Some algorithms

- Colin de la Higuera
- University of Nantes

UNIVERSITÉ DE NANTES

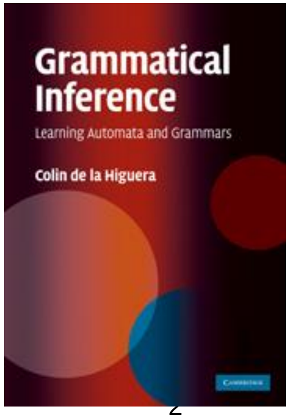
Statistical and symbolic language modeling

Acknowledgements

- Laurent Miclet, Jose Oncina and Tim Oates for previous versions of these slides.
- Rafael Carrasco, Paco Casacuberta, Rémi Eyraud, Philippe Ezequel, Henning Fernau, Thierry Murgue, Franck Thollard, Enrique Vidal, Frédéric Tantini,...
- List is necessarily incomplete. Excuses to those that have been forgotten.

http://pagesperso.lina.univ-nantes.fr/~cdlh/slides/

Chapter 12 (and part of 14)



Outline

1. K-Testable language learning
2. The rules of the game
3. Basic elements for learning DFA
4. RPNI
5. Complexity discussion
6. Heuristics
7. Open questions and conclusion

P. García and E. Vidal. Inference of K-testable languages in the strict sense and applications to syntactic pattern recognition. *Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990

P. García, E. Vidal, and J. Oncina. Learning locally testable languages in the strict sense. In *Workshop on Algorithmic Learning Theory (*ALT *90)*, pages 325–338, 1990

1 Learning k-testable languages

Definition

Let $k \geq 0$, a k-testable language in the strict sense (k-TSS) is a 5-tuple $Z_k=(\Sigma, I, F, T, C)$ with:

- Σ a finite alphabet
- $I, F \subseteq \Sigma^{k-1}$ (allowed prefixes of length k-1 and suffixes of length k-1)
- $T \subseteq \Sigma^k$ (allowed segments)
- $C \subseteq \Sigma^{<k}$ contains all strings of length less than k
- Note that $I \cap F = C \cap \Sigma^{k-1}$

- The k-testable language for Z_k is

  $\mathbf{L}(Z_k)=I\Sigma^* \cap \Sigma^* F - \Sigma^*(\Sigma^k-T)\Sigma^* \cup C$

- Strings (of length at least k) have to use a good prefix and a good suffix of length k-1, and all sub-strings have to belong to T. Strings of length less than k should be in C
- Or: Σ^k-T defines the prohibited segments
- Key idea: use a window of size k

Window languages

- We use a window to decide if computation is OK
- At any moment, the content of the window has to be permitted

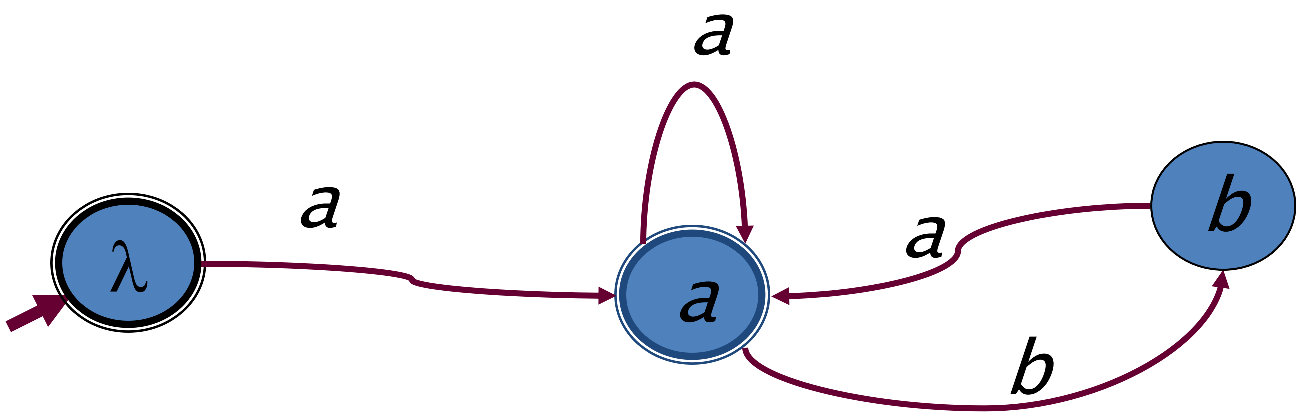
An example (2-testable)

$I = \{a\}$

$F = \{a\}$

$T = \{aa, \ ab, \ ba\}$

$C = \{\lambda, a\}$



Window language

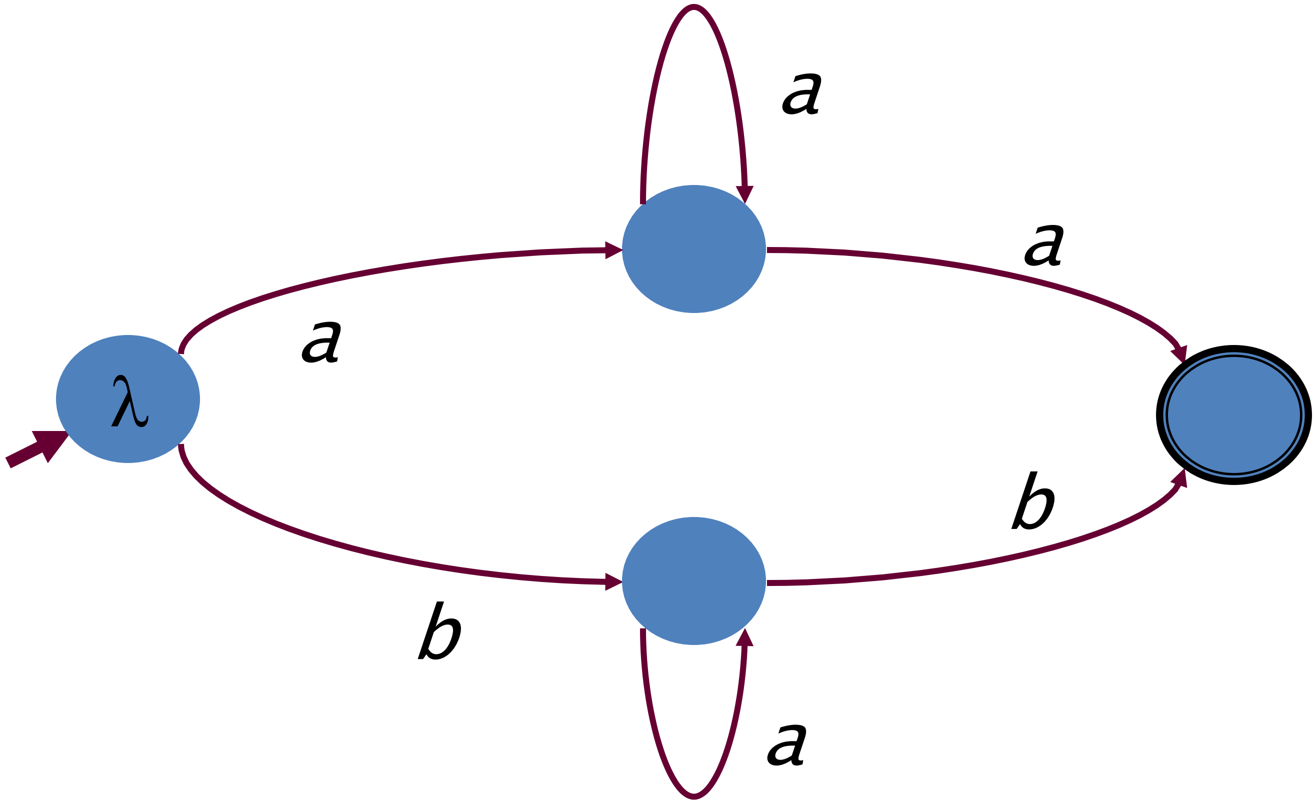
By sliding a window of size 2 over a string we can parse

- *ababaaabababababaaaa* OK
- *aaabbaaaababab* not OK

The hierarchy of k-TSS languages

- k-$TSS(\Sigma)=\{L\subseteq\Sigma^*: L$ is k-$TSS\}$
- All finite languages are in k-$TSS(\Sigma)$ if k is large enough!
- k-$TSS(\Sigma) \subset [k+1]$-$TSS(\Sigma)$
- $(ba^k)^* \in [k+1]$-$TSS(\Sigma)$
- $(ba^k)^* \notin k$-$TSS(\Sigma)$
- With a window of length only k, if we accept window a^k, then string a^{k+22} is in L

A language that is not *k*-testable



K-TSS inference

Given a sample S, $\mathbf{L}(\mathbf{a}_{k\text{-}TSS}(S)) = Z_k$ where
$Z_k = (\Sigma(S), I(S), F(S), T(S), C(S))$ and

- $\Sigma(S)$ is the alphabet used in S
- $C(S) = \Sigma(S)^{<k} \cap S$
- $I(S) = \Sigma(S)^{k-1} \cap \mathrm{Pref}(S)$
- $F(S) = \Sigma(S)^{k-1} \cap \mathrm{Suff}(S)$
- $T(S) = \Sigma(S)^{k} \cap \{v : uvw \in S\}$

Example

- $S=\{a, aa, abba, abbbba\}$
- Let $k=3$
  - $\Sigma(S)=\{a, b\}$
  - $I(S)= \{aa, ab\}$
  - $F(S)= \{aa, ba\}$
  - $C(S)= \{a, aa\}$
  - $T(S)=\{abb, bbb, bba\}$

- $\mathbf{L}(\boldsymbol{a}_{3\text{-}TSS}(S))= ab^*a+a$

Building the corresponding automaton

- Each string in $I \cup C$ and PREF($I \cup C$) is a state
- Each substring of length k-1 of strings in T is a state
- λ is the initial state
- Add a transition labeled b from u to ub for each state ub
- Add a transition labeled b from au to ub for each aub in T
- Each state/substring that is in F is a final state
- Each state/substring that is in C is a final state

Running the algorithm

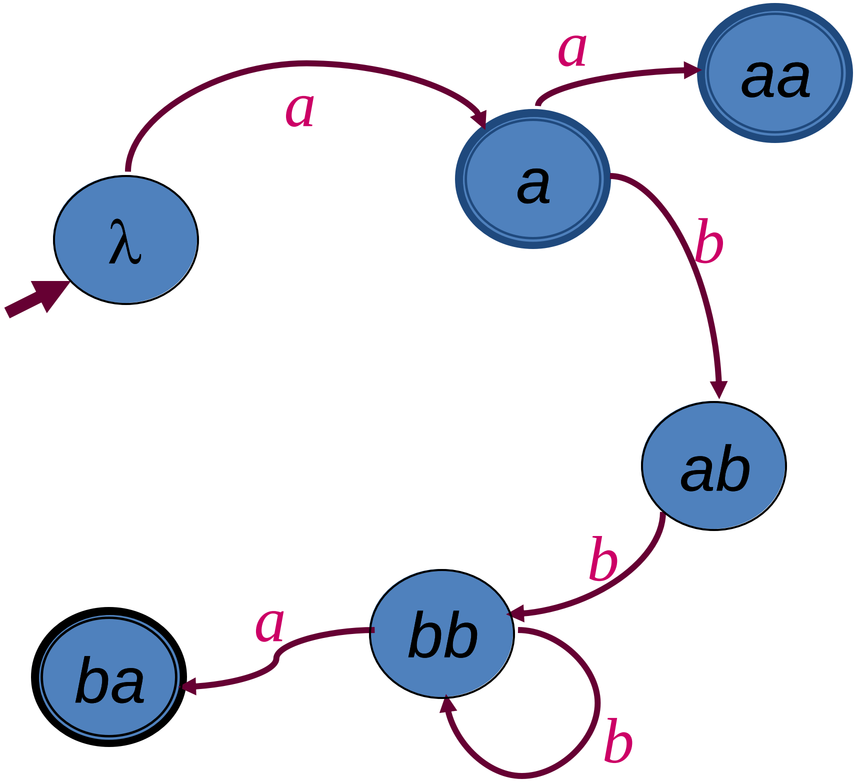
S={*a*, *aa*, *abba*, *abbbba*}

I={*aa*, *ab*}

F={*aa*, *ba*}

T={*abb*, *bbb*, *bba*}

C={*a*, *aa*}



Properties (1)

- $S \subseteq \mathbf{L}(\mathbf{a}_{k\text{-}TSS}(S))$

- $\mathbf{L}(\mathbf{a}_{k\text{-}TSS}(S))$ is the smallest *k-TSS* language that contains S
  - *If there is a smaller one, some prefix, suffix or substring has to be absent*

Properties (2)

- $\mathbf{a}_{k\text{-}TSS}$ identifies any *k-TSS* language in the limit from polynomial data
  - Once all the prefixes, suffixes and substrings have been seen, the correct automaton is returned
- If $Y \subseteq S$, $\mathbf{L}(\mathbf{a}_{k\text{-}TSS}(Y)) \subseteq \mathbf{L}(\mathbf{a}_{k\text{-}TSS}(S))$

Properties (3)

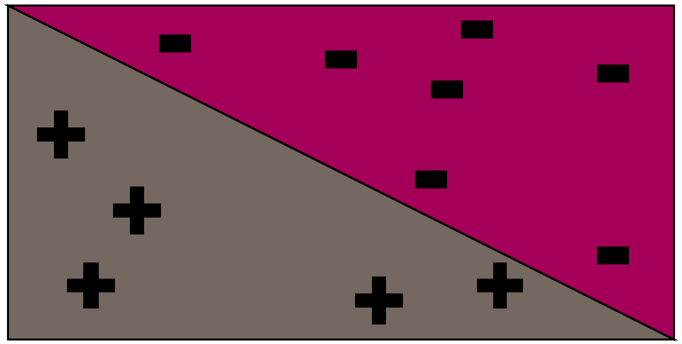
- $\mathbf{L}(\mathfrak{a}_{k+1\text{-}TSS}(S)) \subseteq \mathbf{L}(\mathfrak{a}_{k\text{-}TSS}(S))$
  In I_{k+1} (resp. F_{k+1} and T_{k+1}) there are less allowed prefixes (resp. suffixes or substrings) than in I_k (resp. F_k and T_k)
- $\forall k > \max_{x \in S} |x|$, $\mathbf{L}(\mathfrak{a}_{k\text{-}TSS}(S)) = S$
  - Because for a large k, $T_k(S) = \emptyset$

2. The rules of the game

Motivation

- We are given a set of strings S_+ and a set of strings S_-
- Goal is to build a classifier
- This is a traditional (or typical) machine learning question
- How should we solve it?

Σ^*

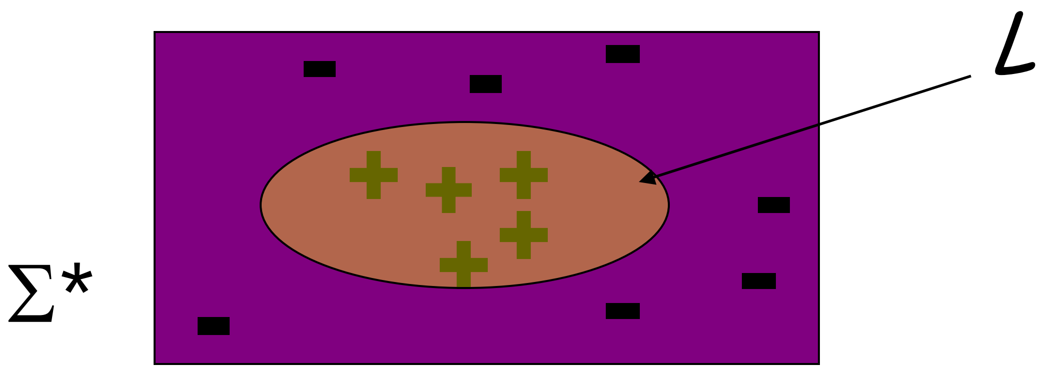


Ideas

- Use a distance between strings and try *k*-NN (nearest neighbours)
- Embed strings into vectors and use some off-the-shelf technique (decision trees, SVMs, other kernel methods)

Alternative

- Suppose the classifier is some grammatical formalism
- Thus we have *L* and $\Sigma^*\backslash L$



Informed presentations

- An *informed* presentation (or an informant) of $L \subseteq \Sigma^*$ is a function $\phi : \square \rightarrow \Sigma^* \times \{-,+\}$ such that $\phi(\square)=(L,+)\cup(L,-)$
- ϕ is an infinite succession of all the elements of Σ^* labelled to indicate if they belong or not to L.

Obviously many possible candidates

- Any Grammar *G* such that
  - $S_+ \subseteq \mathbf{L}(G)$
  - $S_- \cap \mathbf{L}(G) = \varnothing$
- But there is an infinity of such grammars!

A first bias: structural completeness

- (of S_+ *re* a *DFA* A)

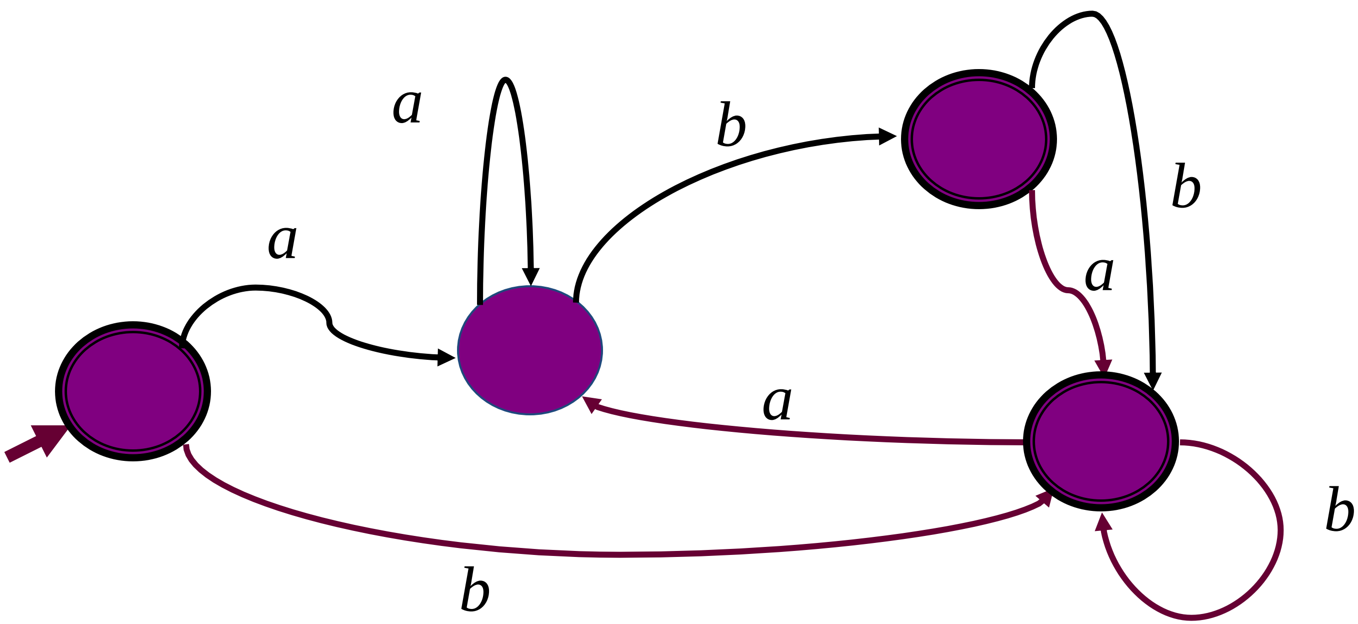
  each edge of A is used at least once by one element of S_+

  each final state accepts at least one string

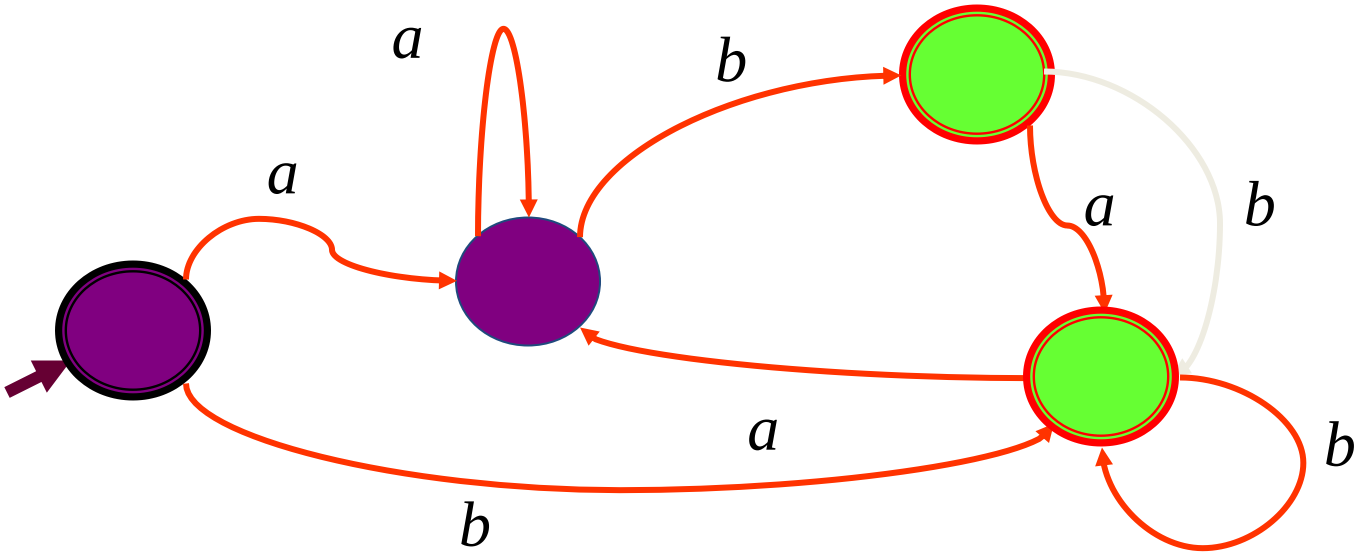
- Look only at *DFA* for which the sample is structurally complete!
- Search space becomes finite

Example

- $S_+ = \{aab,\ b,\ aaaba,\ bbaba\}$



- S_+={*aab*, *b*, *aaaba*, *bbaba*} …



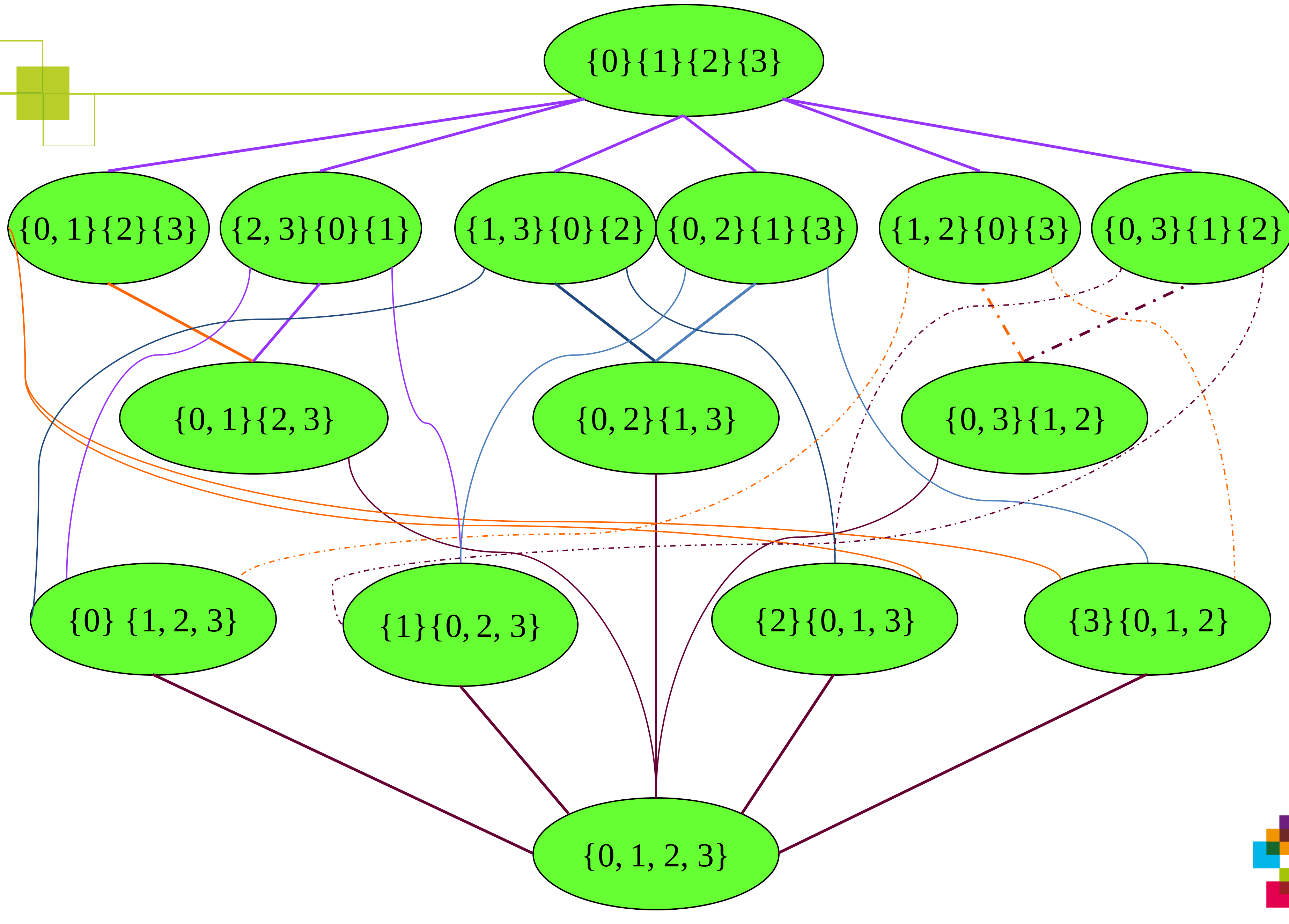
We should add λ and *abb*

Defining the search space by structural completeness

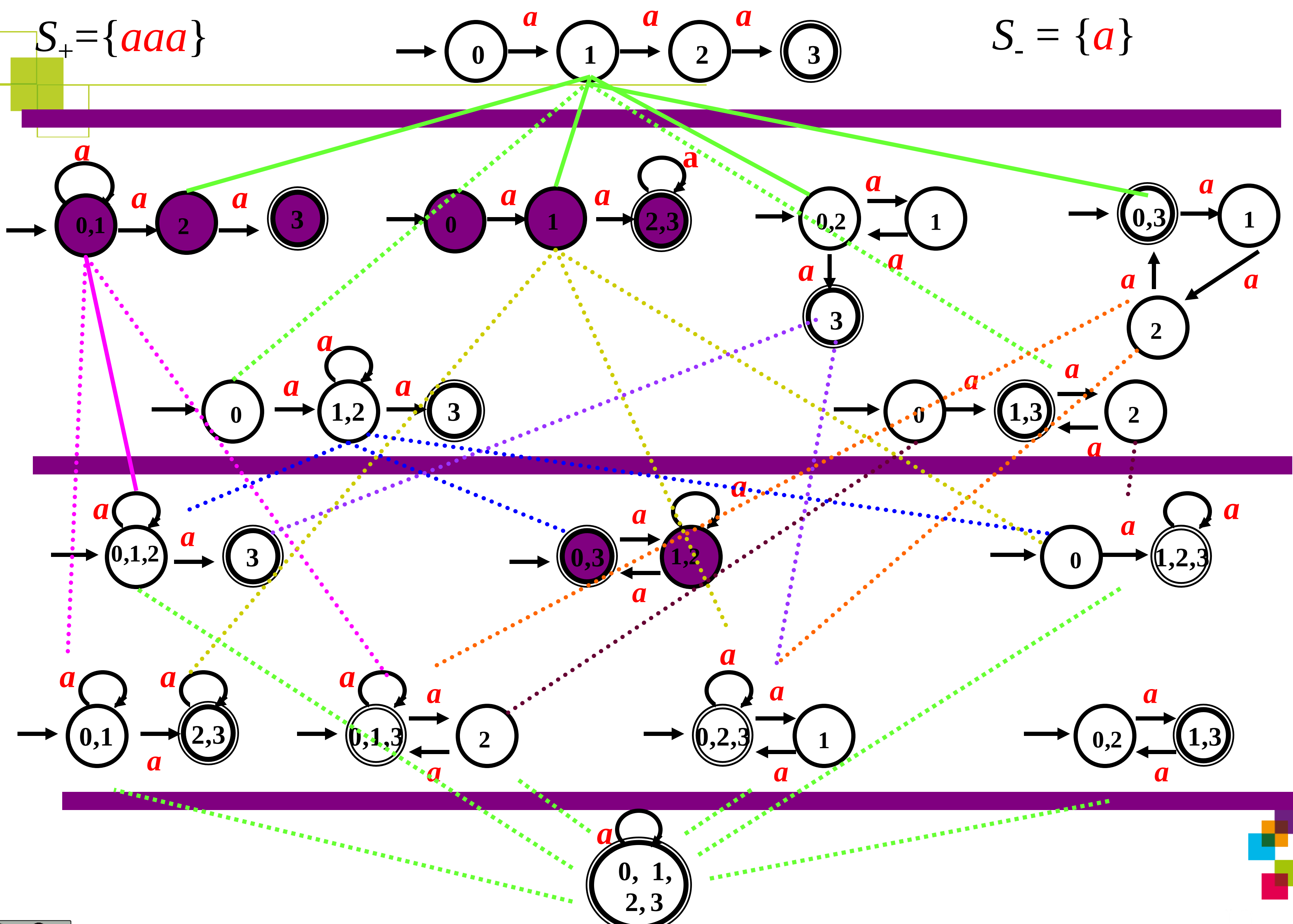
(Dupont, Miclet, Vidal 94)

- the basic operation: merging two states
- a bias on the concepts: structural completeness of the positive sample S_+
- a theorem: every biased solution can and can only be obtained by merging states in $CA(S_+)$
- the search space is a partition lattice

$CA(S_+)$ is the canonical automaton


{0}{1}{2}{3}
{0, 1}{2}{3}
{2, 3}{0}{1}
{1, 3}{0}{2}
{0, 2}{1}{3}
{1, 2}{0}{3}
{0, 3}{1}{2}
{0, 1}{2, 3}
{0, 2}{1, 3}
{0, 3}{1, 2}
{0} {1, 2, 3}
{1}{0, 2, 3}
{2}{0, 1, 3}
{3}{0, 1, 2}
{0, 1, 2, 3}

$S_+ = \{aaa\}$ $S_- = \{a\}$



The partition lattice

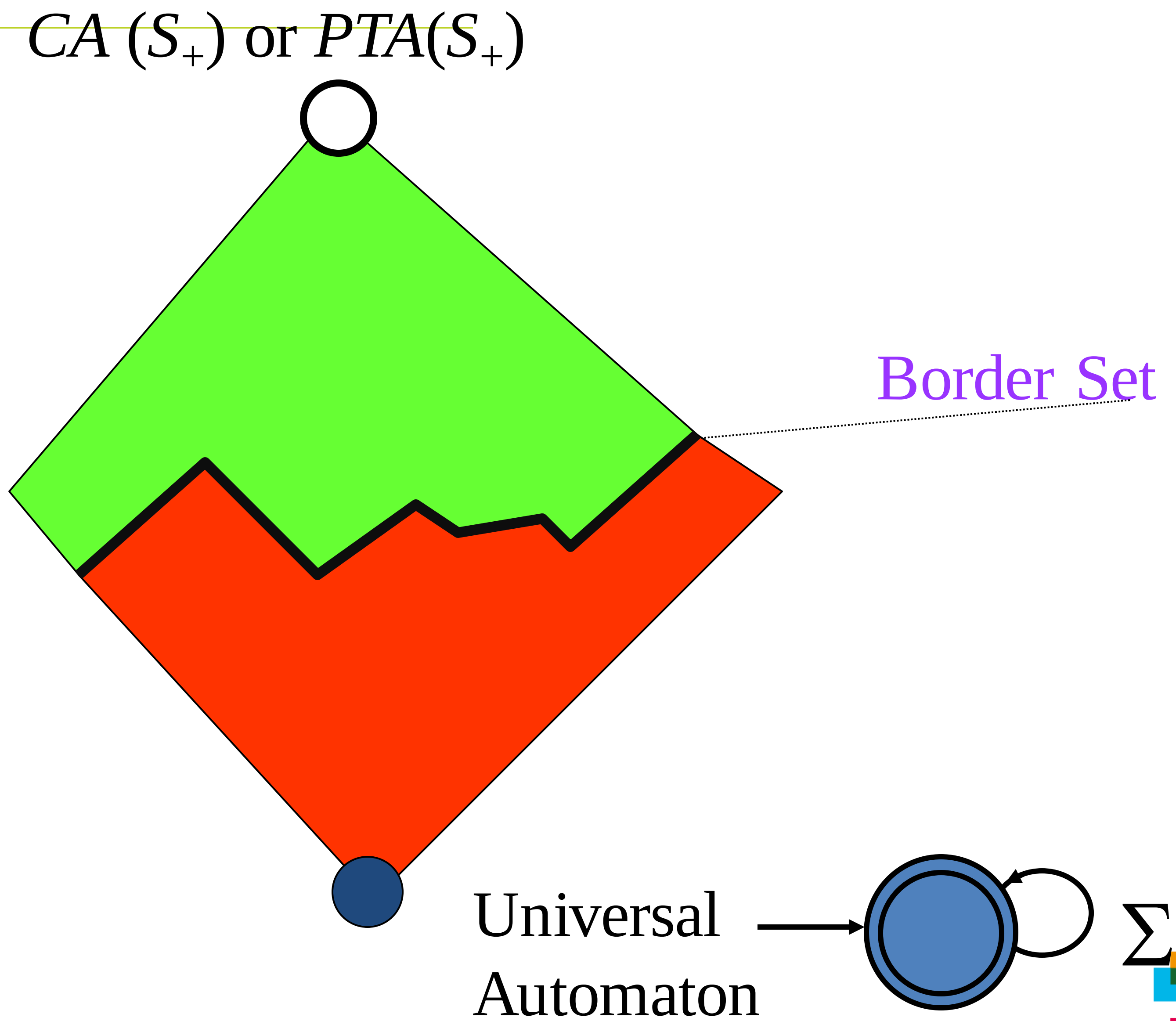
- Let *E* be a set with *n* elements
- The number of partitions of *E* is given by the Bell number

$$\begin{cases} \omega(0) = 1 \\ \omega(n+1) = \sum_{p=0}^{n} \binom{n}{p} . \, \omega(n) \end{cases}$$

$\omega(16)=$10 480 142 147

Regular inference as search

- another result: the smallest *DFA* fitting the examples is in the lattice constructed on $PTA(S_+)$
- generally, algorithms would start from $PTA(S_+)$ and explore the corresponding lattice of solutions using the merging operation. S_- is used to control the generalization.

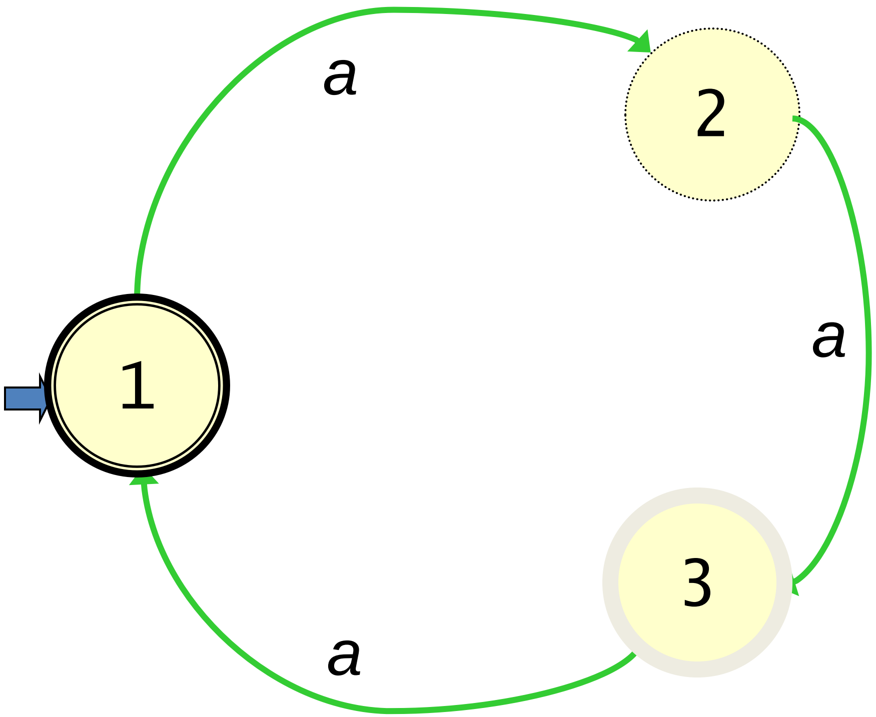

$CA\ (S_+)$ or $PTA(S_+)$
Border Set
Universal
Automaton
Σ

3. Basic structures

Two types of final states

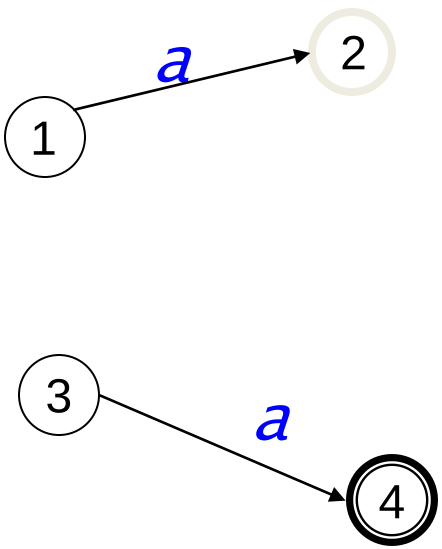
$S_+=\{\lambda, \ aaa\}$

$S_-=\{aa, \ aaaaa\}$

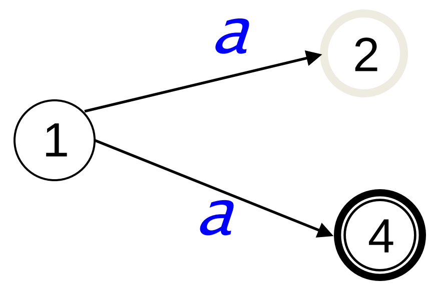


State 1 is accepting
State 3 is rejecting
What about state 2?

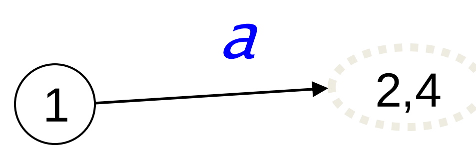
What is determinism about?



Merge 1 and 3?



But…

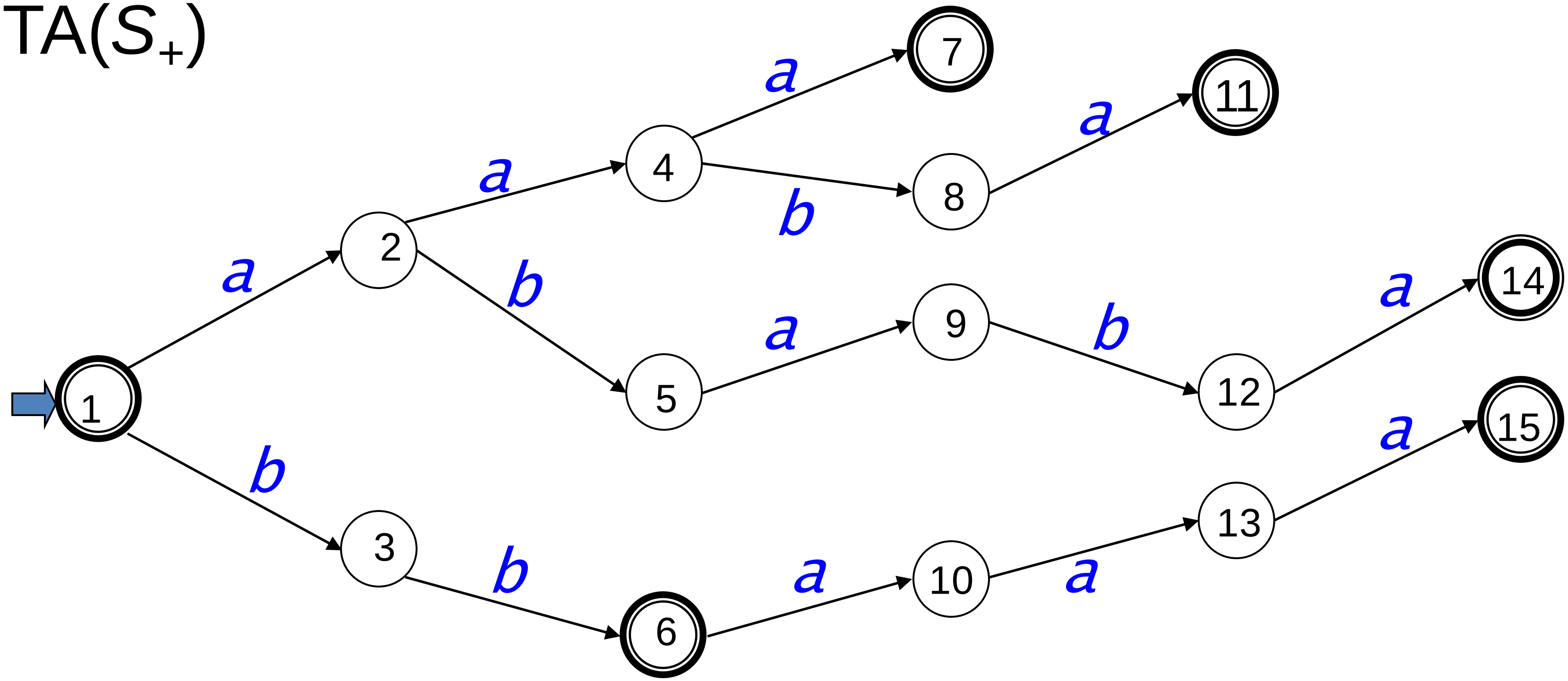


The prefix tree acceptor

- The smallest tree-like DFA consistent with the data
- Is a solution to the learning problem
- Corresponds to a rote learner

From the sample to the PTA

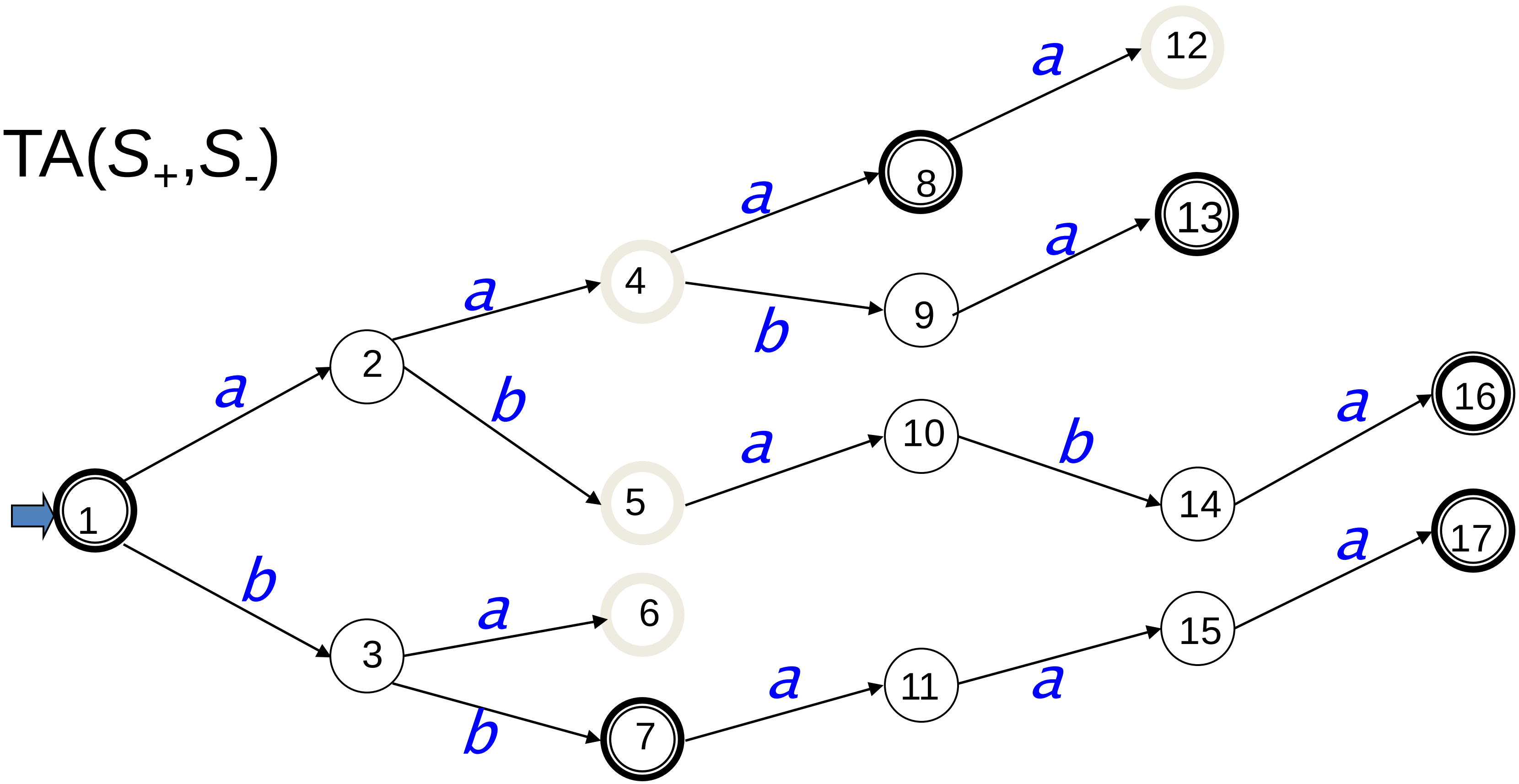
PTA(S_+)



$S_+=\{\lambda,\ aaa,\ aaba,\ ababa,\ bb,\ bbaaa\}$
$S_-=\{aa,\ ab,\ aaaa,\ ba\}$

From the sample to the PTA (full PTA)

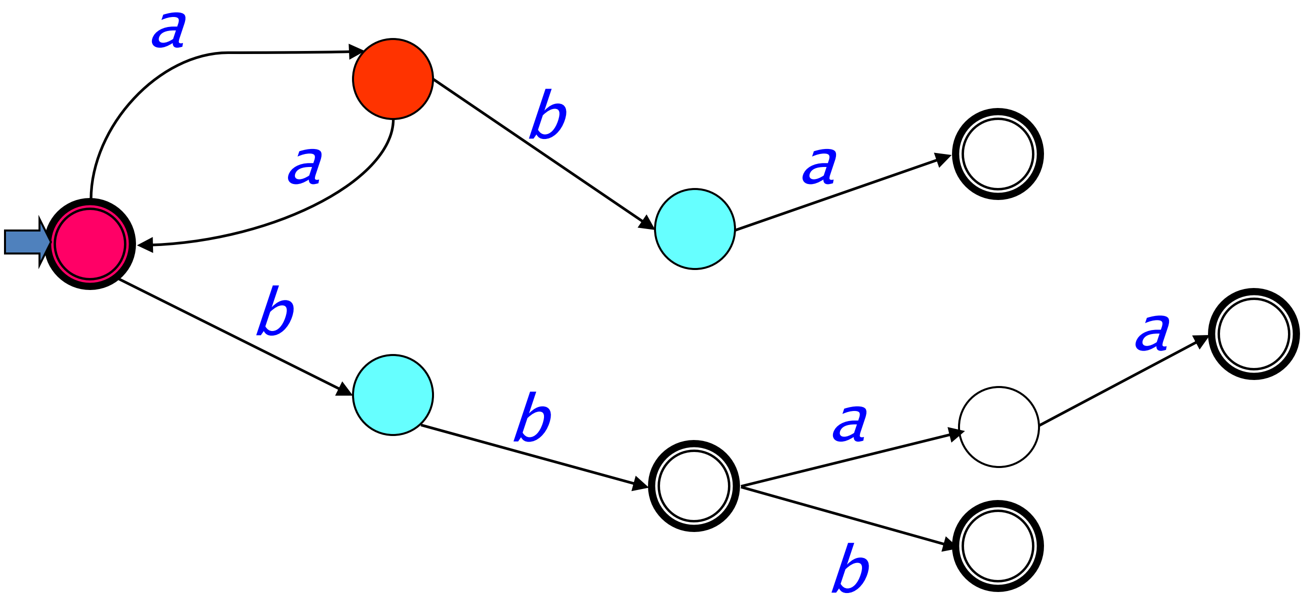
PTA(S_+,S_-)



S_+={λ, *aaa*, *aaba*, *ababa*, *bb*, *bbaaa*}
S_-={*aa*, *ab*, *aaaa*, *ba*}

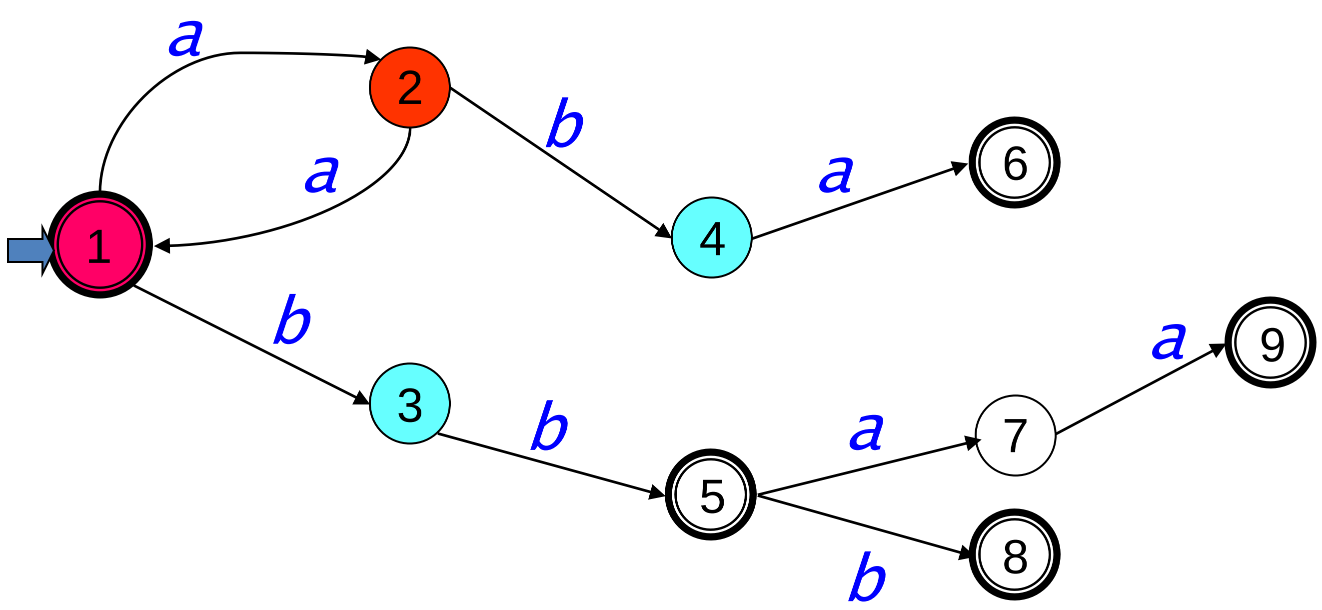
Red, Blue and White states

-Red states are confirmed states
-Blue states are the (non Red) successors of the Red states
-White states are the others



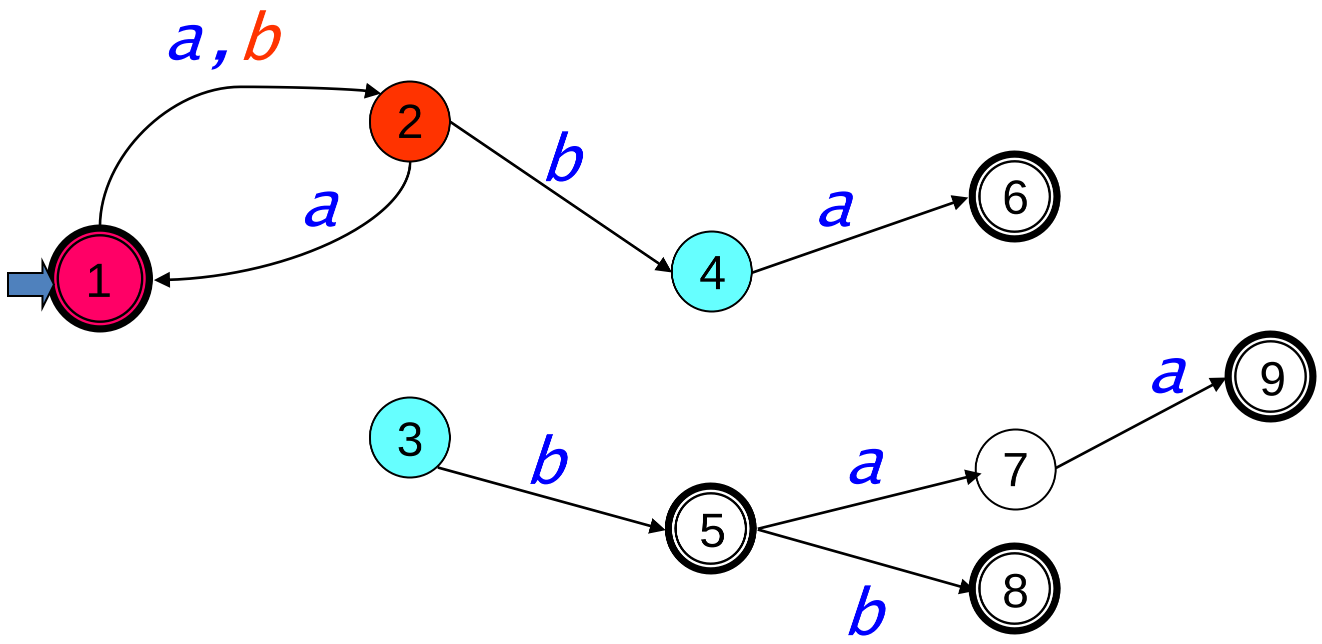
Merge and fold

Suppose we want to merge state 3 with state 2



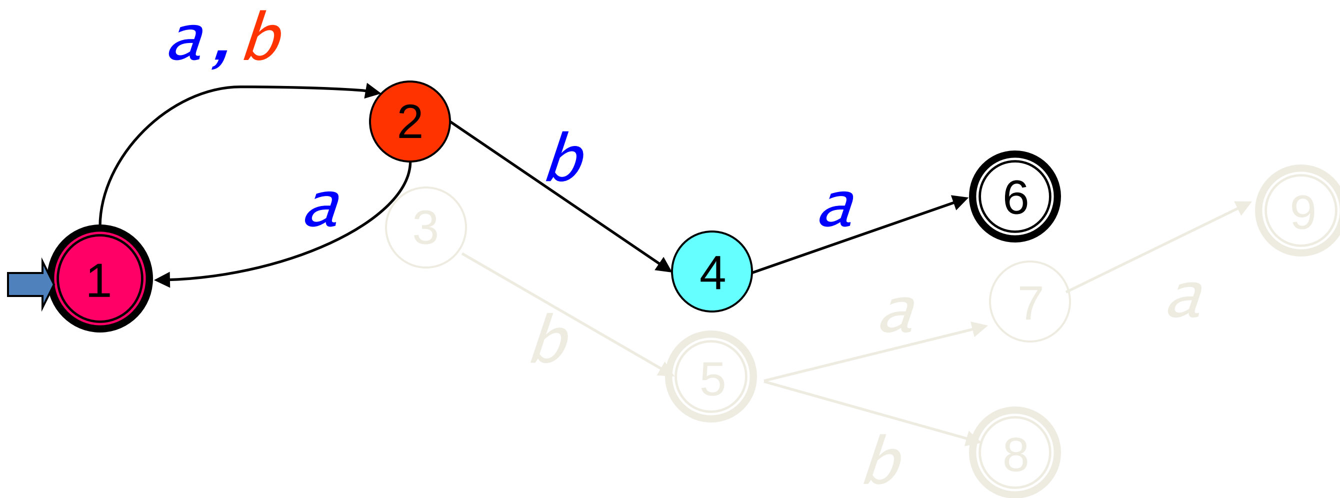
Merge and fold

First disconnect 3
and reconnect to 2



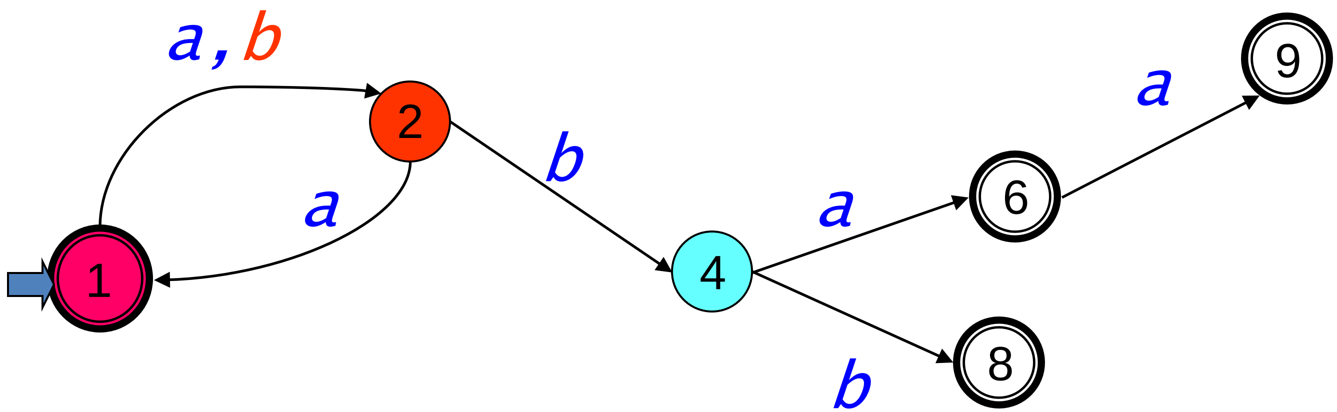
Merge and fold

Then fold subtree rooted in 3 into the DFA starting in 2



Merge and fold

Then fold subtree rooted in 3 into the DFA starting in 2



Other search spaces

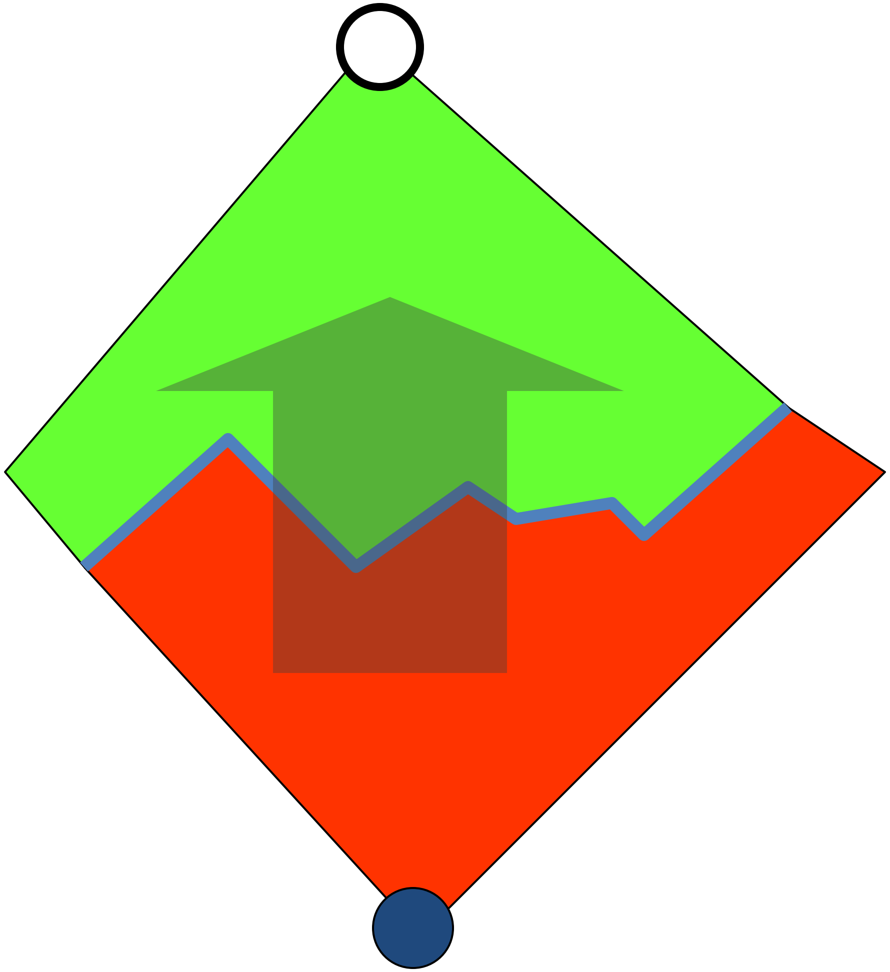
an augmented *PTA* can be constructed from both S_+ and S_- (Coste 98, Oliveira 98)

- but not every merge is possible
- the search algorithms must run under a set of dynamic constraints

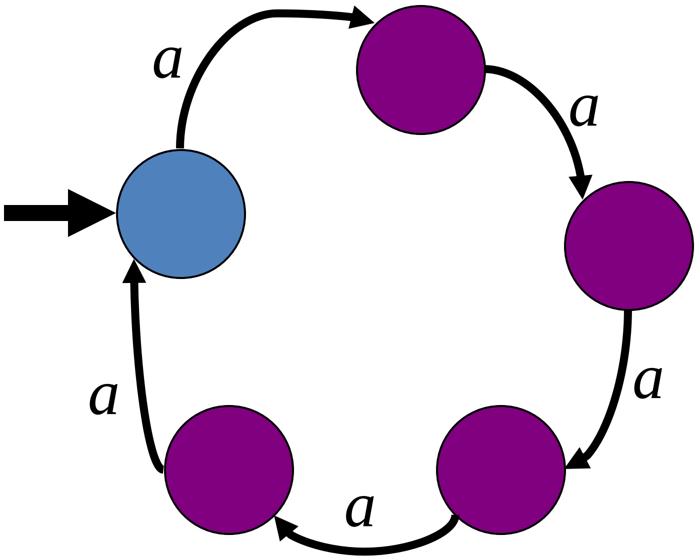
State splitting

Searching by splitting:

start from the one-state universal automaton, keep constructing *DFA* controlling the search with <S_+, S_->



That seems a good idea… but take a^{5^*}. What 4 (or 3, 2, 1) state automaton is a decent approximation of a^{5^*} ?



Go to RPNI

4. RPNI
Regular Positive and Negative Grammatical Inference

Inferring regular languages in polynomial time. Jose Oncina & Pedro García. *Pattern recognition and image analysis, 1992*

- RPNI is a state merging algorithm
- RPNI identifies any regular language in the limit
- RPNI works in polynomial time
- RPNI admits polynomial characteristic sets

$A \leftarrow$ PTA(S+);

$Red \leftarrow \{q_I\}$

$Blue \leftarrow \{\delta(q_I,a)$: $a \in \Sigma$ };

While $Blue \neq \emptyset$ do

choose q from $Blue$

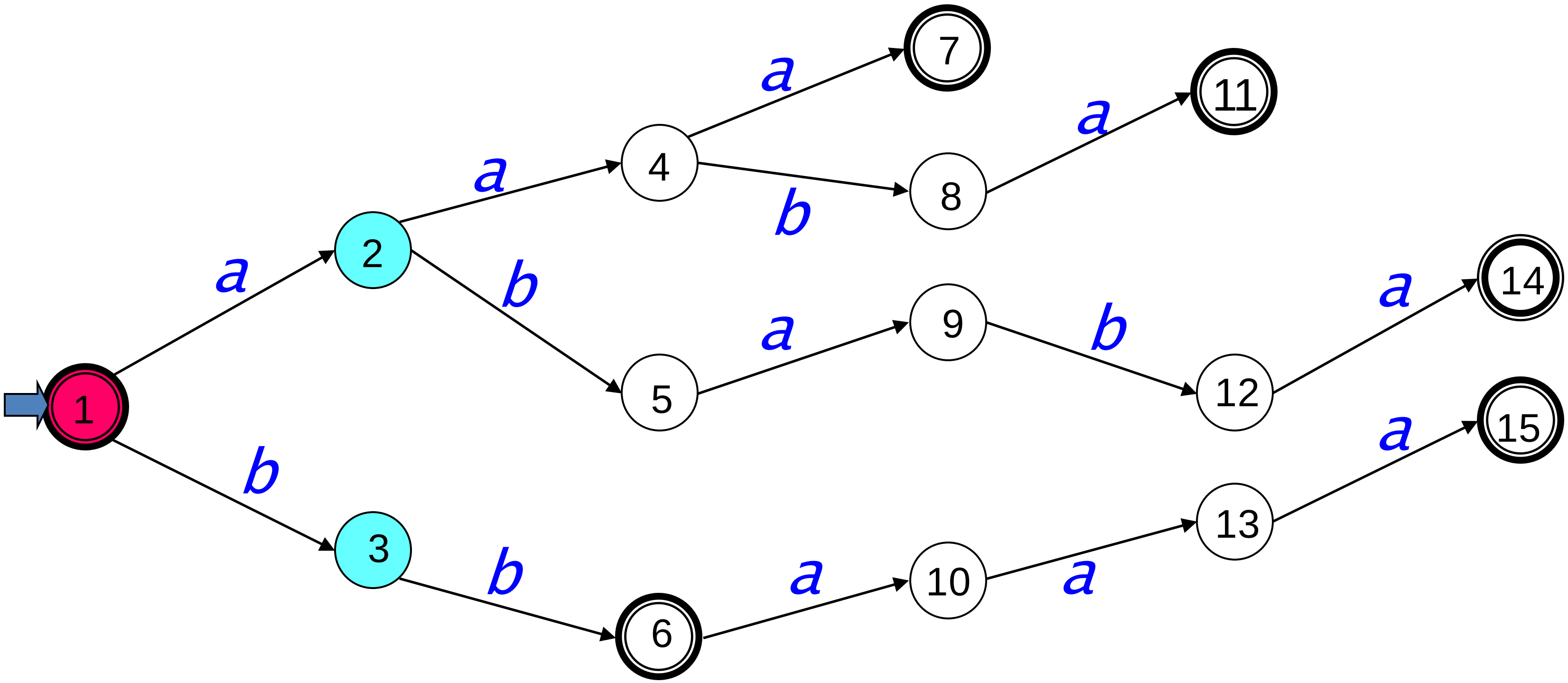
if $\exists p \in Red$: $\mathbf{L}$(merge_and_fold(A,p,q))$\cap S- = \emptyset$

then $A \leftarrow$ merge_and_fold(A,p,q)

else $Red \leftarrow Red \cup \{q\}$

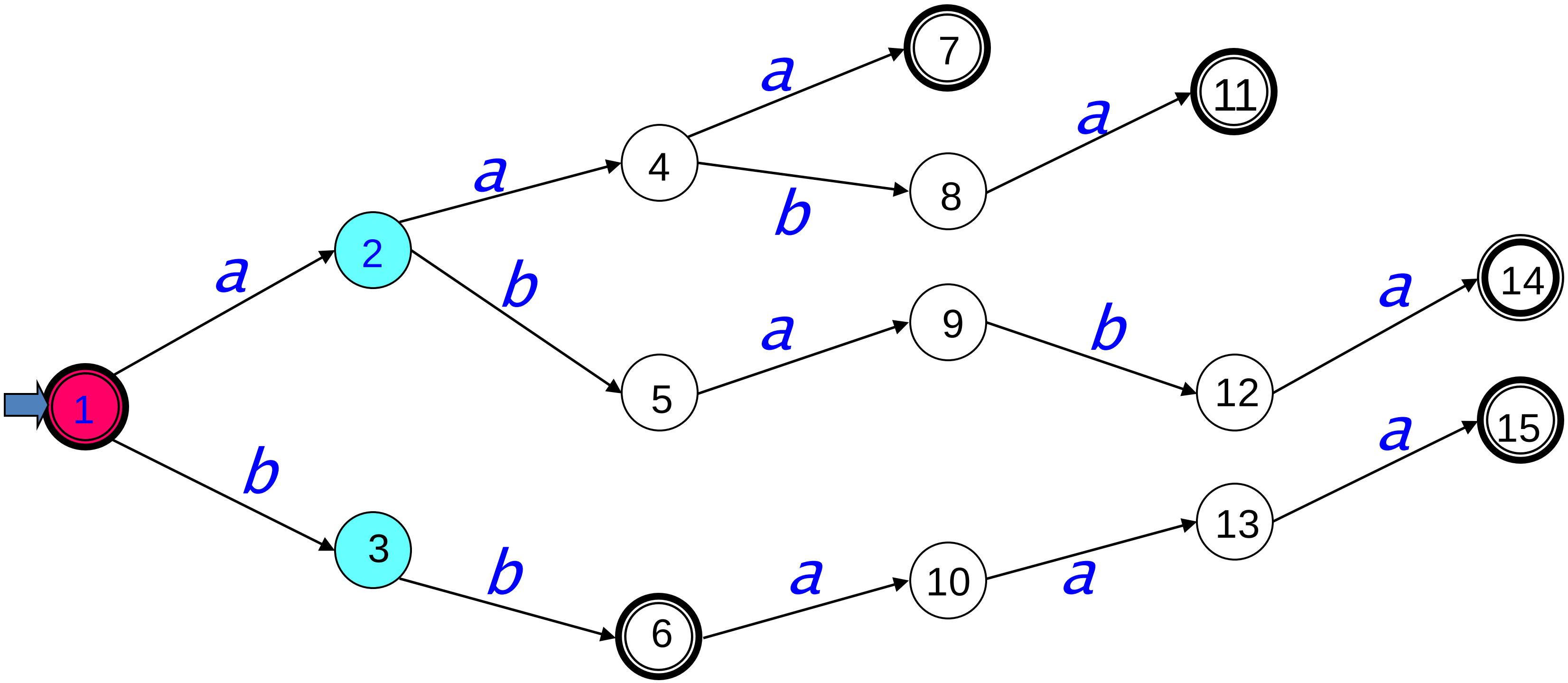
$Blue \leftarrow \{\delta(q,a)$: $q \in Red\}$ - $\{Red\}$

$S_+=\{\lambda, \ aaa, \ aaba, \ ababa, \ bb, \ bbaaa\}$



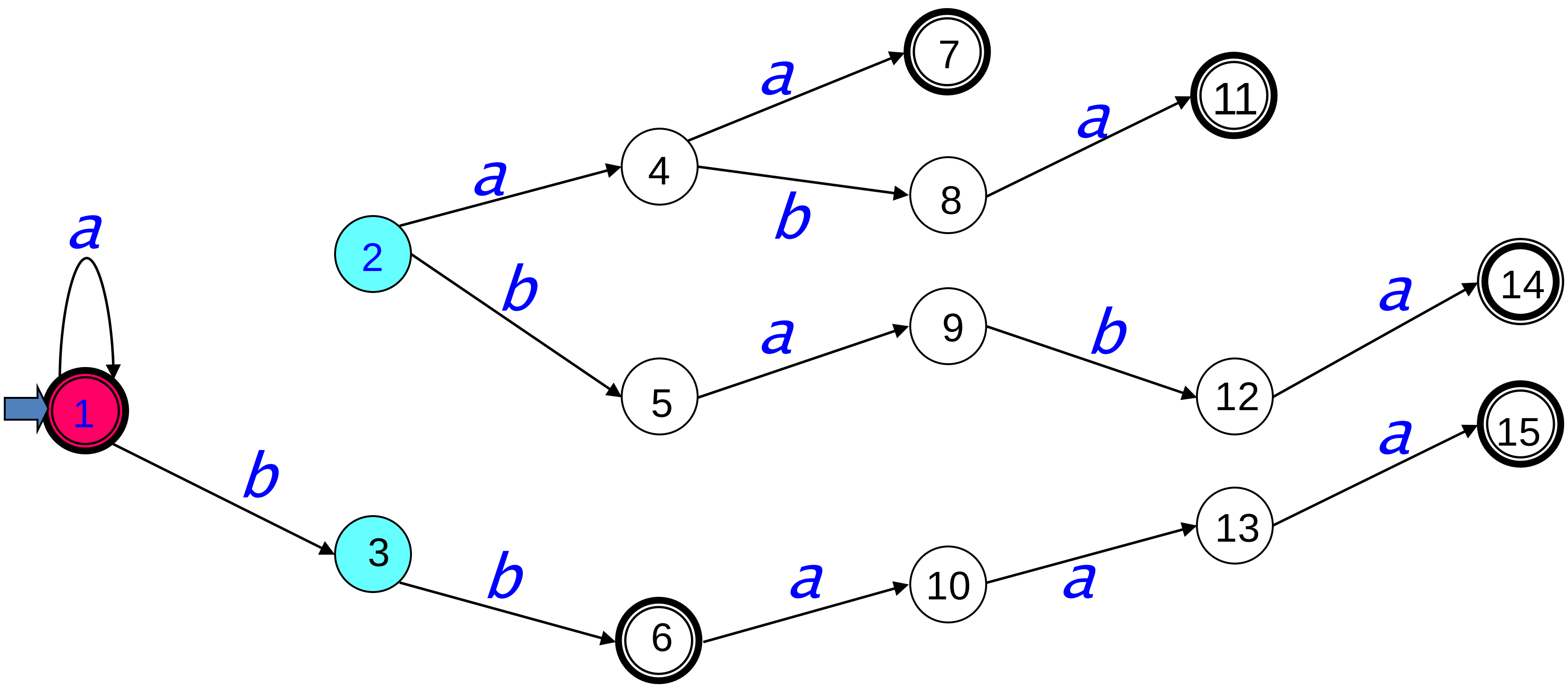
$S_-=\{aa, \ ab, \ aaaa, \ ba\}$

Try to merge 2 and 1



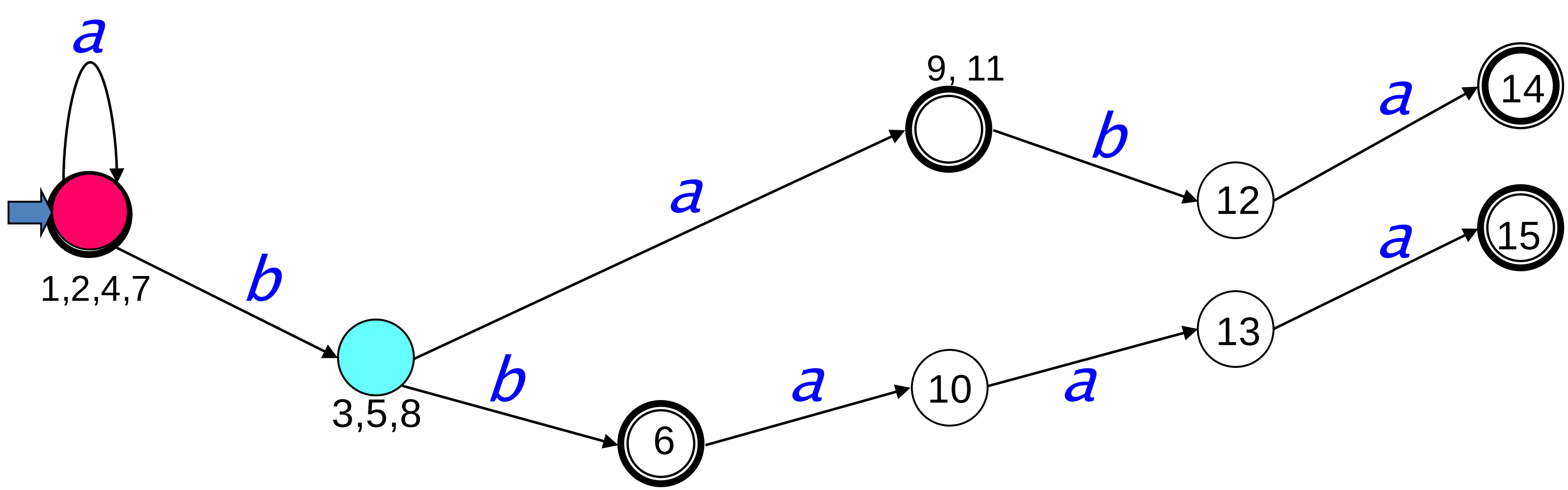
$S_-=\{aa,\ ab,\ aaaa,\ ba\}$

First merge, then fold



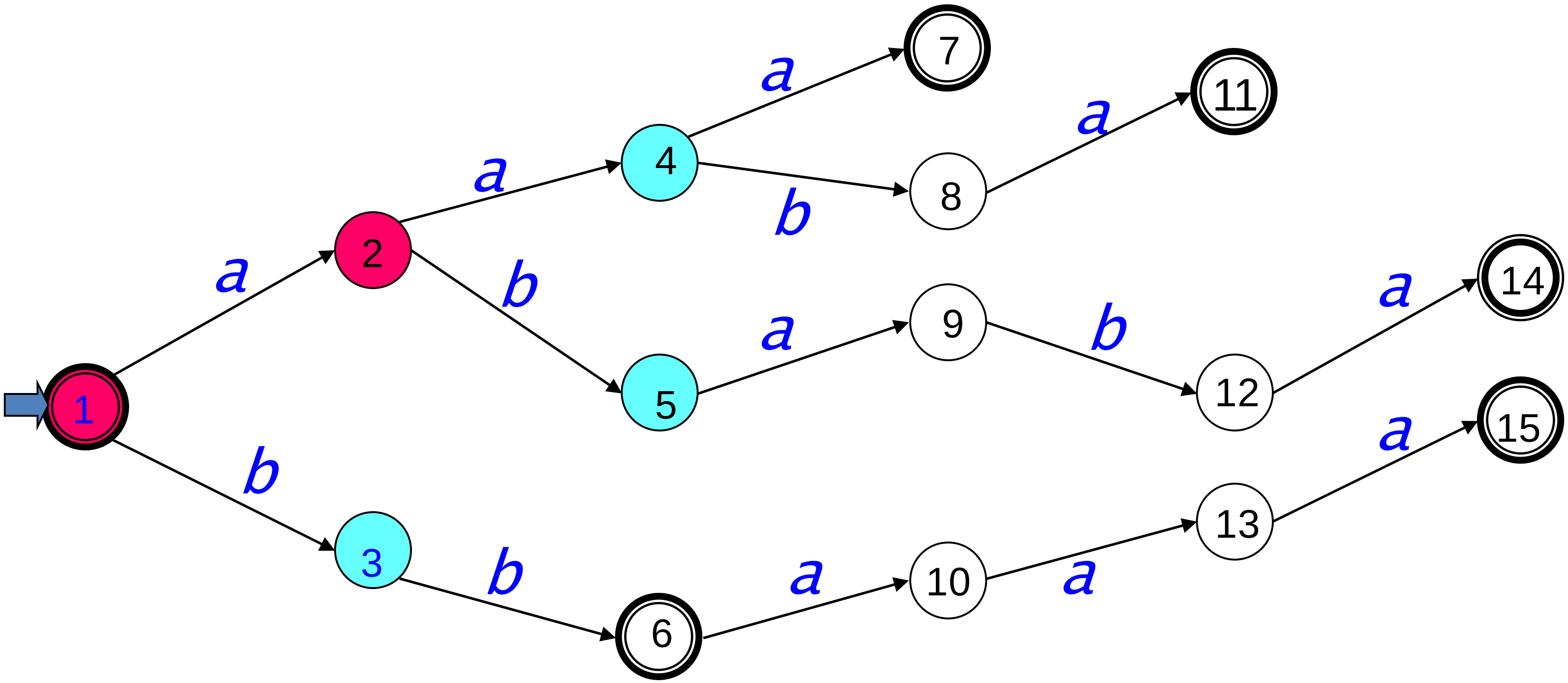
$S_{-}=\{aa,\ ab,\ aaaa,\ ba\}$

But now string aaaa is accepted, so the merge must be rejected, and state 2 is promoted



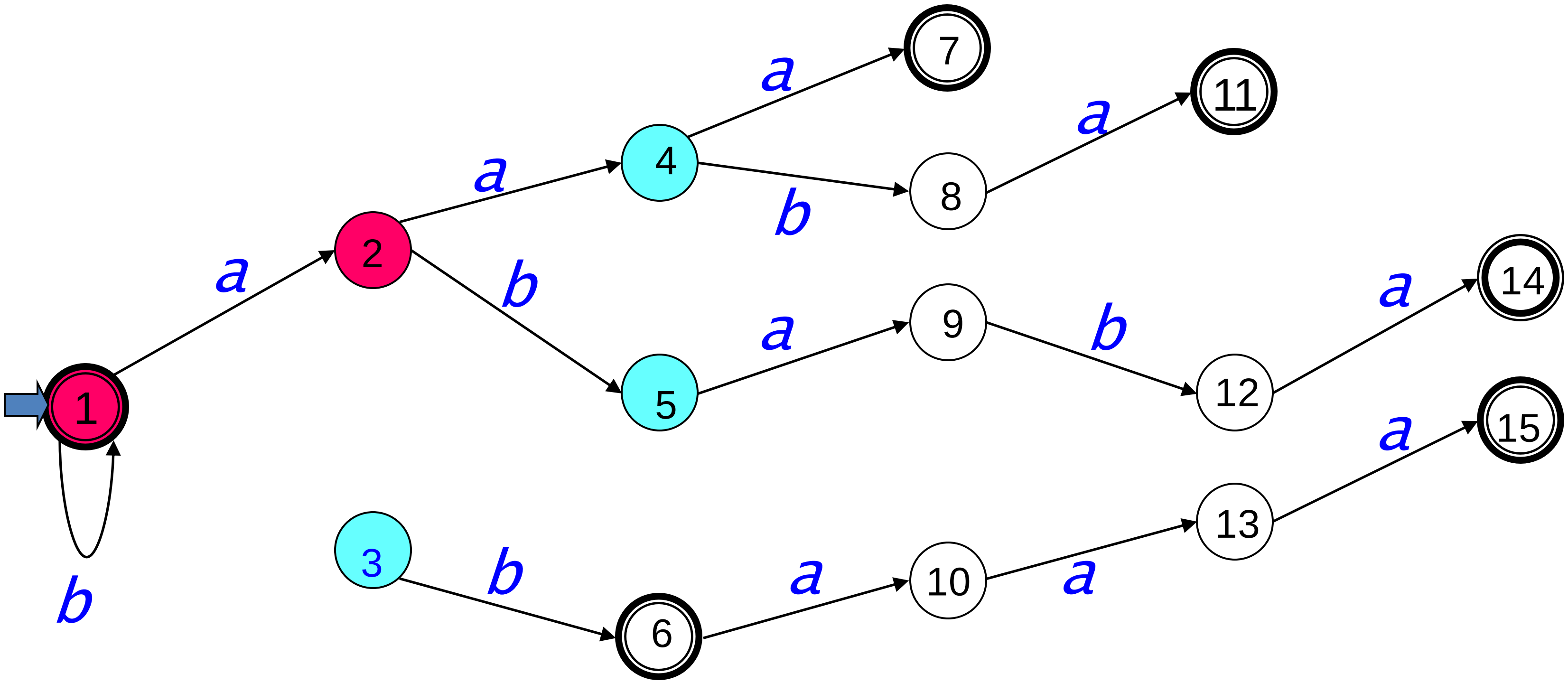
S_{-}={*aa*, *ab*, *aaaa*, *ba*}

Try to merge 3 and 1



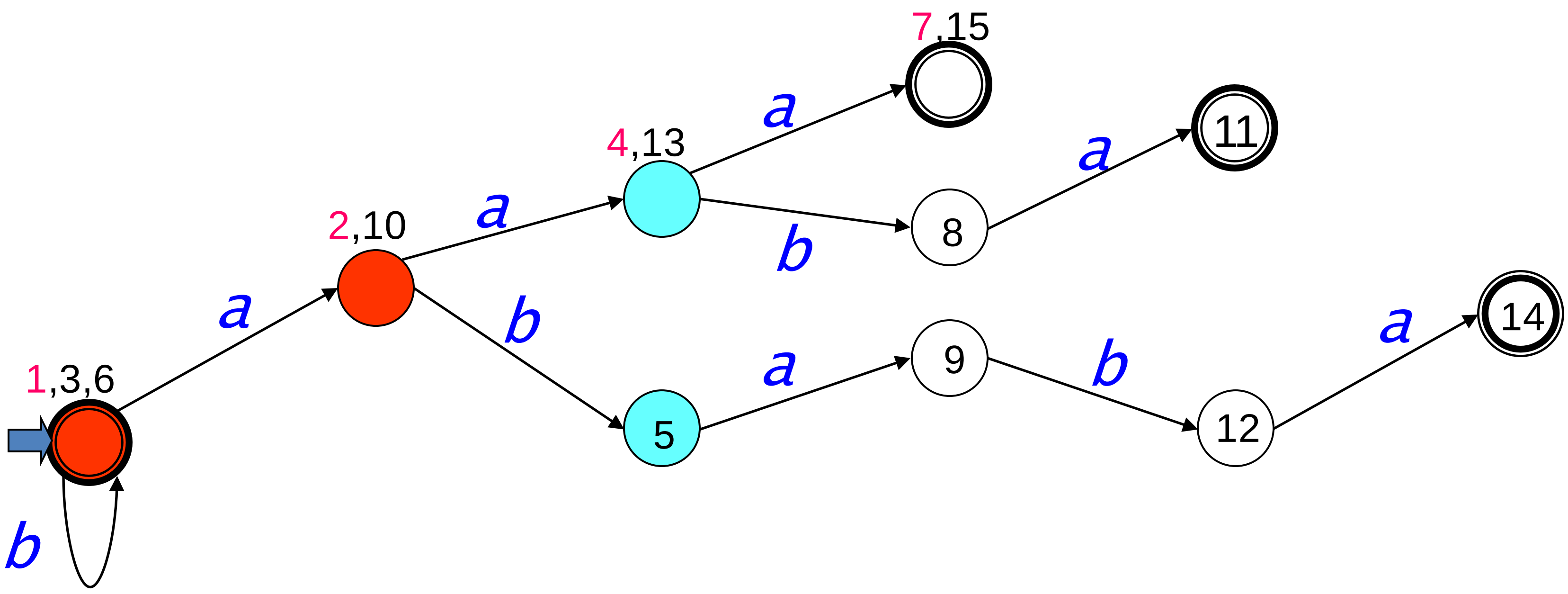
$S_- = \{aa,\ ab,\ aaaa,\ ba\}$

First merge, then fold



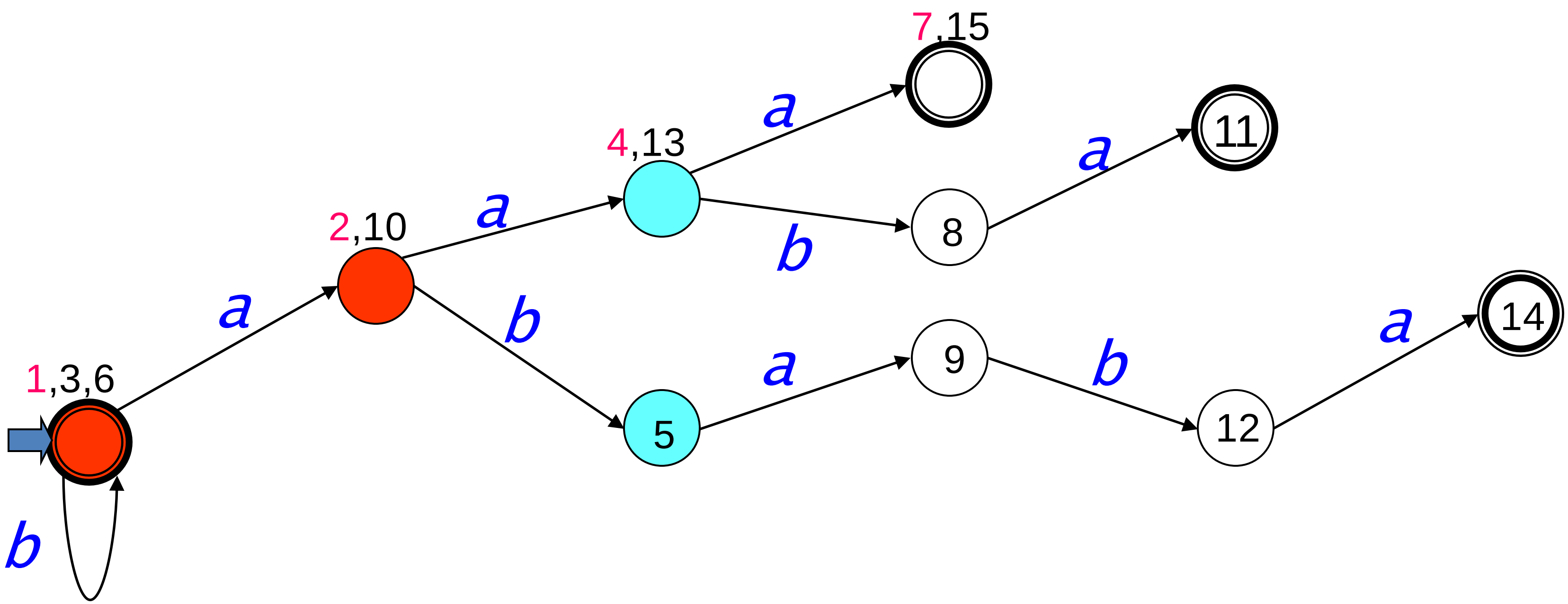
$S_-=\{aa,\ ab,\ aaaa,\ ba\}$

No counter example is accepted so the merge is kept



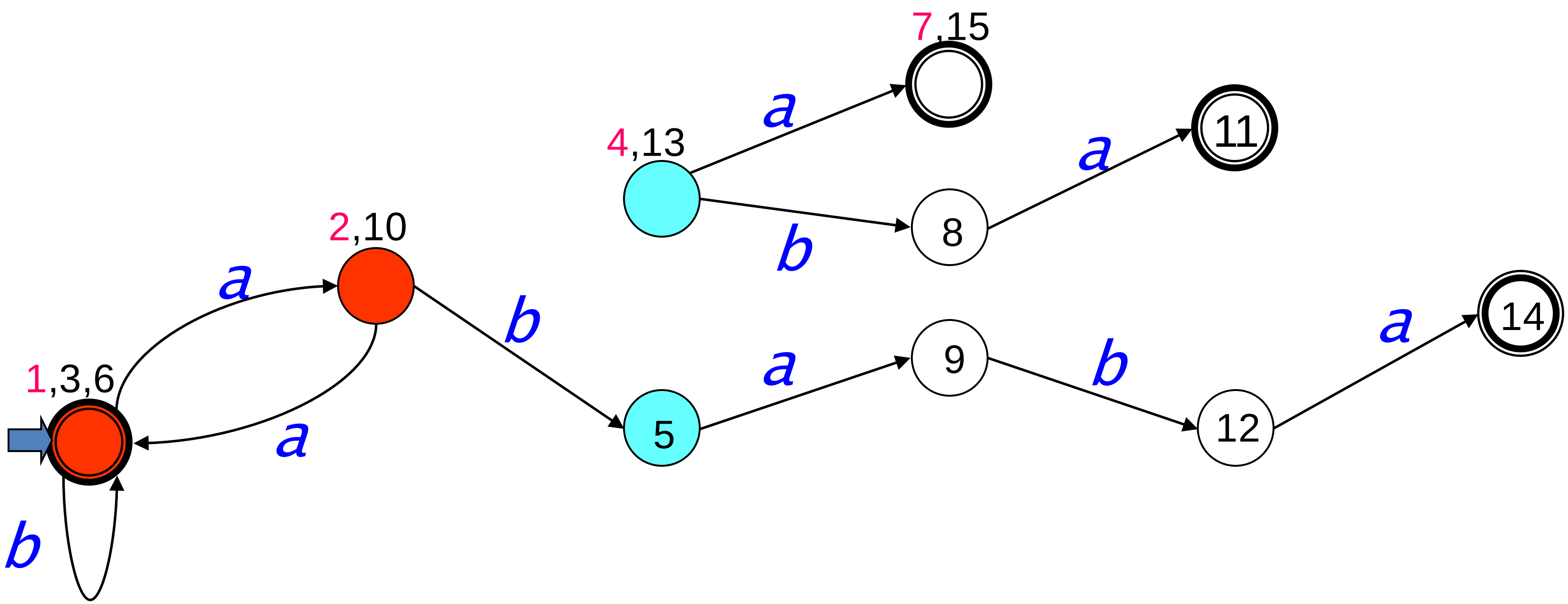
S_{-}={*aa*, *ab*, *aaaa*, *ba*}

Next possible merge to be checked is {4,13} with {1,3,6}



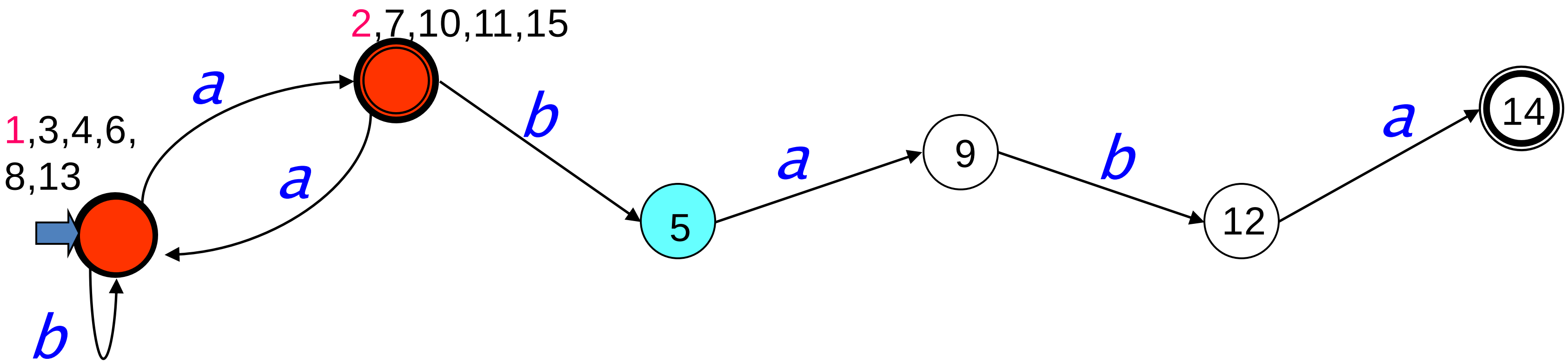
$S_{-}=\{aa, ab, aaaa, ba\}$

Merged. Needs folding subtree in {4,13} with {1,3,6}



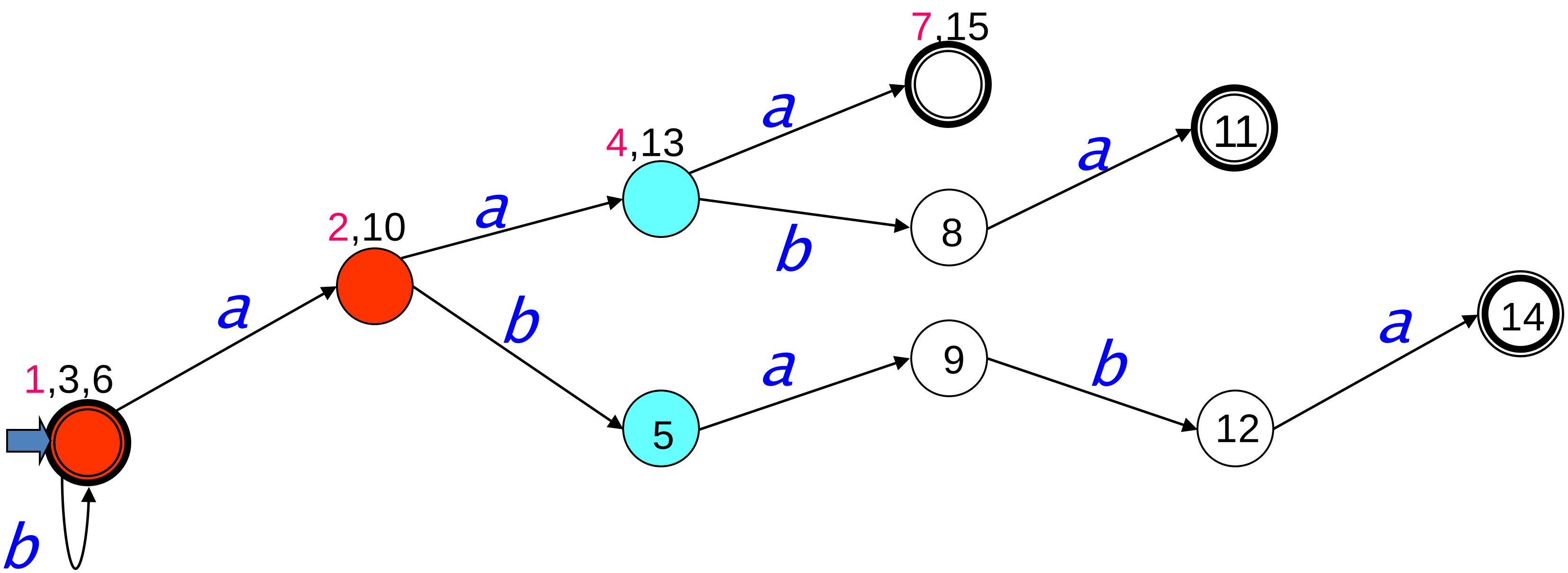
S_{-}={*aa*, *ab*, *aaaa*, *ba*}

But now aa is accepted



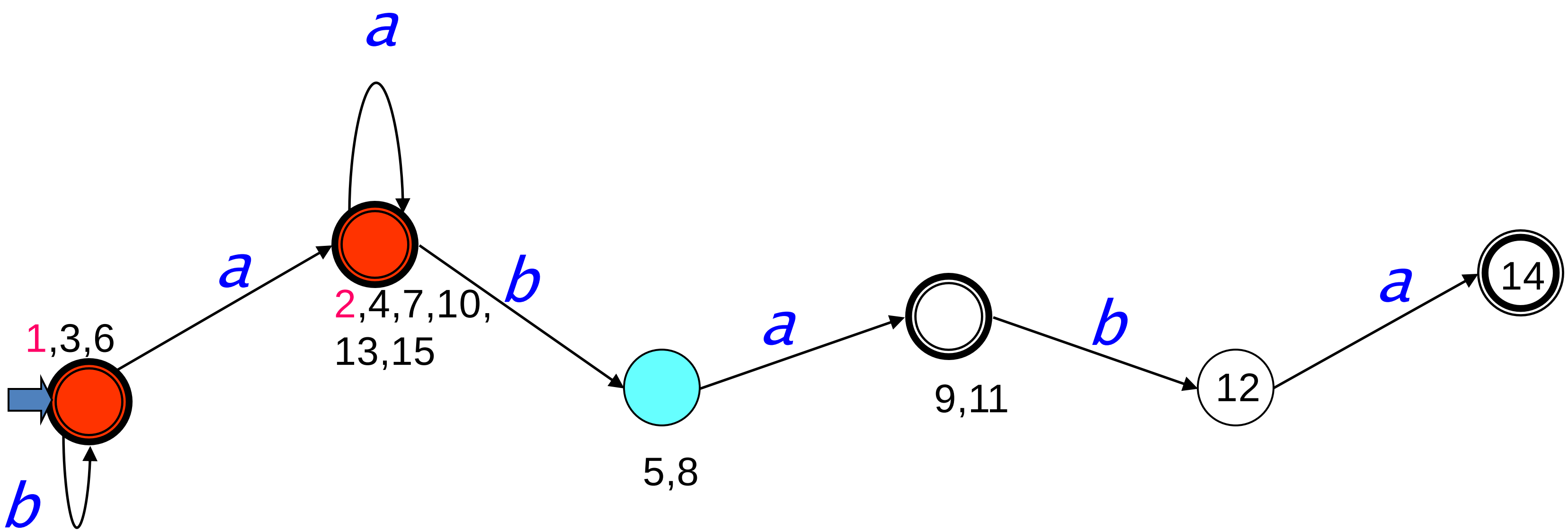
S_{-}={aa, ab, aaaa, ba}

So we try {4,13} with {2,10}



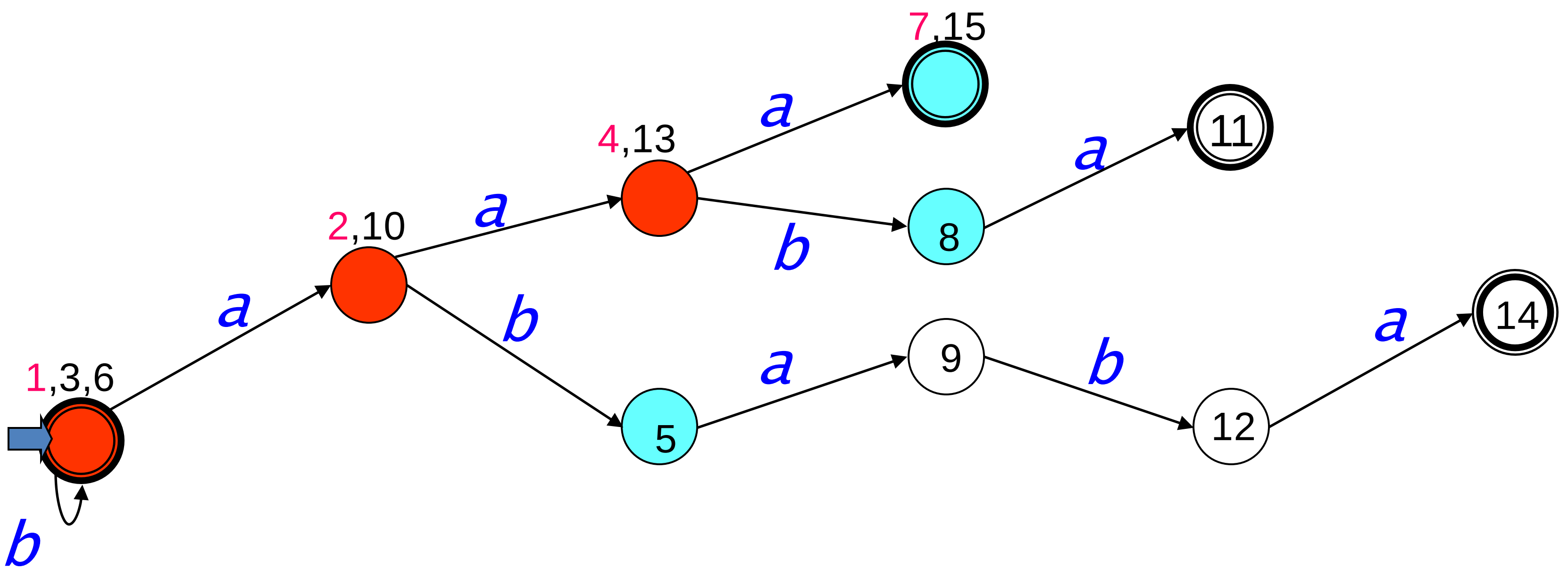
$S_- = \{aa,\ ab,\ aaaa,\ ba\}$

Negative string aa is again accepted. Since we have tried all Red for merging, state 4 is promoted.



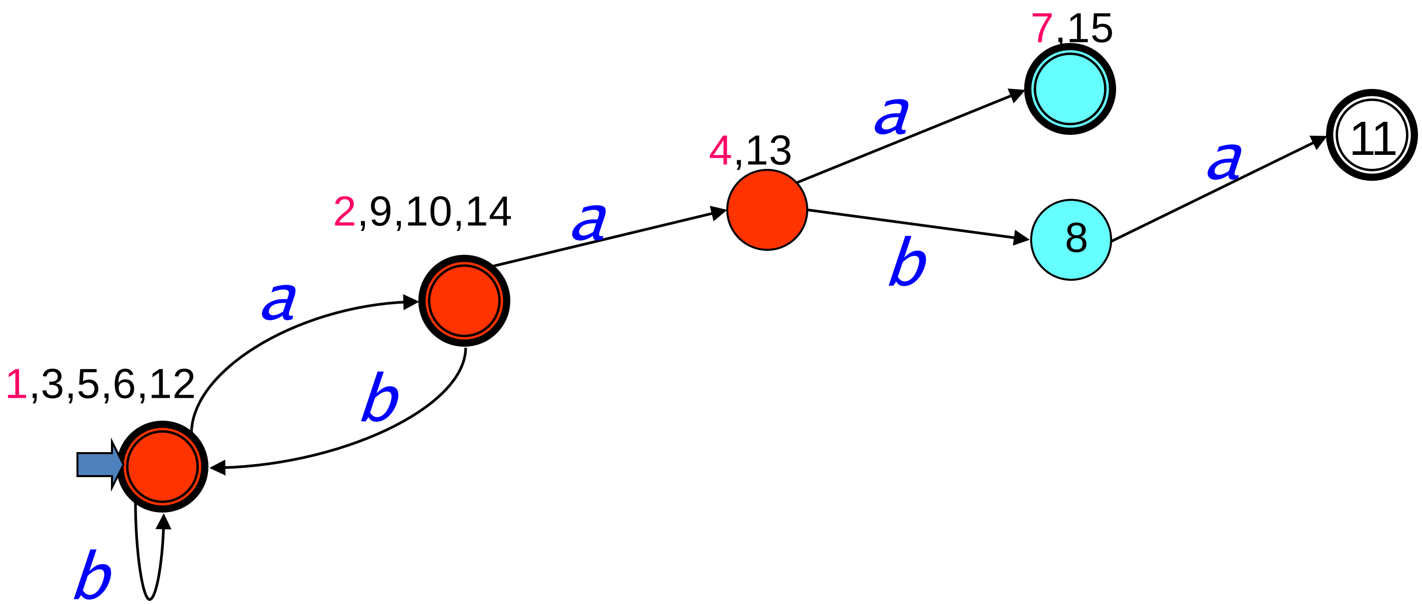
S_{-}={*aa*, *ab*, *aaaa*, *ba*}

So we try 5 with {1,3,6}



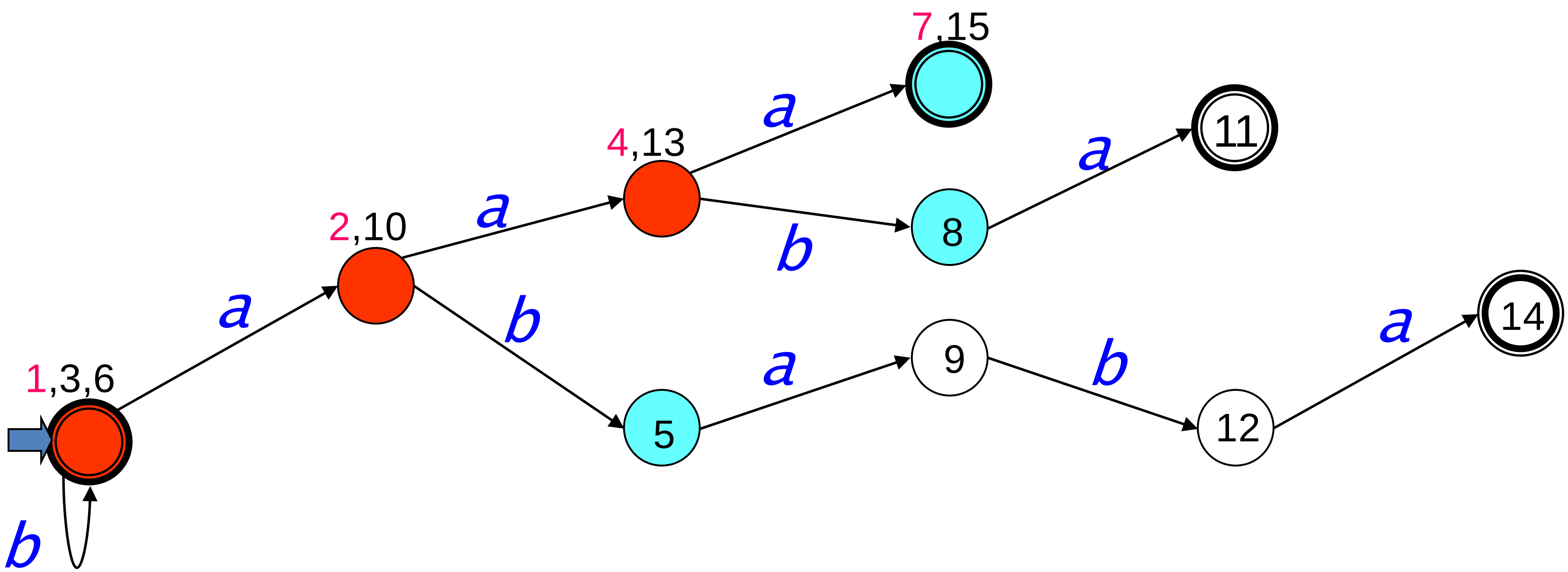
S_{-}={*aa*, *ab*, *aaaa*, *ba*}

But again we accept *ab*



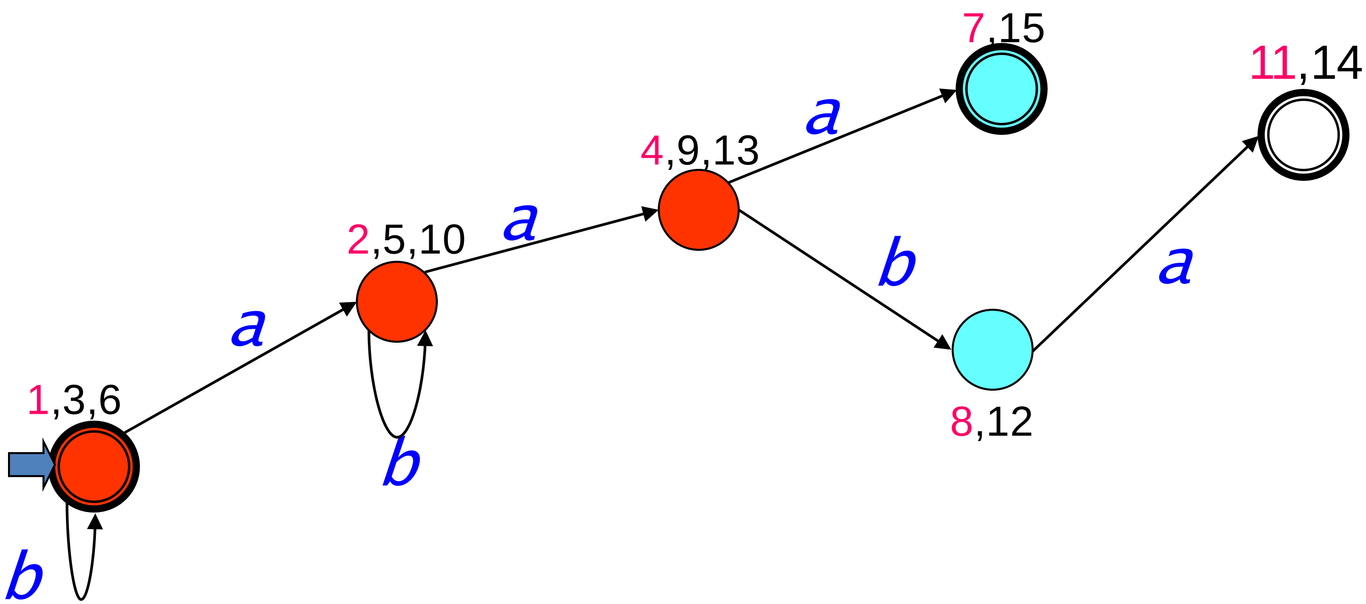
$S_{-}=\{aa,\ ab,\ aaaa,\ ba\}$

So we try 5 with {2,10}



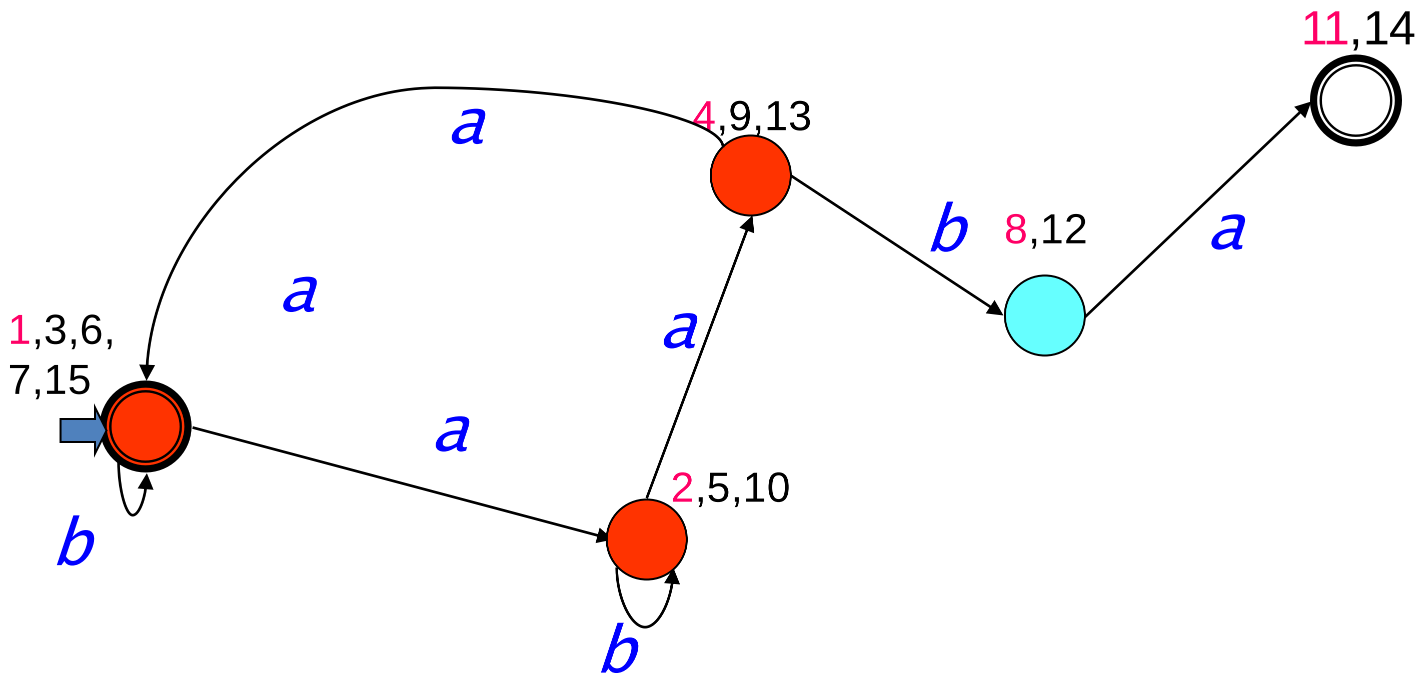
$S_{-}=\{aa,\ ab,\ aaaa,\ ba\}$

Which is OK. So next possible merge is {7,15} with {1,3,6}



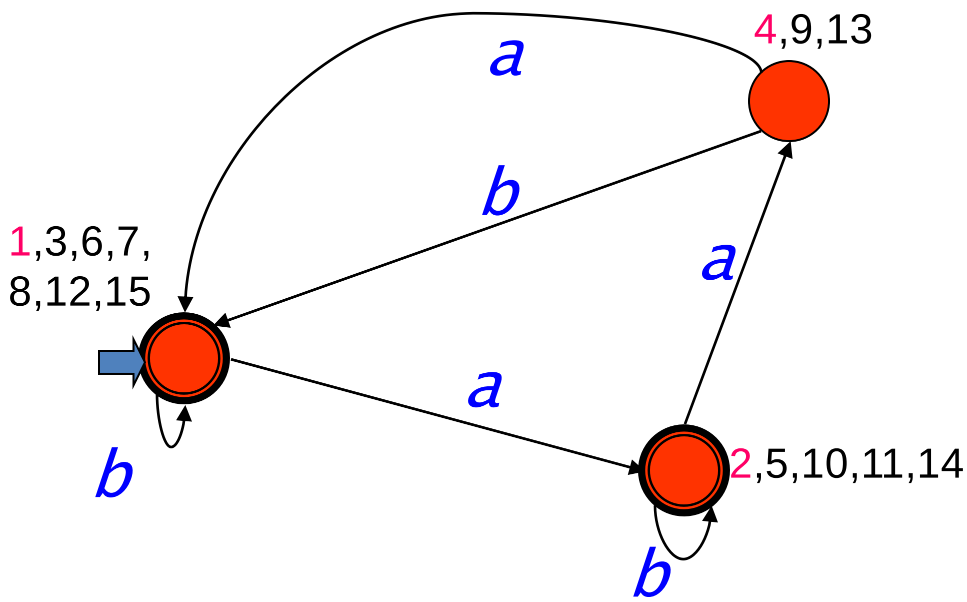
$S_- = \{aa,\ ab,\ aaaa,\ ba\}$

Which is OK. Now try to merge {8,12} with {1,3,6,7,15}



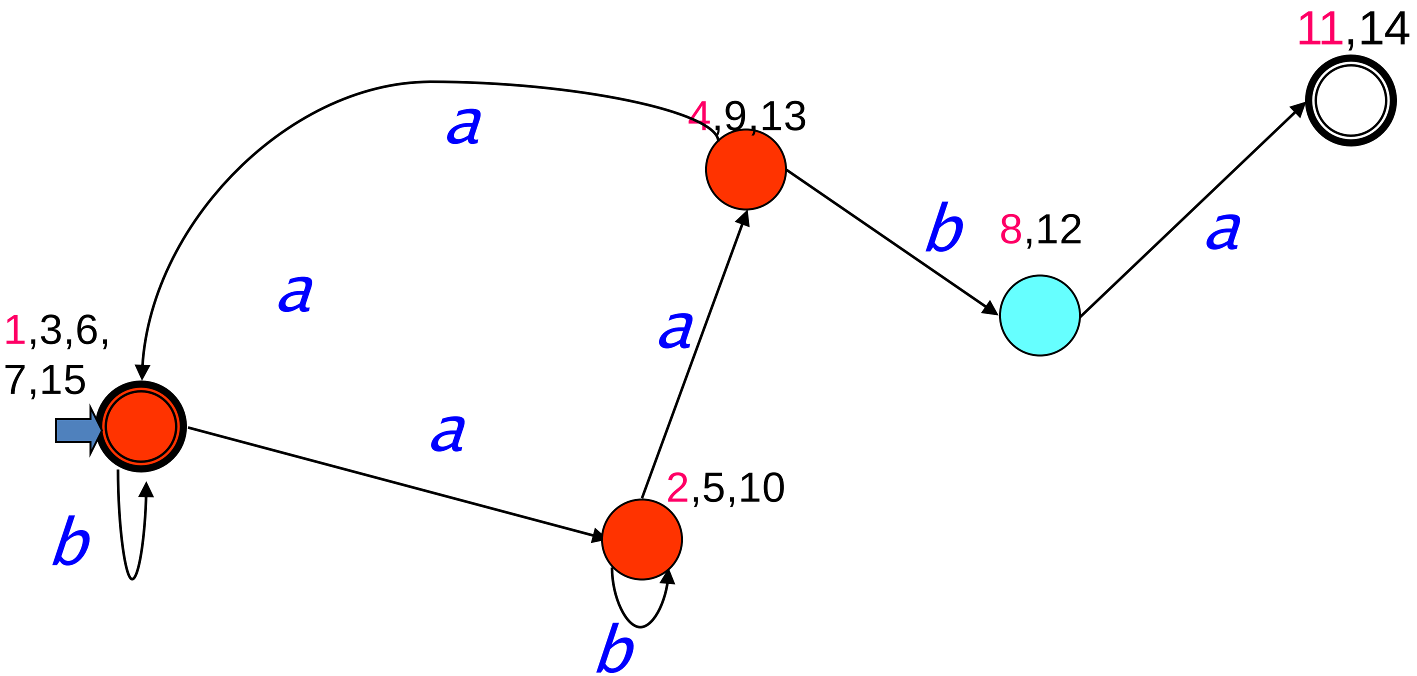
S_{-}={*aa*, *ab*, *aaaa*, *ba*}

And ab is accepted



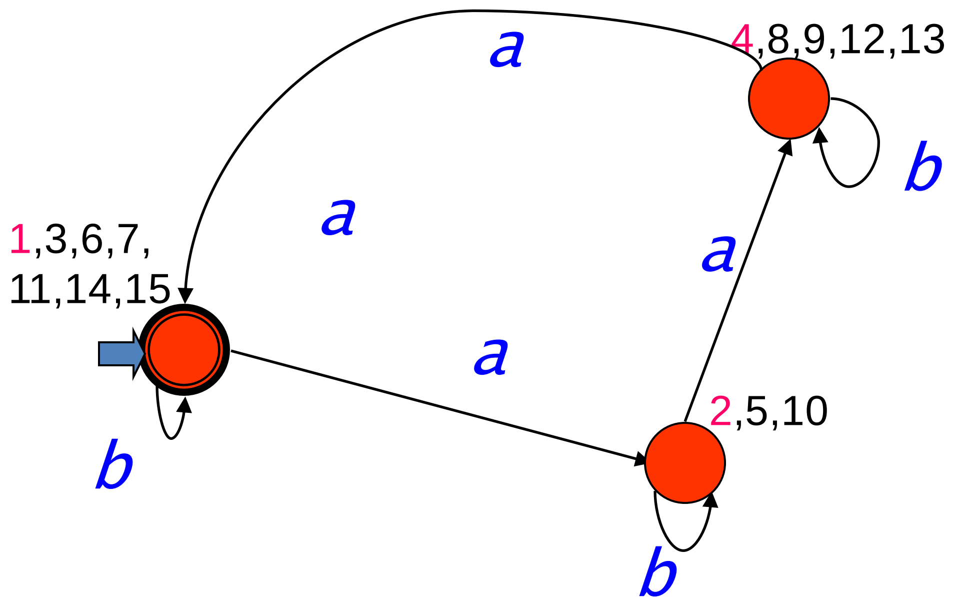
$S_- = \{aa, ab, aaaa, ba\}$

Now try to merge {8,12} with {4,9,13}



S_{-}={*aa*, *ab*, *aaaa*, *ba*}

This is OK and no more merge is possible so the algorithm halts



$S_- = \{aa, ab, aaaa, ba\}$

Properties

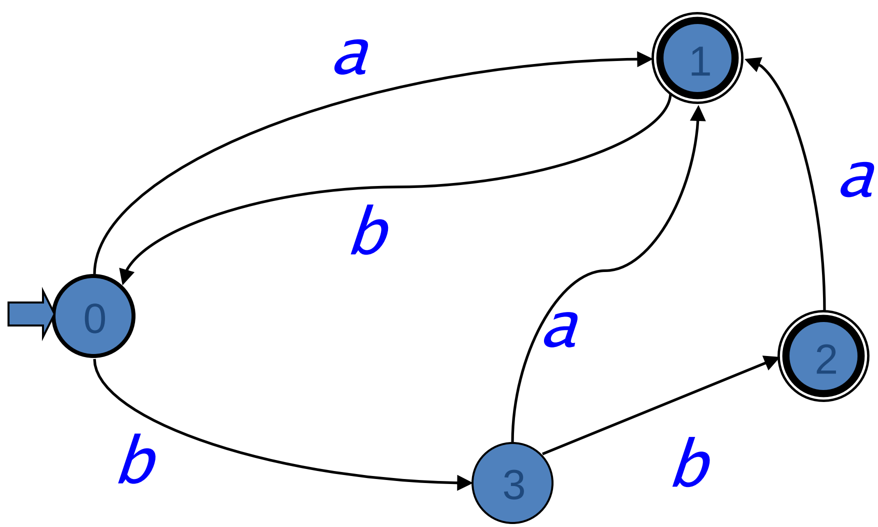
- RPNI identifies any regular language in the limit
- RPNI works in polynomial time. Complexity is in

$$O(|\mathrm{Red}|^2 . |\Sigma| (\|S_+\| + \|S_-\|))$$

- There are many significant variants of RPNI
- RPNI can be extended to other classes of grammars

Exercices

- Run RPNI on
  - $S_+=\{a,bba,bab,aabb\}$
  - $S_-=\{b,ab,baa,baabb\}$
- Find a characteristic sample for:



5. Complexity issues for RPNI

A characteristic sample

- A sample is characteristic (for some algorithm) whenever, when included in the learning sample, the algorithm returns the correct DFA
- The characteristic sample should be of polynomial size
- There is an algorithm which given a DFA builds a characteristic sample for RPNI

Definition : polynomial characteristic sample

$\mathcal{G}$ has polynomial characteristic samples for identification algorithm $\mathbf{a}$ if there exists a polynomial $p()$ such that: given any G in $\mathcal{G}$, $\exists CS$ correct sample for G, such that when $CS \subseteq f_n$, $\mathbf{a}(f_n) \equiv G$ and $\|CS\| \leq p(\|G\|)$

About characteristic samples

- If you add more strings to a characteristic sample it still is characteristic
- There can be many different characteristic samples (EDSM, tree version,…)
- Change the ordering (or the exploring function in RPNI) and the characteristic sample will change

Open problems

- RPNI's complexity is not a tight upper bound. Find the correct complexity
- The definition of the characteristic sample is not tight either. Find a better definition
- Can there be a linear time DFA learner?

Collusion

- Collusion consists in having the learner and the teacher agree of some specific encoding system. Then, the teacher can just pass one string which is the encoding of the target
- Is that cheating?
- Is that learning?

6. Heuristics

6.1 Genetic Algorithms

- The principle: via evolutionary mechanisms, nature increases the quality of its population

- Allow a population of solutions to interact and evolve

Mechanisms (gene level):

- Mutation
- Crossing-over

(a solution is just a string)

Mutation

TTAGCCTTC

TTTGCCTTC

Crossing-over

- TTATCCGT TAGGCTTC

TTATC CGT TAGG CTTC

TTATC CTTC TAGG CGT

TTATCCTTC TAGGCGT

Idea: define the solutions as sequences

- Be able to measure the quality of a solution
- Conceive a first generation
- Define the genetic operations (mutation, crossing over)
- Keep the best elements of the second generation
- Iterate

Genetic algorithms in Grammatical Inference

- (Dupont 94)
  - code the automata (the partition of states of *$PTA(S_+)$*) into partitions
  - define genetic operators
  - define an optimum as an automaton with as few states as possible and rejecting S_-
  - run the genetic algorithm

Structural Mutation

- Select a state from a block and move it to another block
- Example:

{{1,3} {2} {4,5}}

{{1} {2} {3,4,5}}

{{1} {2} {3} {4,5}}

Structural crossover

{1,2}{3,4,5}   {1,3}{4}{2,5}

{1,3}{3,4,5}

{1,2,3}{4,5}

{1,2}{4}{2,5}

{1,2,3}{4}{5}

Group number encoding

Partition

{{1,2,6}{3,7,9,10}{4,8,12}{5}{11}}

is encoded by

(112341232253)

6.2 Tabu search

- (Giordano 96, based on Glover 89)

- General idea: search a space by choosing a point, and going to its best neighbor that is not in the tabu list

$R \leftarrow$ the set of rules of the grammars in the search space

$G \leftarrow$ an initial grammar

$G^* \leftarrow G$ the best solution reached so far

$T \leftarrow \varnothing$ the Tabu list that cannot occur

$k \leftarrow 0$ the iterations counter

While $k \neq kmax$ **do**

select r in $R \backslash T$, such that the addition or deletion of r from G realizes the maximum of val on X

add or delete r from G

if $val(G) > val(G^*)$ **then** $G^* \leftarrow G$

Update T

$k \leftarrow k+1$

Return G^*

- Procedure Update(T, r)

  if $card(T) = n$ **then** delete its last element

  Add r as the first element of T

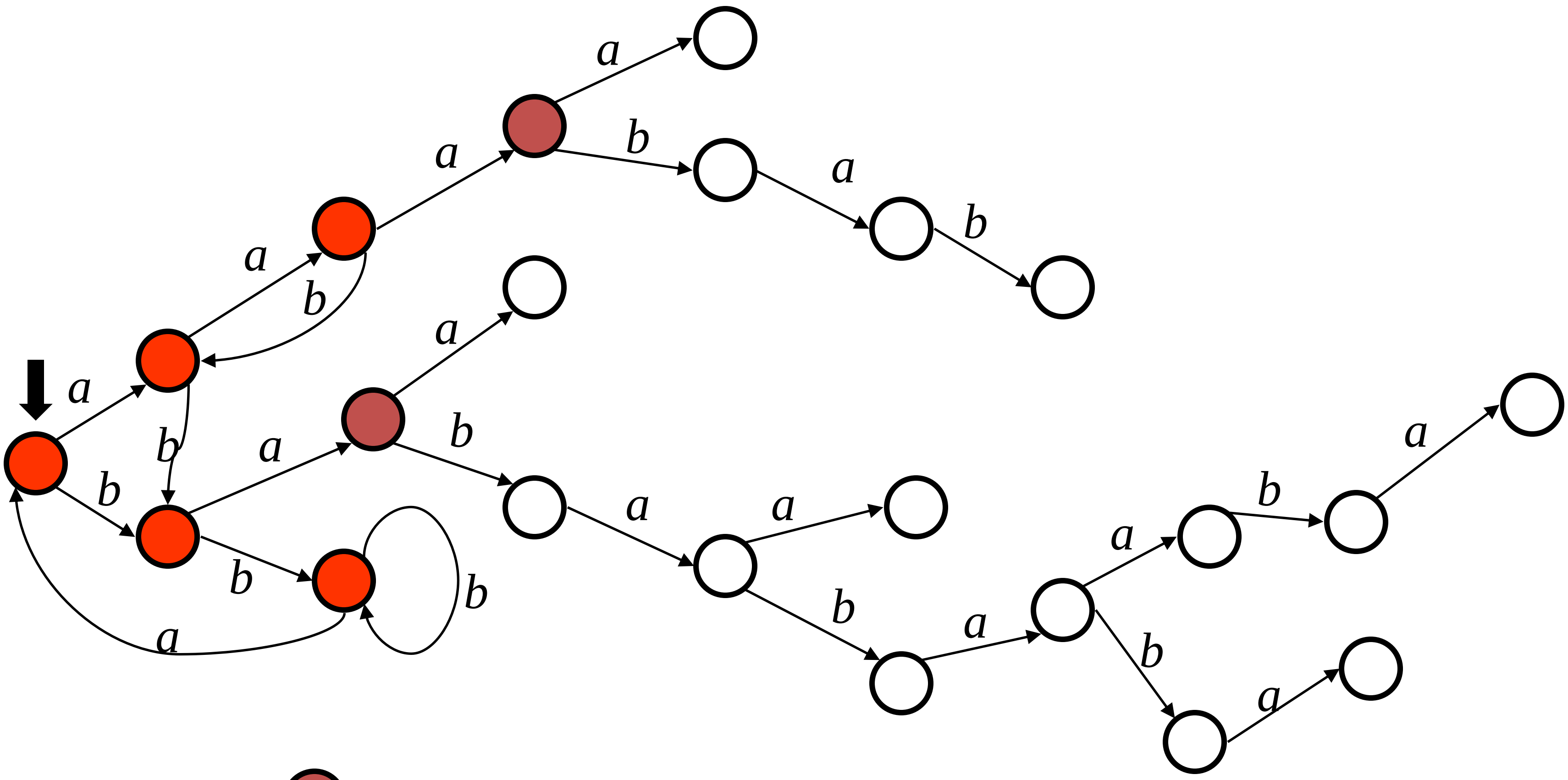
- Tricks
  - **If** *blocked* **then** delete oldest rule
  - *blocked* $\leftarrow$ 6 iterations
  - **if** new G^* **then** empty(T)

6.3 Heuristic greedy State Merging

- RPNI chooses to merge the first 2 states that can be merged
- This is an optimistic view
- There may be another…
- But remember: RPNI identifies in the limit!

How do greedy state merging algorithms work?

- choose two states
  - perform a cascade of forced merges to get a deterministic automaton
  - if it accepts sentences of *S*-, backtrack and choose another couple
  - if not, loop until no merging is still possible



The blue fringe (Lang 98)

What moves are allowed?

- Merging a with a
- Promoting a to and all its successors that are not to

- Promotion:
- when a can be merged with no

What if there are many merges possible?

- Heuristics
- compute a score
- choose highest score

Evidence driven (Lang 98)

for each possible pair (,) **do**

parse S_+ and S_- on *A resulting from the merge*

assign a score to each state of *A* according to the sentences that they accept

if there is a conflict: $-\infty$

else the number of sentences “merged”

sum over all states $\Rightarrow$ the score of the merge

if there is a such that all pairs (,) have score $-\infty$ **then** promote this

select the merge with the highest score

Data driven (cdlh, Oncina & Vidal 96)

For every 🔵 or 🔴 state in *A* count

$$\boxed{\begin{aligned} v_+(q) &= \sum\nolimits_{w\in S_+} \left|\{u\in \mathrm{Pr}ef(w) : \delta(q_0,u)=q\}\right| \\ v_-(q) &= \sum\nolimits_{w\in S_-} \left|\{u\in \mathrm{Pr}ef(w) : \delta(q_0,u)=q\}\right| \end{aligned}}$$

Choose the pair (🔵, 🔴) such that

$$min(v_+(\text{🔵}),v_+(\text{🔴}))+min(v_-(\text{🔵}),v_-(\text{🔴}))$$

is maximal

Careful

- Count first...

  ...then try to merge

- Keep track of all tries
- if some is not mergeable, promote it!

Main differences

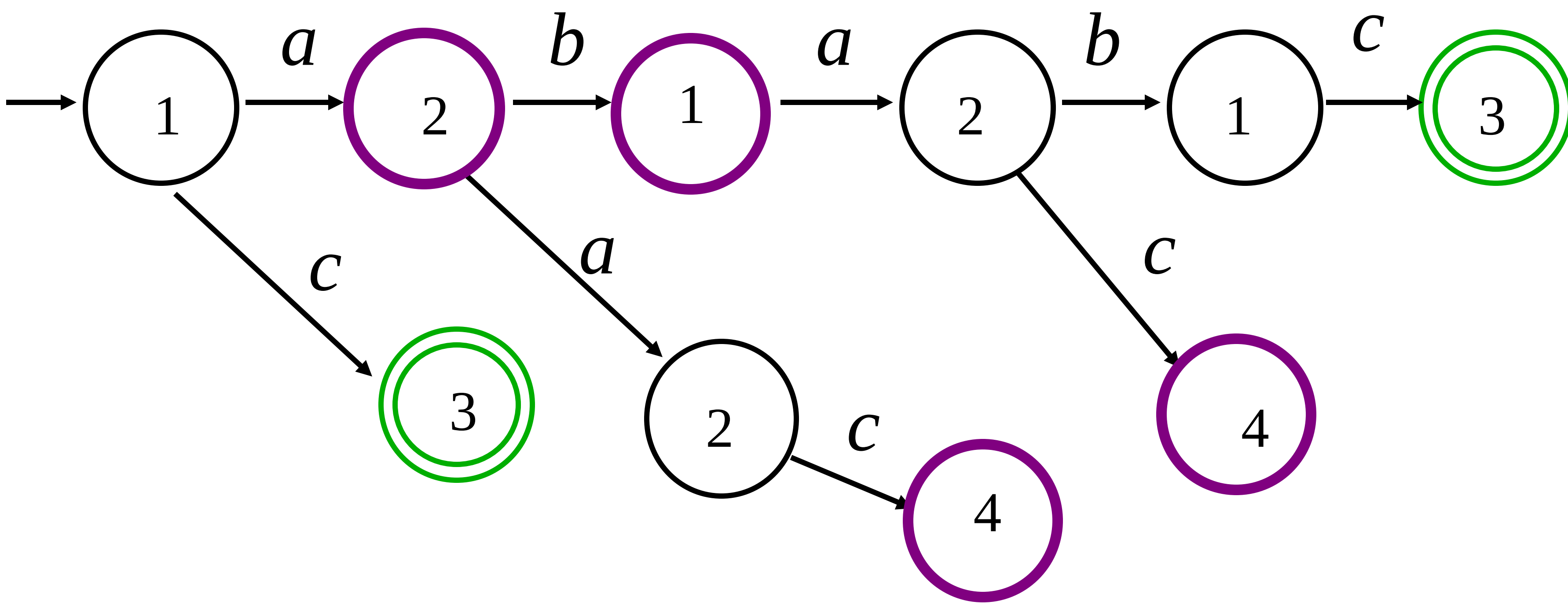
- data driven is cheaper
- evidence driven won Abbadingo competition

- In the stochastic case, it seems that data driven is a good option…

6.4 Constraint Satisfaction

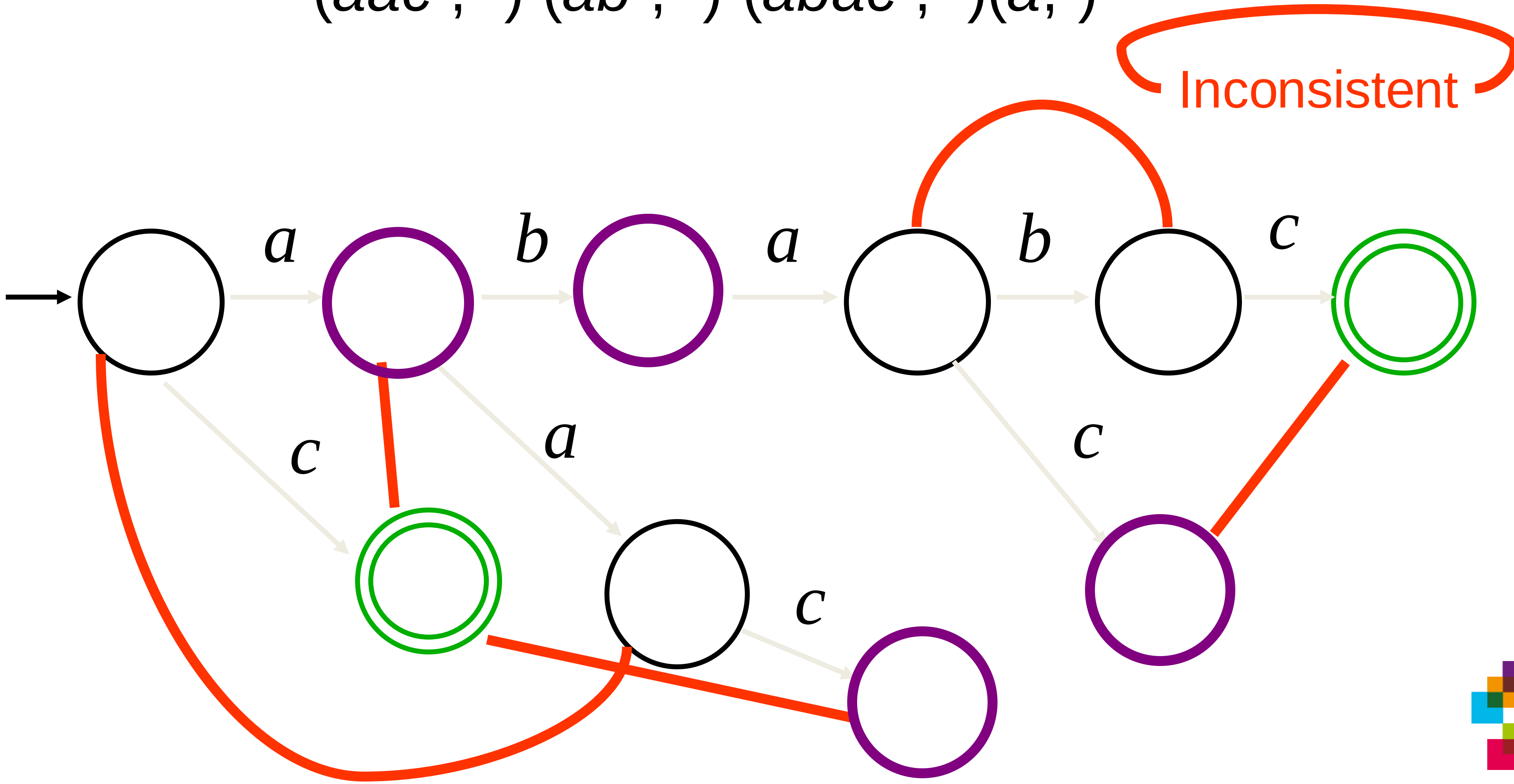
PTA

(*ababc* , +) (*c* , +)
(*aac* , -) (*ab* , -) (*abac* , -)(*a*,-)



(*ababc* , +) (*c* , +)

(*aac* , -) (*ab* , -) (*abac* , -)(*a*,-)



Consider (*Q*, *incompatible*)

- All you have to do is find a maximum clique…
- Another NP-hard problem, but for which good heuristics exist
- Careful : the maximum clique only gives you a lower bound…

Alternatively

- You have $|Q|$ variables $S_1..S_{|Q|}$, and n values $1..n$.
- You have constraints

$$S_i \neq S_j$$

$$\text{or} \quad S_i = S_j \Rightarrow S_k = S_l$$

Solve

Biermann 72, Oliveira & Silva 98, Coste & Nicolas 98, Verwer 2012

7. Open questions and conclusions

Other versions

- A Matlab version of RPNI

http://www.sec.in.tum.de/~hasan/matlab/gi_toolbox/1.0-Beta/

- A JAVA version

http://pagesperso.lina.univ-nantes.fr/~cdlh/Downloads/RPNI.tar.gz

- A parallel version exists, and also an OCAML, C, C++…

Some open questions

- Do better than EDSM (still some unsolved Abbadingo task out there…)
- Write a $O(\| f(n) \|)$ algorithm which identifies DFA in the limit (Jose Oncina and cdlh have a log factor still in the way)
- Identify and study the collusion issues
- Deal with noise.