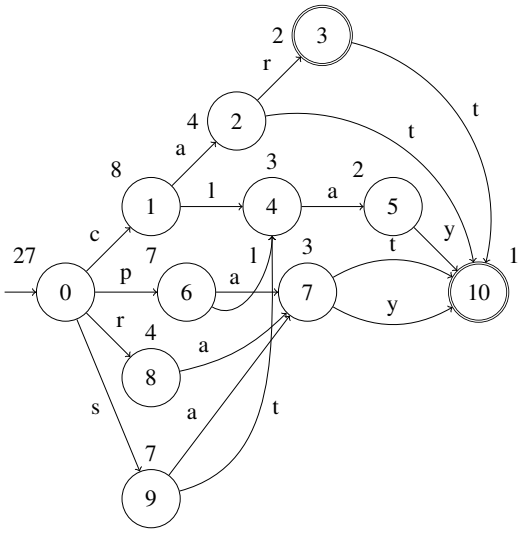


Figure 2: A minimal DFA recognizing the same words as the tree in Figure 1.

Figure 2 shows a DFA corresponding to the tree. Let us repeat the calculations. As $\delta^*(q_0, pl) = 4$, and $c(4) = 3$, $\mu(pl) = 3 - 1 + \eta(pl)$. We have $\eta(pl) = \eta(p) + \kappa(6, l)$, and $\kappa(6, l) = 3$. As $\eta(p) = \eta(\varepsilon) + \kappa(0, p)$, $\kappa(0, p) = 8$, and $\eta(\varepsilon) = 0$, we have $\mu(pl) = 3 - 1 + ((0 + 8) + 3) = 13$. In the other direction, $\lambda(0, 13) = p\lambda(6, 13 - \kappa(0, p) - 1) = p\lambda(6, 13 - 8 - 1) = p\lambda(6, 4) = pl\lambda(4, 4 - 3 - 1) = pl\lambda(4, 0) = pl\varepsilon = pl$. The only thing that changes with regard to the tree are state numbers.

## 4 Edges Can Be Counted Too

Sometimes, it may be more beneficial to use edges of the tree instead of nodes. It is possible to provide equations very similar to those already given for states. However, they are not necessary. Let us look at Figure 1 again. An edge number is the number of the node that the edge leads to. Therefore, we can index the edges using exactly the same formulas as those given for nodes.

## 5 Implementation

To implement the mappings one needs to store the value of $c(q)$ in states $q$, or the value of $\kappa(q, \sigma)$ on the transition going out from states $q$ and labeled with $\sigma$. The second solution is faster, but it requires more memory, as there are more transitions than states.

The first solution delivers smaller memory footprint than the second one as there are more transitions than states in a minimal DFA (unless the DFA is a tree). However, in many automata representations, states are implicit; there addresses are the addresses of their first outgoing transition. The additional number can be stored in front of the first transition, but that would impede storing one state inside another (a common compression technique). *Superimposed coding* (Liang, 1983) based on a *sparse matrix representation* technique (Tarjan and Yao, 1979; Dencker et al., 1984; Fredman et al., 1984) enforces fixed-length transitions belonging to overlapping states, which excludes storing something else before the first transition of a state. It is possible to provide an additional vector to store the desired values, but there is a problem with addressing them, as state numbers are not used for addressing a states in the DFA (transition numbers are used instead).

The mapping from prefixes (identifying nodes in a tree) to numbers is implemented as function `Pref2N` given below. In order to get the prefix number for prefix $w$, one calls `Pref2N`$(M, w, q_0)$.

**Function** Pref2N$(M, w = ax, q)$

```
1  if w = ε then
2  |  return c(q) − 1
3  else
4  |  s ← 0
5  |  for σ ∈ Σ : σ < a do
6  |  |  s ← s + c(δ(q, σ))
7  |  if δ(q, a) = ⊥ then
8  |  |  raise an exception
9  |  else
10 |  |  return s+ Pref2N(M, x, δ(q, a))
```

Line 2 computes $c(\delta^*(q_0, u)) - 1$. The `for` loop in lines 5 and 6 computes $\kappa(\delta(q_0, x), a)$. Function `Pref2N` is called $|w|$ times for each symbol $a$ in the prefix. The `for` loop runs at most $|\Sigma|$ times in each invocation of function `Pref2N`. This gives

us overall time complexity of $\mathcal{O}(|w||\Sigma|)$ times, i.e. linear with regard of the prefix. With the values of $\kappa(q, a)$ stored on transitions $\delta(q, a)$, we replace line 4 with $s \leftarrow \kappa(q, a)$ and remove lines 5 and 6, making the function a bit faster.

The inverse mapping is implemented as function `N2Pref`. To get a prefix number $n$, one should call `N2Pref`$(M, n, q_0)$.

**Function** N2Pref$(M, n, q)$

```
1  if n = c(q) − 1 then
2      return ε
3  else
4      s ← 0
5      for σ ∈ Σ do
6          if δ(q, σ) ≠ ⊥ then
7              if s + c(δ(q, s)) ≥ n then
8                  return σ·N2Pref(n-s)
9              else
10                 s ← s + c(δ(q, s))
11     raise an exception
```

Function `N2Pref` is called as many times as there are symbols in the computed prefix. The `for` loop runs at most $|\Sigma|$ times. As other operations take constant time, the time complexity of the function is $\mathcal{O}(|\Sigma||w_{max}|)$, where $w_{max}$ is the longest word in the language of the DFA. Note that in the superimposed coding (sparse matrix) representation, the `for` loop must run for every symbol of the alphabet. In a list representation, the loop runs only on (symbols on) the outgoing transitions, which can be significantly faster. Having pre-computed values of $\kappa$ on transitions saves us computing the sum in line 10; the $\kappa$ value is used directly in line 7.

## 6 Conclusions

We have shown methods to store indexes of nodes or edges of a tree in a minimal (or pseudo-minimal) automaton. Trees are sometimes used in place of minimal DFAs because information associated with their nodes or edges is needed. The presented methods render such trees obsolete. The minimal DFA can provide indexes in vectors that would store tree-related data. As a minimal DFA is usually much smaller than a tree, the methods can save a lot of memory. We have also developed similar but more complicated methods for indexing subtrees in a minimal, deterministic, bottom-up tree automaton.

## 7 Acknowledgments

We wish to thank Marcin Kuropatwiński from speech recognition company Voice Lab for discussions that led to development of the algorithms presented in this paper.

## References

[Daciuk et al.2010] Jan Daciuk, Jakub Piskorski, and Strahil Ristov. 2010. Natural language dictionaries implemented as finite automata. In Carlos Martín-Vide, editor, *Scientific Applications of Language Methods*, pages 133–204. Imperial College Press.

[Dencker et al.1984] Peter Dencker, Karl Dürre, and Johannes Heuft. 1984. Optimization of parser tables for portable compilers. *ACM Transactions on Programming Languages and Systems*, 6(4):546–572, October.

[Fredman et al.1984] Michael L. Fredman, János Komlós, and Endre Szemerédi. 1984. Storing a sparse table with 0(1) worst case access time. *Journal of the ACM*, 31(3):538–544, July.

[Liang1983] Franklin Mark Liang. 1983. *Word Hyphen-a-tion by Com-put-er*. Ph.D. thesis, Stanford University.

[Ortmanns et al.1997] S. Ortmanns, H. Ney, N. Coenen, and A. Eiden. 1997. Look-ahead techniques for fast beam search. In *proceedings of IEEE International Conference on Acoutic, Speech and Signal Processing*, volume 3, pages 1783–1786, Munich, Germany, April.

[Ristov and Korencic2015] Strahil Ristov and Damir Korencic. 2015. Fast construction of space-optimized recursive automaton. *Software: Practice and Experience*, (45).

[Roche1995] Emmanuel Roche. 1995. Finite-state tools for language processing. In *ACL'95*. Association for Computational Linguistics. Tutorial.

[Tarjan and Yao1979] Robert Endre Tarjan and Andrew Chi-Chih Yao. 1979. Storing a sparse table. *Communications of the ACM ACM*, 22(11):606–611, November.