

Generating large and optimized FSA for morphological analysis

Zoltán Alexin
University of Szeged
Department of Software Engineering
e-mail: alexin@inf.u-szeged.hu

FSA (Finite State Automaton)

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

where

Q – is a finite set of states, $Q = \{q_0, q_1, \dots, q_n\}$

Σ – is the finite input alphabet,

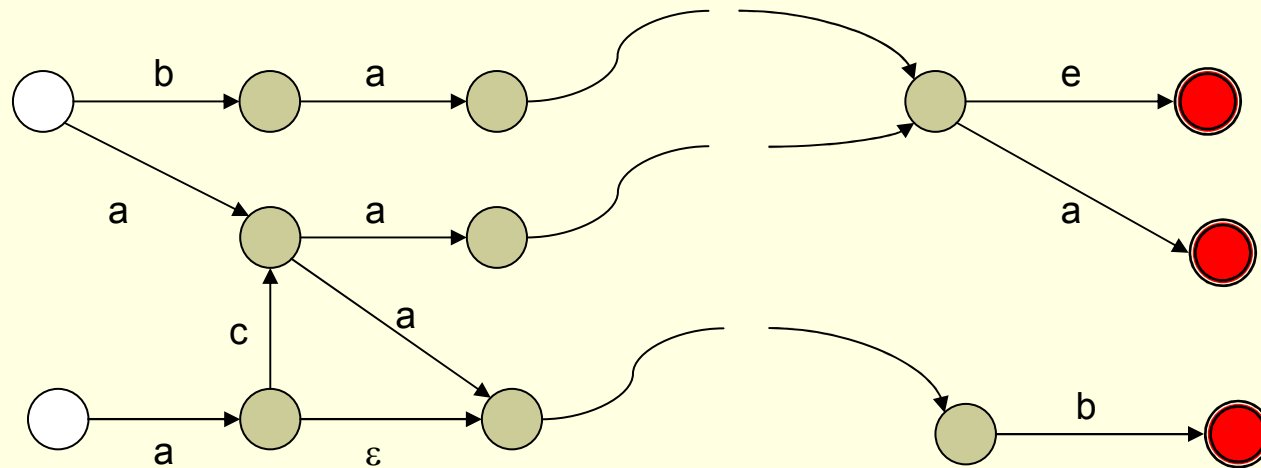
δ – is the transition function $\delta : Q \times (\Sigma \cup \varepsilon) \rightarrow \mathcal{P}(Q)$

q_0 – is the initial state, $q_0 \in Q$

F – is the set of final states, $F \subset Q$

The size of the automaton is $|Q|$

STN (State Transition Network)



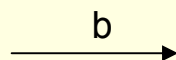
initial state



intermediate state

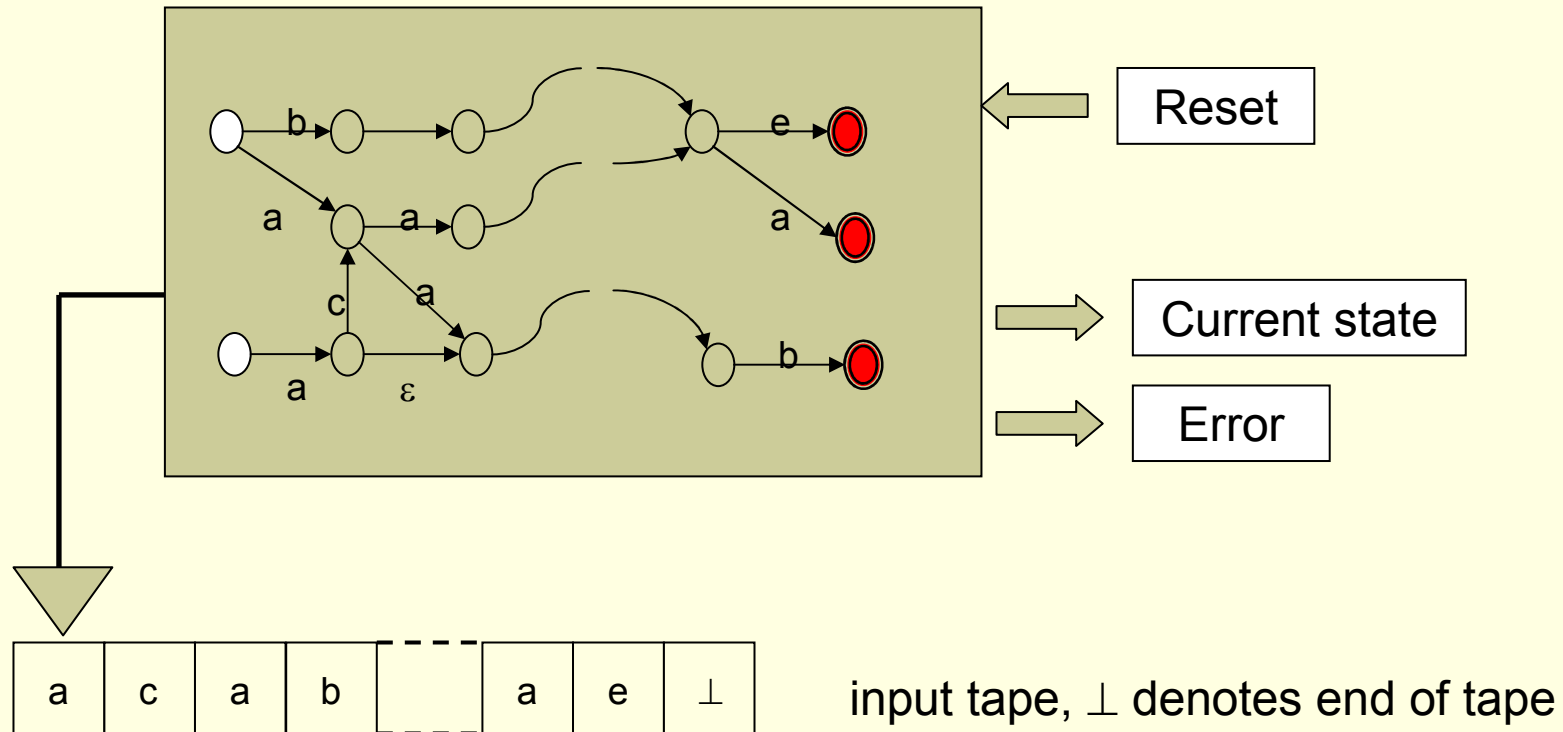


final state



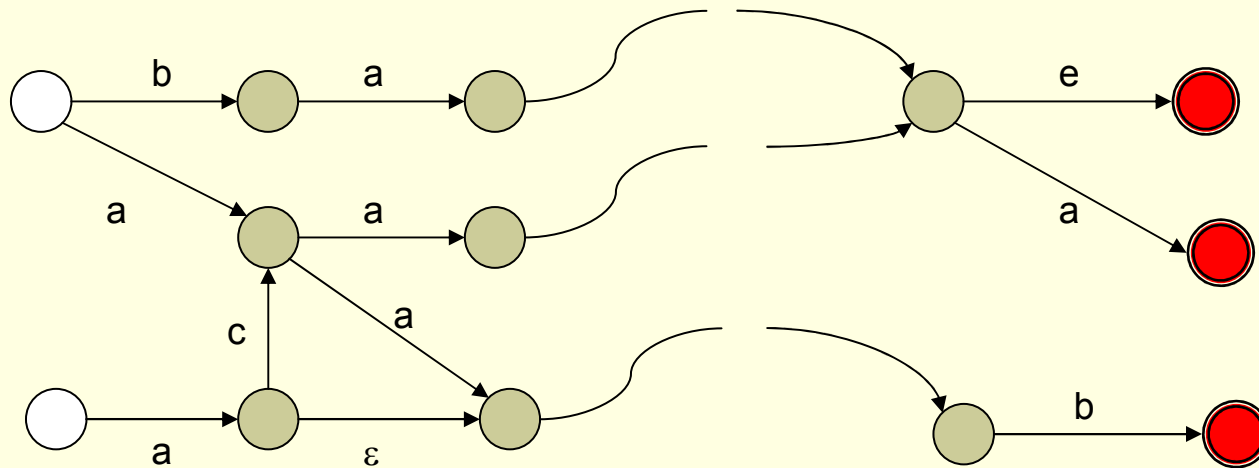
Transition after reading input symbol „b”

How an automaton is working?



The automaton has a register where it stores the current state. On pressing reset button, the automaton goes into the initial state. Upon reading a character from the input tape the automaton goes into the next state. When there is no transition corresponding to the input character, the automaton goes into an error state.

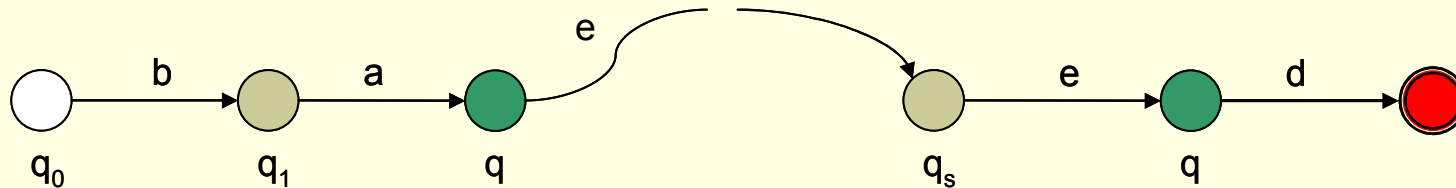
Nondeterministic behavior



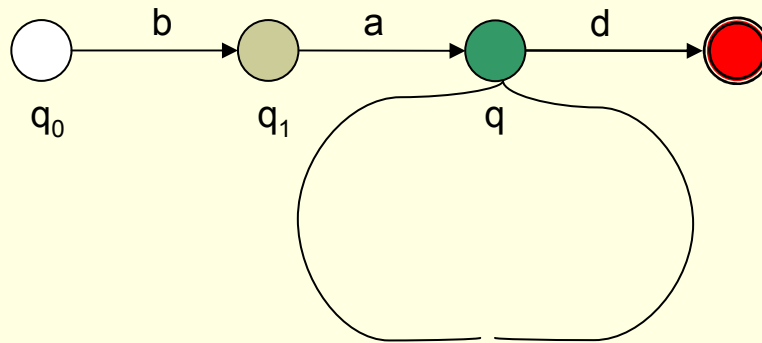
- When there are more than one initial state
- There is an ε -transition in the automaton
- More than one outgoing transitions with the same label

Definition: Deterministic FSA – that has only one initial state, has no ε -transition and for each state, there is at most one transition with the same label.

Bar-Hillel's first pumping lemma



■ Long transition path means a cycle in the automaton.



Each automaton has a constant L .

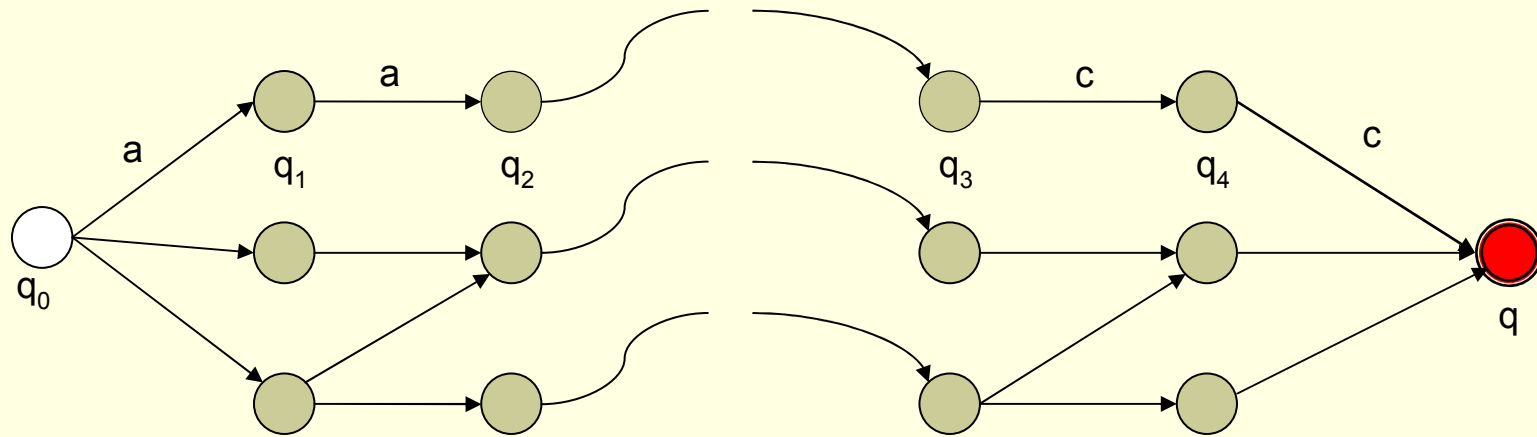
If the automaton recognizes a word w , that is longer than this L ,

then there are α, β, v , $w = \alpha v \beta$, so that for each k , the automaton recognizes all $w_k = \alpha v^k \beta$ words.

Morphological analysis by FSA

- The automaton recognizes finite set of finite words.
- Therefore, according to the Bar-Hillel's lemma it must not have cycle in the transition graph.
- The automaton's graph is a directed acyclic graph (DAG) i.e. a tree.
- For Hungarian we have a definition file, that has 100 000 base words and from it 135 million word forms can be generated.
- From 5000 base words (6 million word forms) an optimal automaton have been generated with 1 million states. From that we may purpose that the automaton for the 135 million word forms may have 20-30 million states.

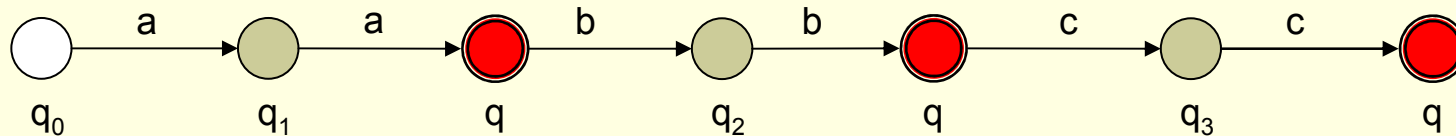
Why do we need more than one final state



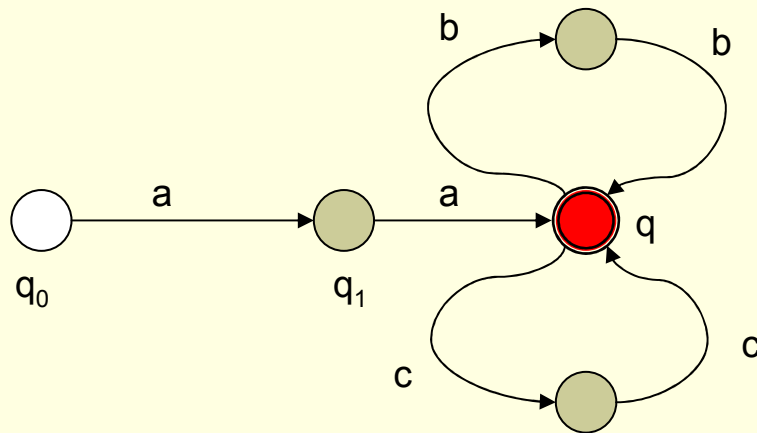
Why do we need more final state? Why one final state is not enough?

Why do we need more than one final state

The automaton can be defined by the language (set of words) it should recognize. Let $L(A) = \{ aa, aabb, aabbcc \}$.

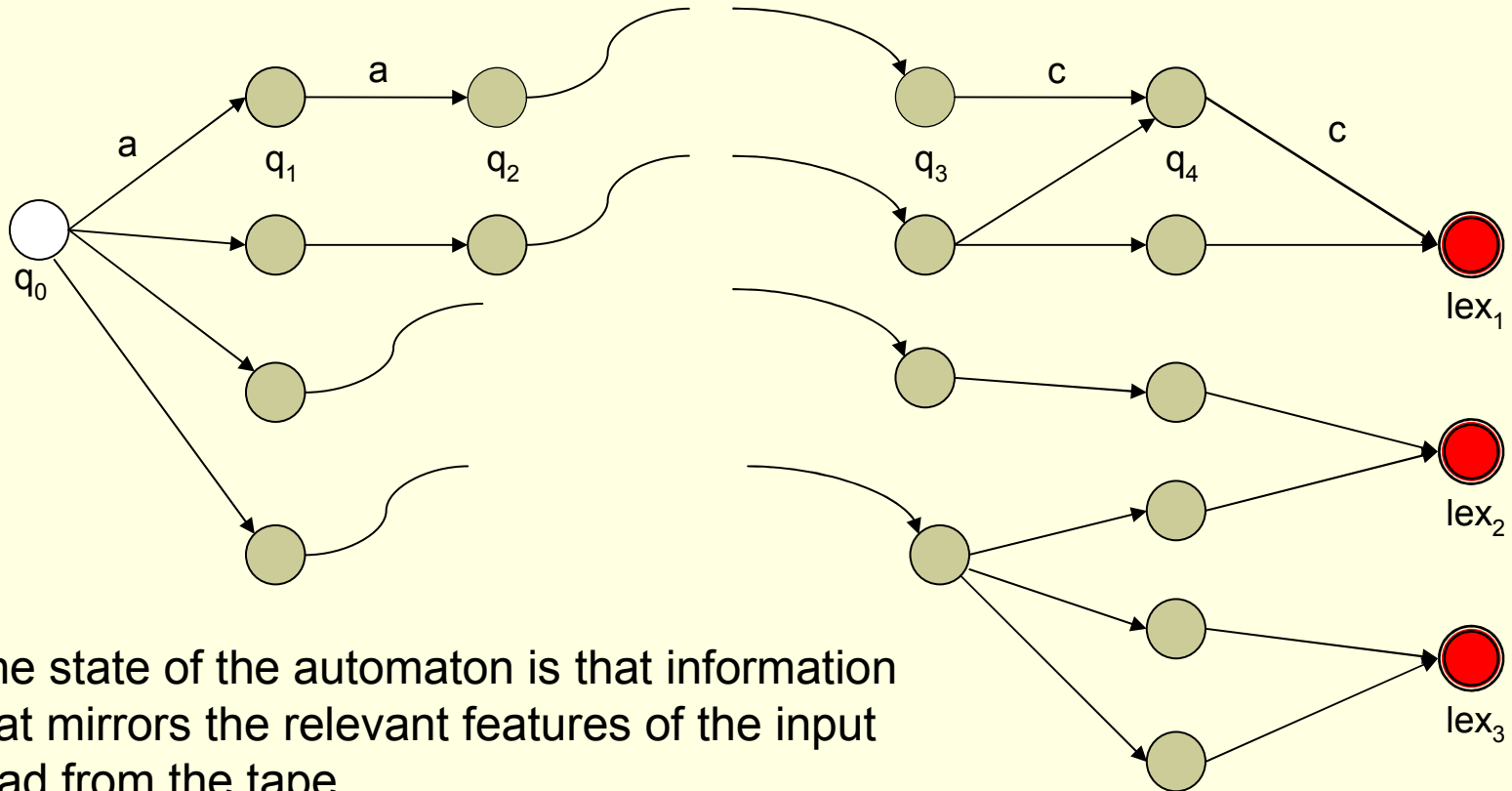


We should distinguish the three final state, because otherwise the automaton would be something very different.



This automaton accepts not only aa , $aabb$ and $aabbcc$, but $aacccc$, $aabbbb$, etc.

Morphological analysis by FSA



The state of the automaton is that information that mirrors the relevant features of the input read from the tape.
Therefore final states of the automaton should be the lexical encodings of input words.

How to encode lemmas?

Among 135 million Hungarian word forms there were 3 364 626 different lexical encodings.

When we talk about morphological analysis, we should pay attention to lemmas as well. Lemma is also a lexical feature of the input word.

If we simply add the lemmas to the lexical information, then we multiply the number of final states. That may result in that each word form may have its own final state. That will block any further optimization.

In 2007 Slim Mesfar suggested to store an edit command (by which one can obtain lemma from the base word) instead of the lemma itself. This idea later proved to be very successful in the case of Hungarian, because usually word having the same lexical encoding have same edit command (same suffixes) by which one can derive the base word.

An edit command may contain deletions (from the beginning and from the end) and insertions of characters.

How to encode lemmas?

Some edit commands:

e

ik

j

ja

je

k

m

t

<leg>

<leg>

<leg>

 - deletion of single arbitrary character from end

<leg> - deletion of „leg” prefix from the beginning

Other characters mean insertions.

How to encode lemmas?

- The idea of edit commands proved to be very successful for Hungarian
- 1920 different edit commands were needed for 135 million word form
- It did not increase substantially the number of necessary final states

Deterministic or Nondeterministic FSA?

- An FSA can be defined by enlisting the words it should accept (together the required final state, i.e. the lexical encoding and the edit command for base words).
- Morphological analysis is ambiguous, because of homonymy and polysemy.
- From this definition basically a nondeterministic automaton can be generated.
- Implementing nondeterministic automaton is more expensive, needs a mean for storing all possible nondeterministic state that the automaton may be in.
- Nondeterministic automata are sufficiently harder to optimize.
- According to the well known theorem, a nondeterministic automaton can always be transformed to a deterministic one.

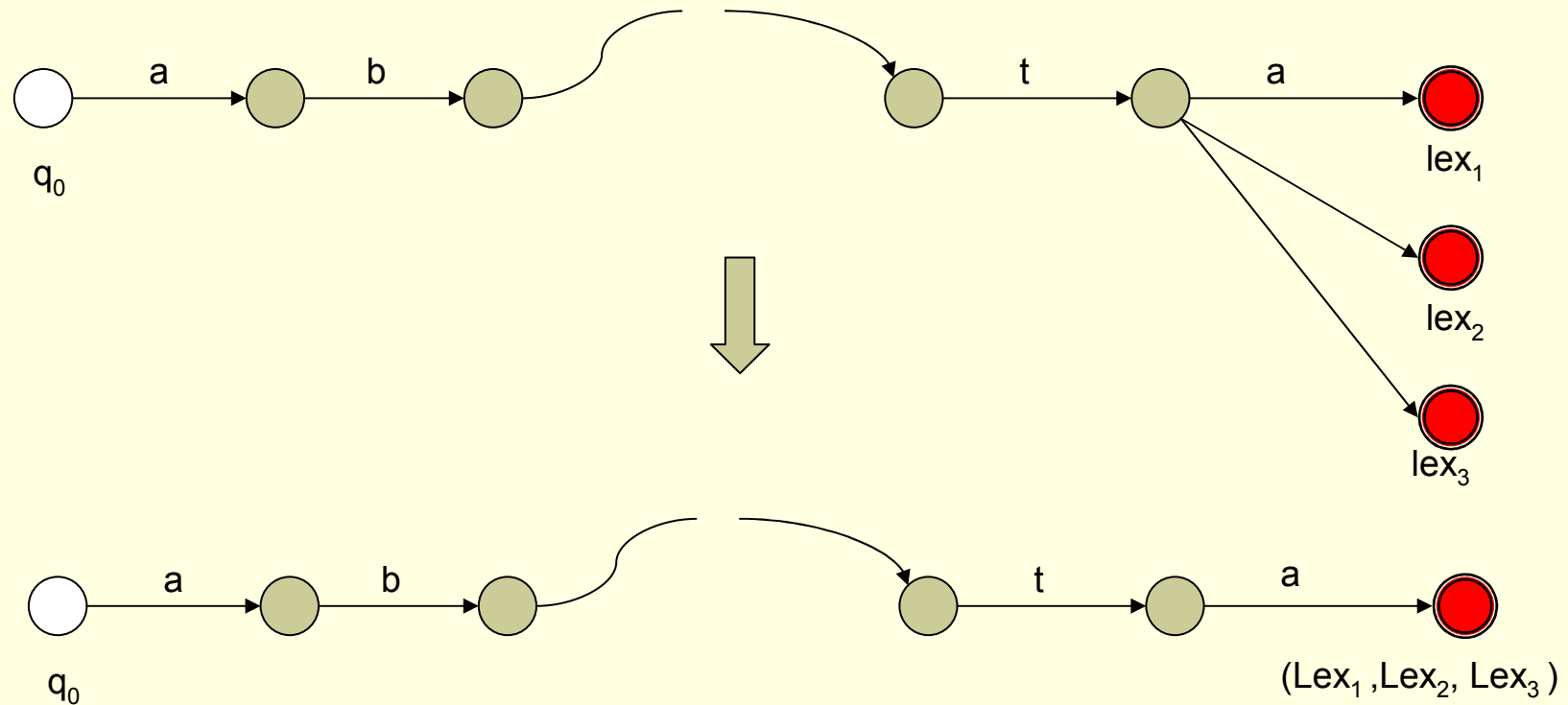
Let's do it!

The power-set method

- The nondeterministic behavior of our FSA is caused by final states.
- Several different final states may belong to the same input word.
- If this were not be the case, the automaton would be deterministic.

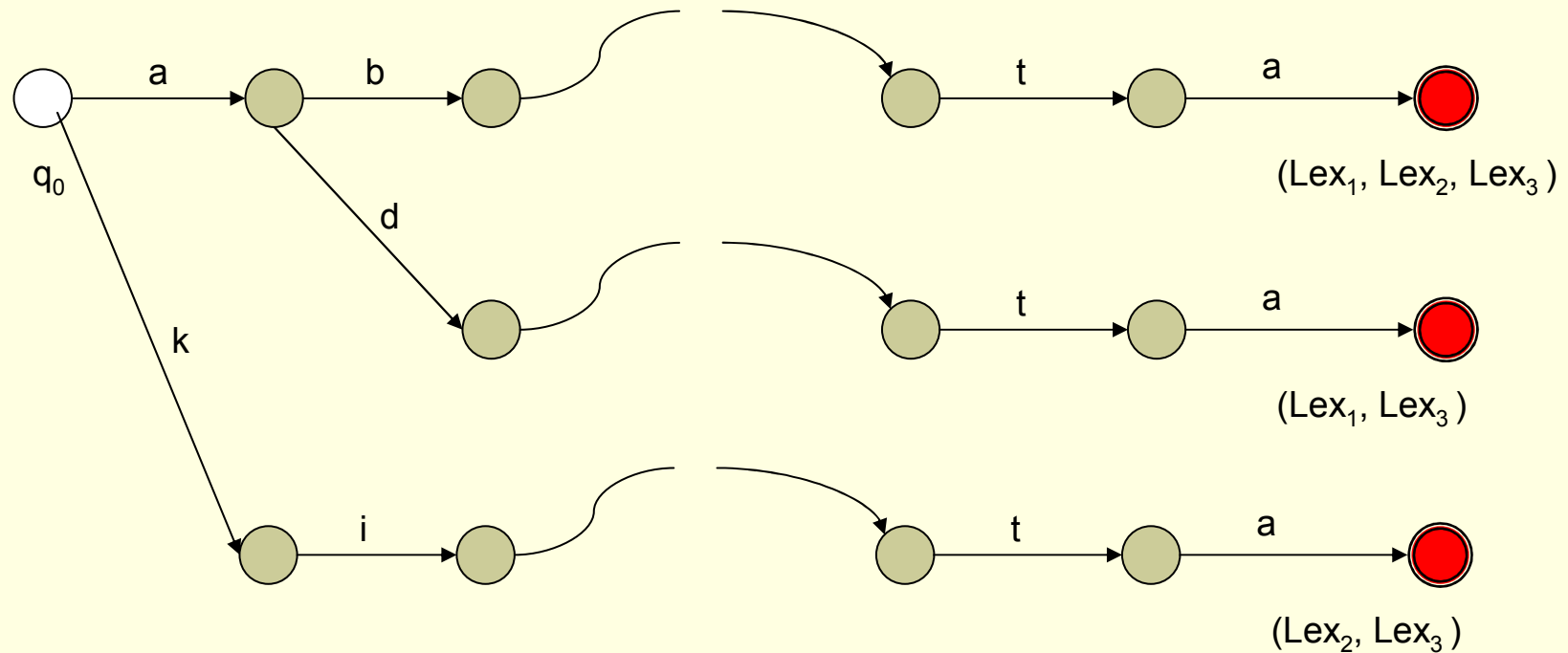
- The standard method for making deterministic automaton is replacing the set of states it may be in by a single element of the power-set.

The power-set method



After doing this transformation all word forms will have one and only one final state. Lex_1 , Lex_2 and Lex_3 include edit command for the lemma as well.

The power-set method

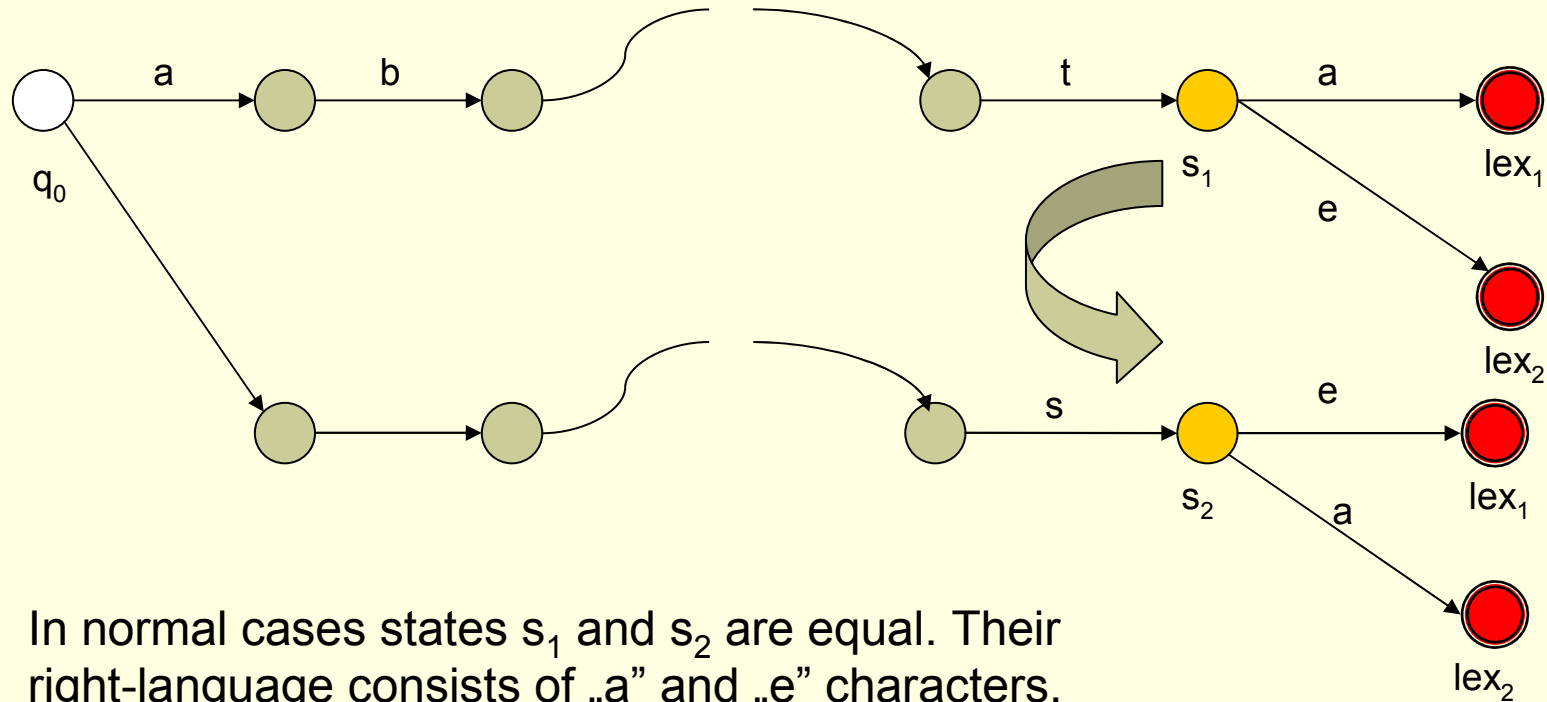


The (Lex_1, Lex_2, Lex_3) , (Lex_1, Lex_3) and (Lex_2, Lex_3) final states all will be different. This may slightly increase the number of possible final states. Let's take into account the impact of edit commands as well.

How to optimize the automaton?

- The basic idea of optimization is uniting equal states.
- According to the literature (e.g. Daciuk's works), one state of the automaton is equal to another state if and only if their right language is the same.
- Right-language (RL) is the set of all possible suffixes that can be reached from the state leading to some final state. ($w \in \text{RL}(s)$, iff $\delta^*(s, w) \in F$)
- **In our case final states encode lexical encodings, therefore equivalence of states should mean that elements of the right-language should pairwise lead to the same final state!**

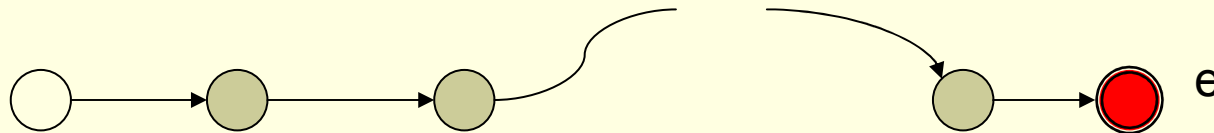
Equivalent states



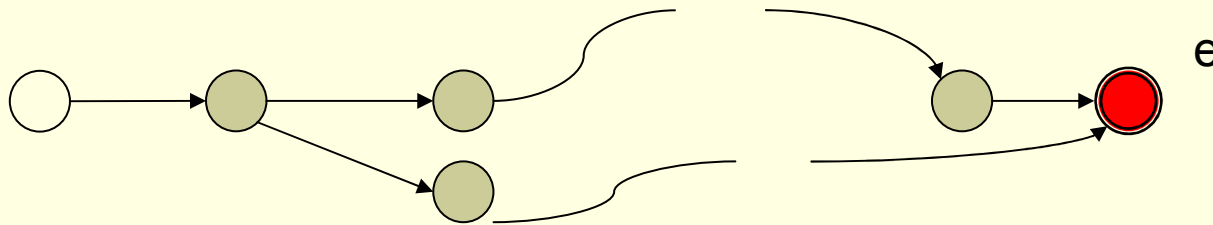
In normal cases states s_1 and s_2 are equal. Their right-language consists of „a” and „e” characters. But in our case, when final states convey important information, we should distinguish these states.

Single and multiple final states

- Def. 1.: single final state: there is one and only one path leading to this final state

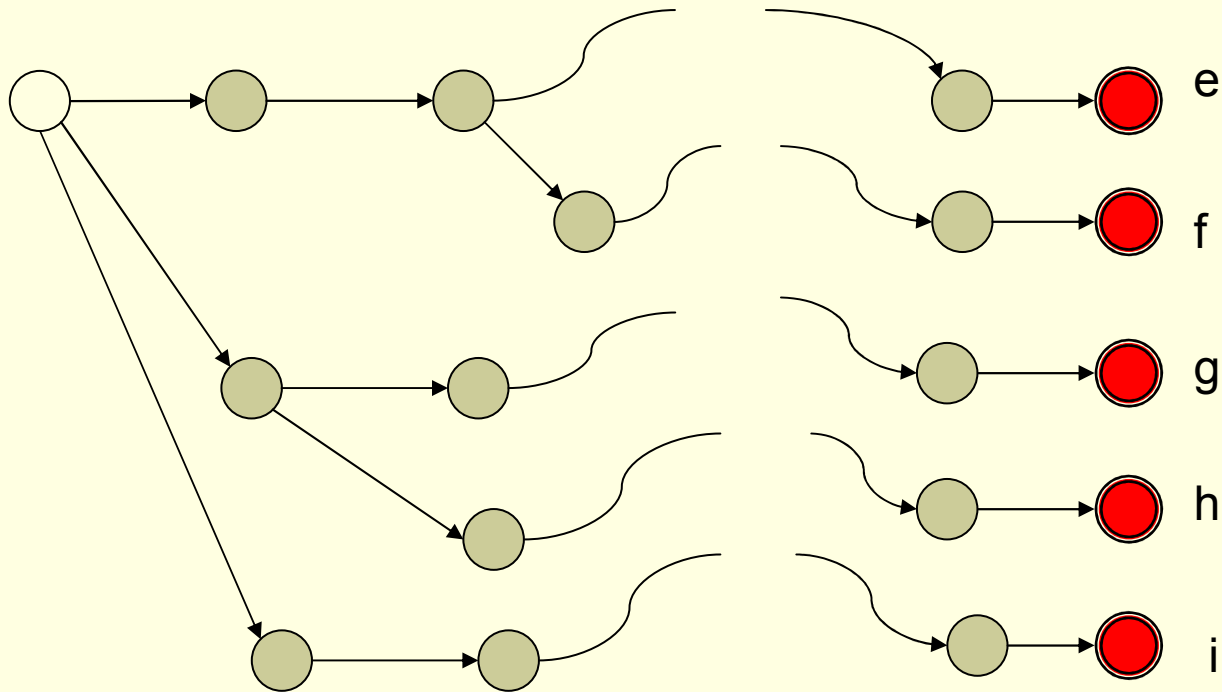


- Def. 2.: multiple final state: there are more than one paths leading to this final state



Spanning tree of single final states

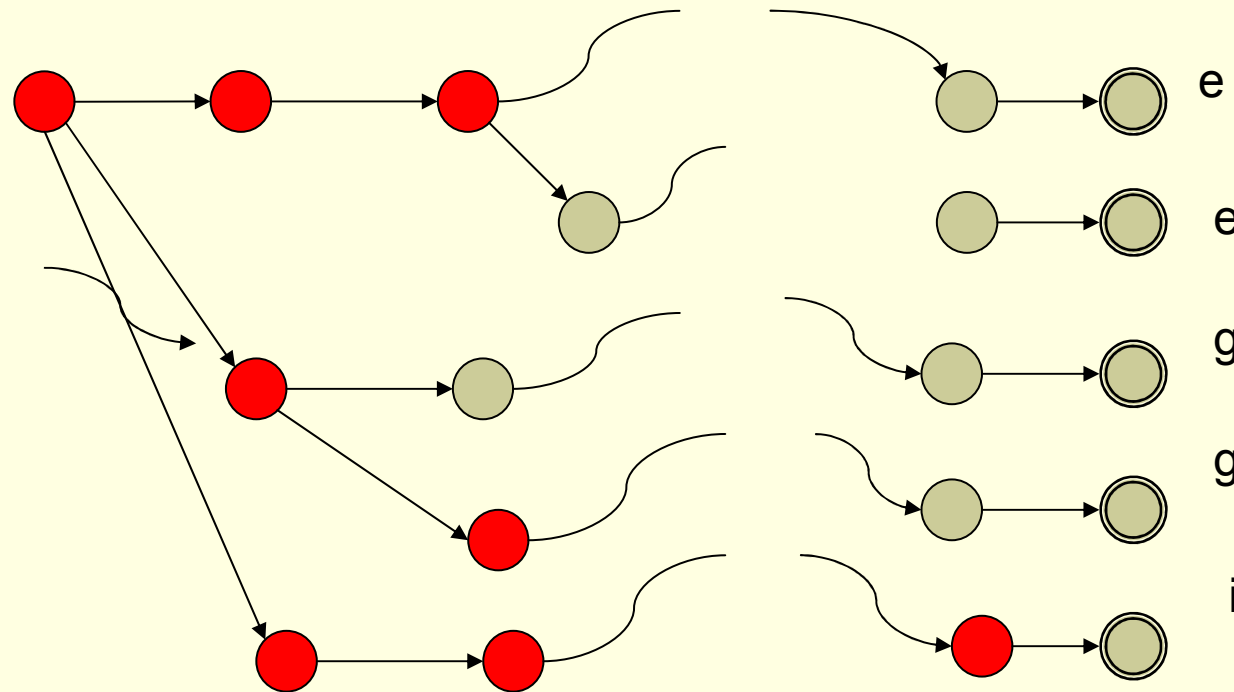
- Def. 3.: Spanning tree of single final states is a sub-automaton that contains all single final states and no multiple final states.



Spanning tree of single final states

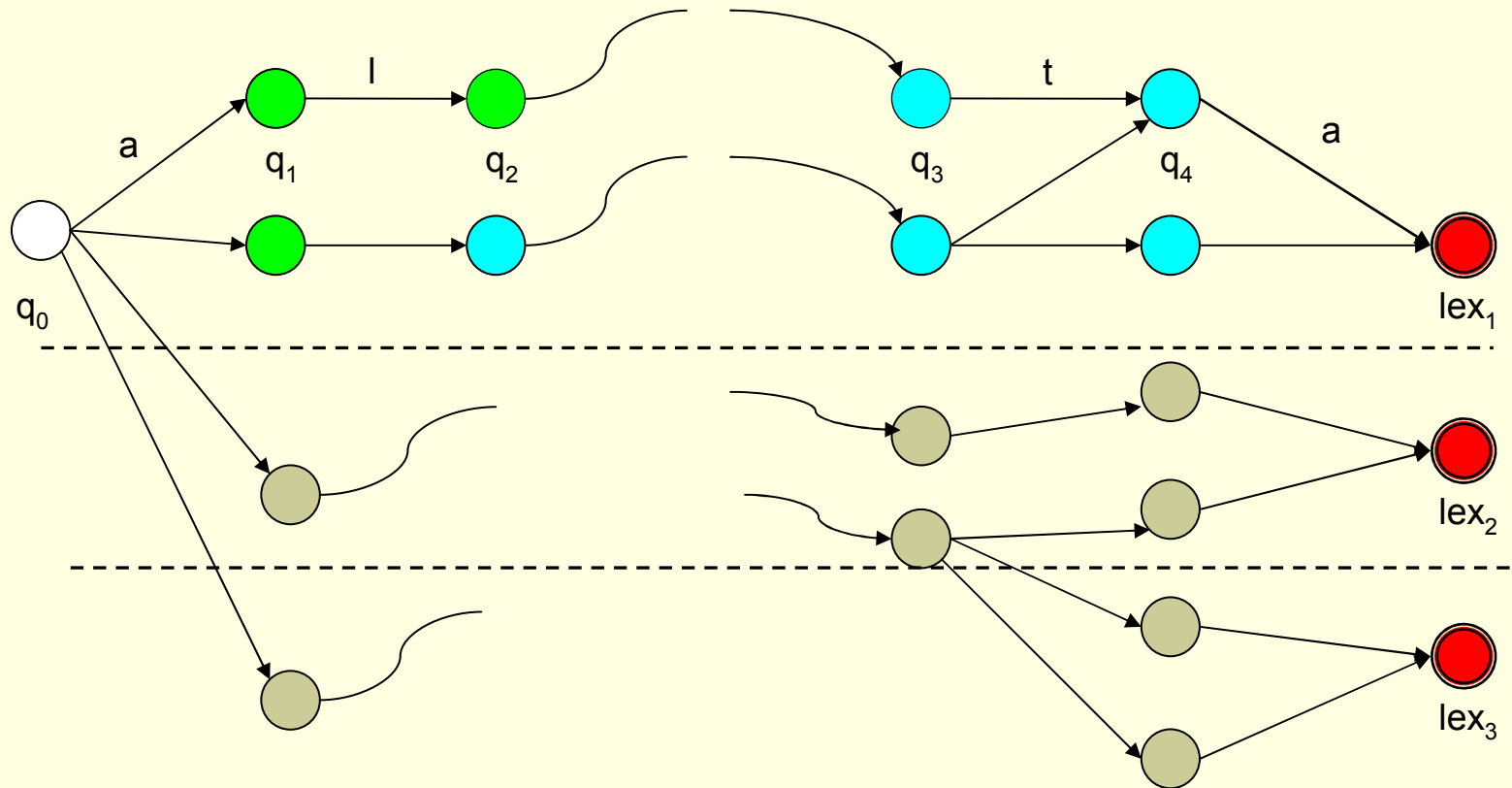
- The spanning tree:
 - is a tree, i. e. there is no loop in it
 - when constructed: all paths are different
 - states having numbers $1, 2, \dots, K$ are belonging to the spanning tree
 - spanning trees have no states that can be equivalent, i.e. may have the same right-language

The generated FSA



- Red states are belonging to the spanning tree, having number less than K .
- Grey states may be target of optimization
- Only states belonging to the same transaction group may have the same right language, i. e. optimization may go transaction-group wise.

Optimizing FSA



State belonging to the spanning tree



Equivalent states can be searched here

FSA Generation with commands

- AMD Athlon, Windows XP, 4GB memory, MS Visual C# .NET express edition.

File	lines	commands	endstates	States	optimized
noun-sample-1K-flx.dic	2011295	227	306699	3761066	591823
noun-sample-2K-flx.dic	4160024	389	456176	7417663	877070
verb-sample-1K-flx.dic	1196674	263	284992	2163756	540335
verb-sample-2K-flx.dic	2389490	352	376319	4158120	713899
verb-sample-5K-flx.dic	5981798	387	556393	9403218	1056477

Summary

- When optimizing, not all states should be considered.
- States of the spanning-tree may be excluded.
- The optimization can be done groupwise, considering states elements of paths leading to the same final state.
- The whole optimization process can be broke down to much smaller tasks, by dividing the optimization to 3-4 million much smaller optimizations.

FSA Generation

- AMD Athlon, Windows XP, 4GB memory, MS Visual C# .NET express edition.

File	lines	commands	endstates	states	optimized
noun-sample-1K-flx.dic	2011295	1	275077	3761066	531819
noun-sample-2K-flx.dic	4160024	1	407993	7417663	786026
noun-sample-5K-flx.dic	10253355	1	?	?	?
verb-sample-1K-flx.dic	1196674	1	259246	2163756	492581
verb-sample-2K-flx.dic	2389490	1	342803	4158120	651744
verb-sample-5K-flx.dic	5981798	1	496268	9403218	944602

Future work

- Move to x64 architecture, and 4GB
- Get serialization source from ROTOR (Microsoft)
- Move to SUN C/C++, and come back to C#
- NooJ Mono transformation