LABORATOIRE D'INFORMATIQUE
DE NANTES ATLANTIQUE

UMR CNRS 6241
Université de Nantes
Ecole des Mines de Nantes

Learning from Text

Colin de la Higuera
University of Nantes

PASCAL
Pattern Analysis, Statistical Modelling and Computational Learning

dépasser les frontières

Région
PAYS DE LA LOIRE

Acknowledgements

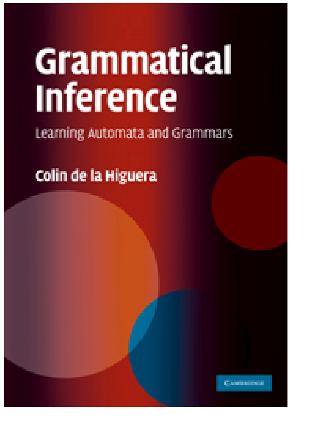
- Laurent Miclet, Jose Oncina, Tim Oates, Anne-Muriel Arigon, Leo Becerra-Bonache, Rafael Carrasco, Paco Casacuberta, Pierre Dupont, Rémi Eyraud, Philippe Ezequel, Henning Fernau, Jean-Christophe Janodet, Satoshi Kobayachi, Thierry Murgue, Frédéric Tantini, Franck Thollard, Enrique Vidal, Menno van Zaanen,...

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

http://videolectures.net/colin_de_la_higuera/

Outline

1. Motivations, definition and difficulties
2. Some negative results
3. Learning *k*-testable languages from text
4. Learning *k*-reversible languages from text
5. Conclusions



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

Chapters 8 and 11

1 Identification in the limit

A class of languages

$$\begin{tikzcd} \mathcal{L} & & \mathcal{P}res \subseteq \mathbb{N} \to X \arrow[ll, dashed, "yields"'] \arrow[dl, "\mathbf{a}"] \\ & \mathcal{G} \arrow[ul, "\mathbf{L}"] & \end{tikzcd}$$

The naming function

A learner

A class of grammars

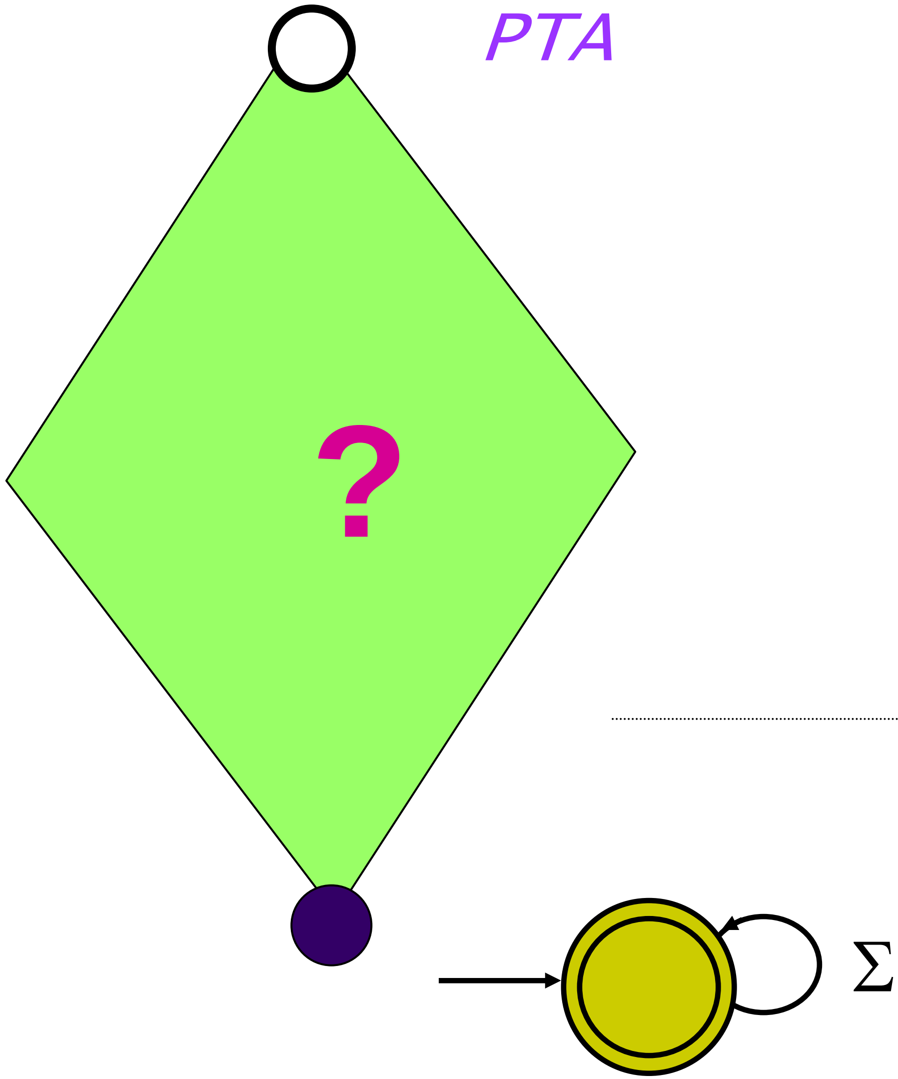
$$\mathbf{L}(\mathbf{a}(\varphi)) = yields(\varphi)$$

$$\varphi(\mathbb{N}) = \psi(\mathbb{N}) \Rightarrow yields(\varphi) = yields(\psi)$$

Learning from text

- Only positive examples are available
- Danger of over-generalization: why not return Σ^*?
- The problem is "basic":
  - Negative examples might not be available
  - Or they might be heavily biased: near-misses, absurd examples…
- Base line: all the rest is learning with help

GI as a search problem



Questions?

- Data is unlabelled…
- Is this a clustering problem?
- Is this a problem posed in other settings?

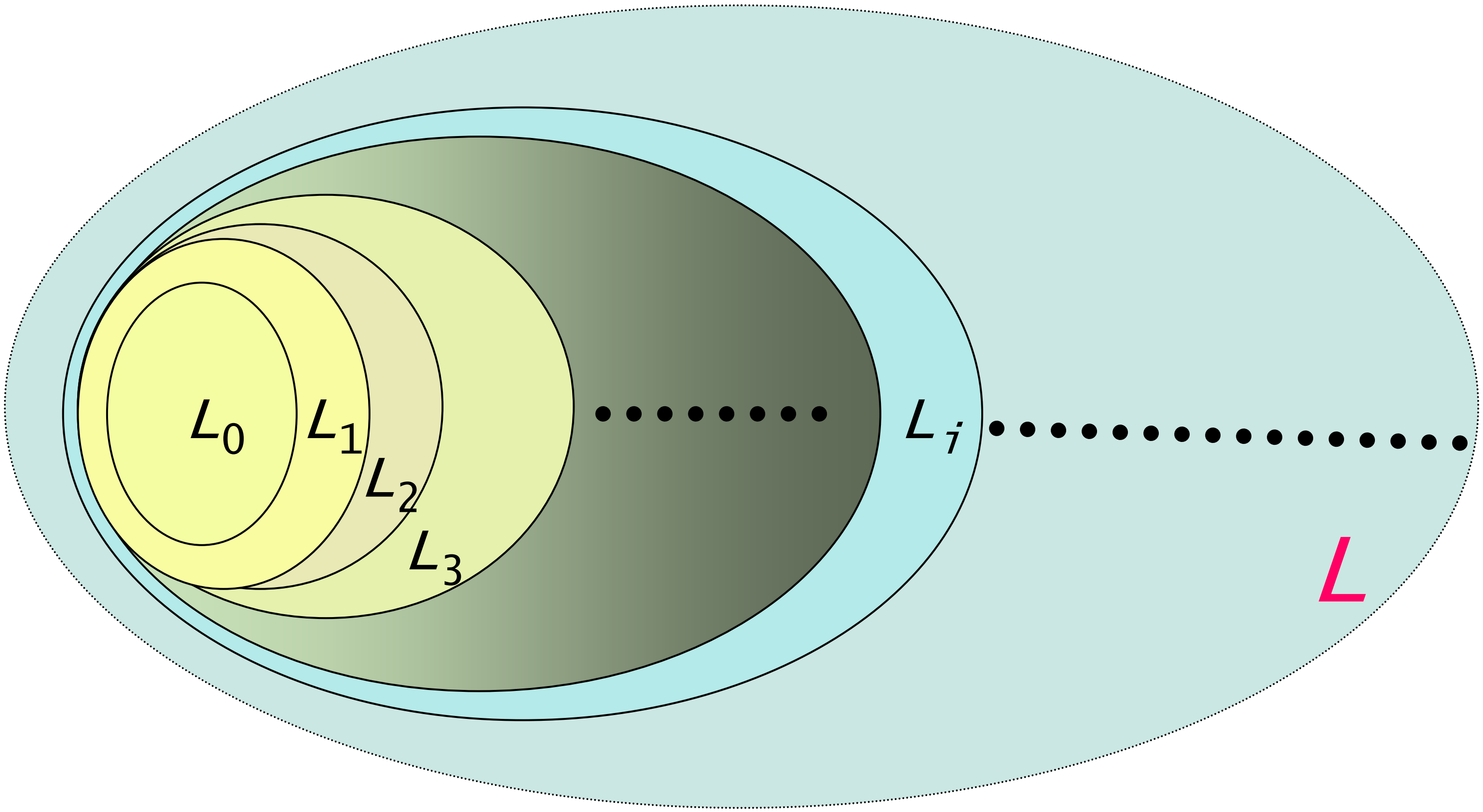
2 The theory

- Gold 67: No super-finite class can be identified from positive examples (or **text**) only
- Necessary and sufficient conditions for learning
- Literature:
  - inductive inference,
  - ALT series, …

Limit point

- A class $\mathcal{L}$ of languages has a limit point *if* there exists an infinite sequence L_n $_{n\in\mathbb{N}}$ of languages in $\mathcal{L}$ such that $L_0 \subset L_1 \subset \ldots L_n \subset \ldots$, and there exists another language $L \in \mathcal{L}$ such that $L = \cup_{n\in\mathbb{N}} L_n$
- L is called a limit point of $\mathcal{L}$

L is a limit point



Theorem

If $\mathcal{L}$ admits a limit point, then $\mathcal{L}$ is not learnable from text

Proof: Let s^i be a presentation in length-lex order for L_i, and s be a presentation in length-lex order for L. Then $\forall n \in \mathbb{N}\ \exists i\ /\ \forall k \leq n$ $s^i_k = s_k$

Note: having a limit point is a sufficient condition for non learnability; not a necessary condition

Mincons classes

- A class is mincons if there is an algorithm which, given a sample S, builds a $G \in \mathcal{G}$ such that $S \subseteq L \subseteq \mathbb{L}(G) \Rightarrow L = \mathbb{L}(G)$

- Ie there is a unique minimum (for inclusion) consistent grammar

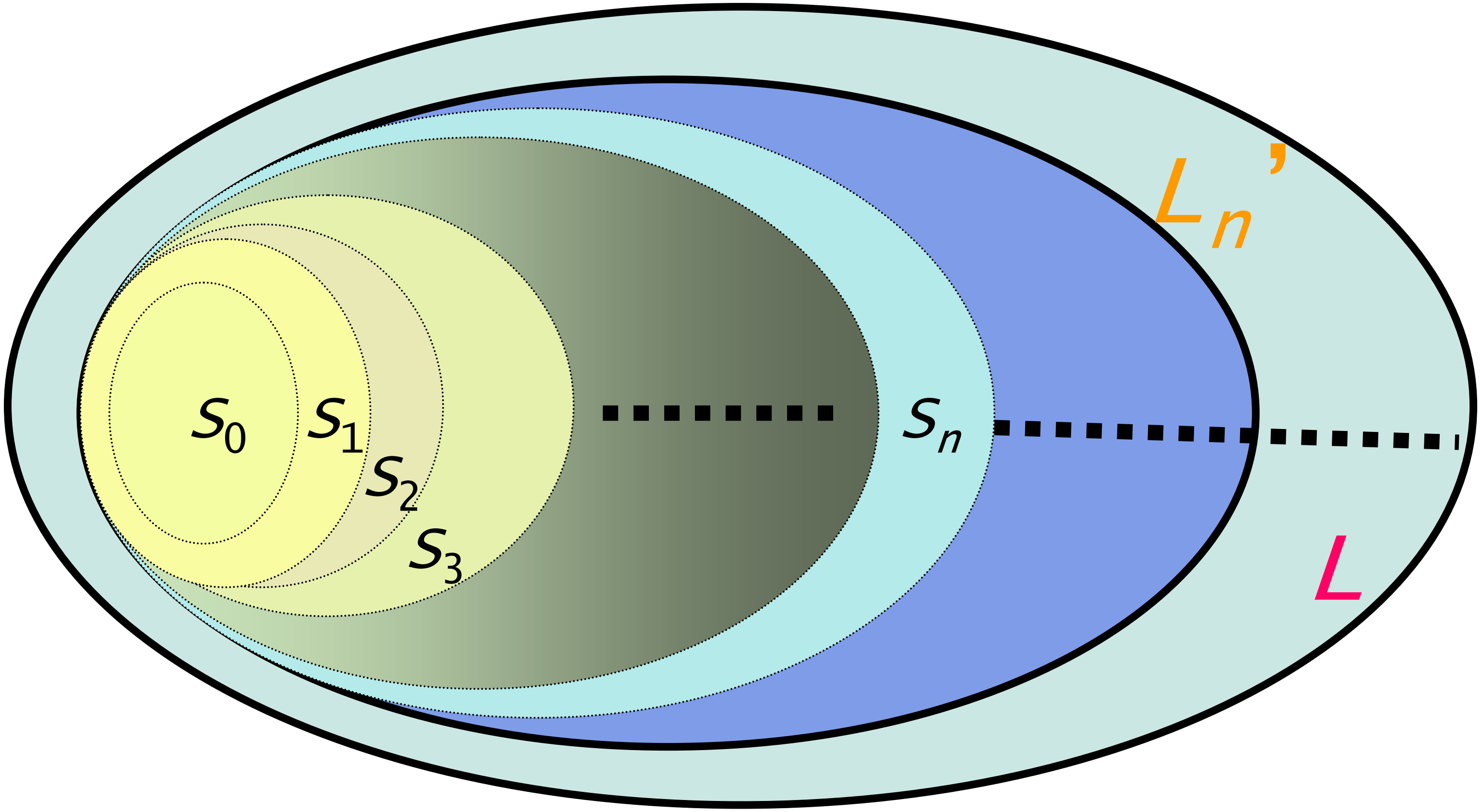
Accumulation point (Kapur 91)

A class $\mathcal{L}$ of languages has an accumulation point *if* there exists an infinite sequence $S_{n\, n\in\mathbb{N}}$ of sets such that $S_0 \subseteq S_1 \subseteq \ldots\ S_n \subseteq \ldots$, and $L = \bigcup_{n\in\mathbb{N}} S_n \in \mathcal{L}$

…and for any $n\in\mathbb{N}$ there exists a language L_n' in $\mathcal{L}$ such that $S_n \subseteq L_n' \subset L$.

The language L is called an accumulation point of $\mathcal{L}$

L is an accumulation point



Theorem (for Mincons classes)

$\mathcal{L}$ admits an accumulation point

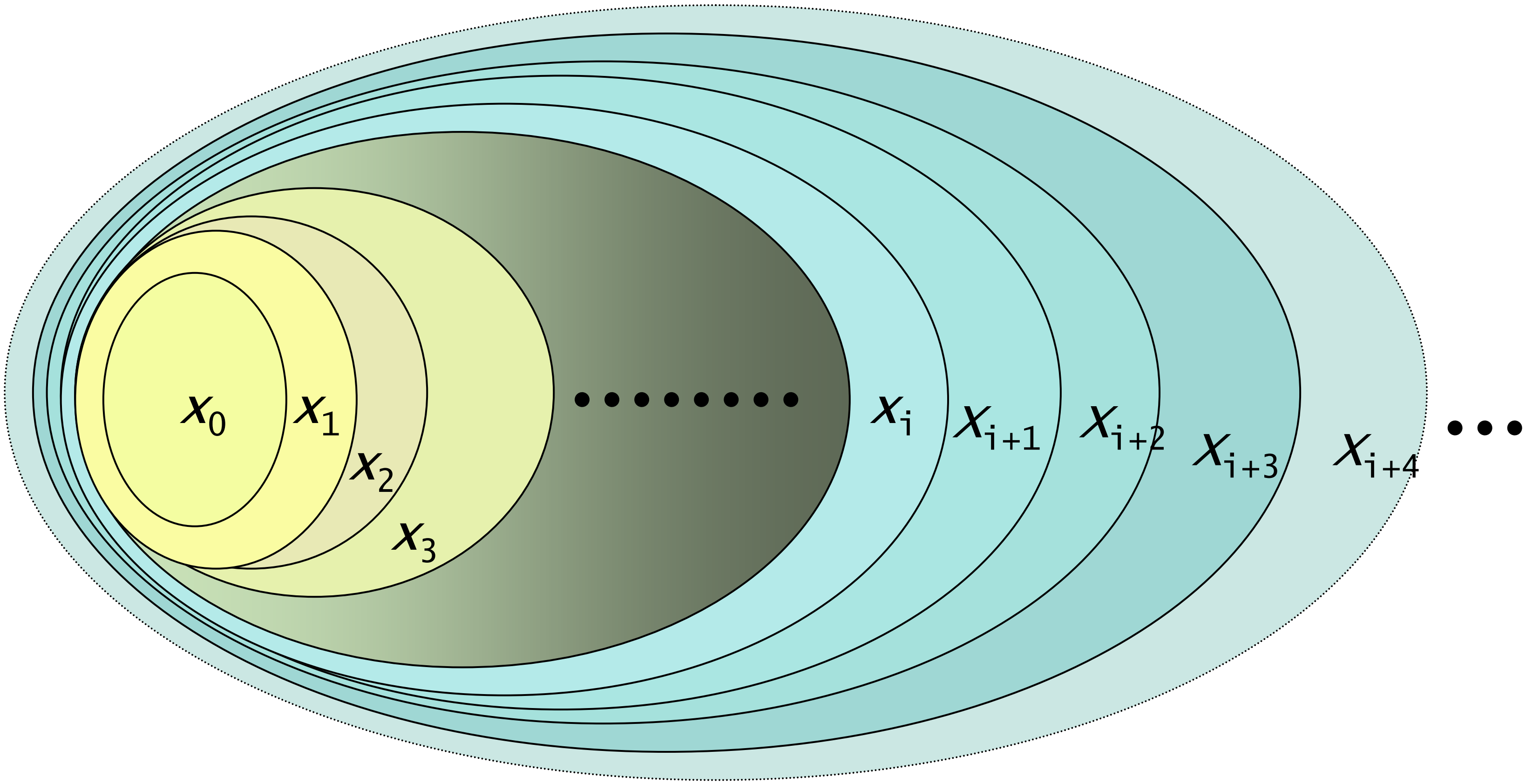
iff

$\mathcal{L}$ is not learnable from text

Infinite Elasticity

- If a class of languages has a limit point there exists an infinite ascending chain of languages $\mathcal{L}_0 \subset \mathcal{L}_1 \subset \ldots \subset \mathcal{L}_n \subset \ldots$.
- This property is called infinite elasticity

Infinite Elasticity



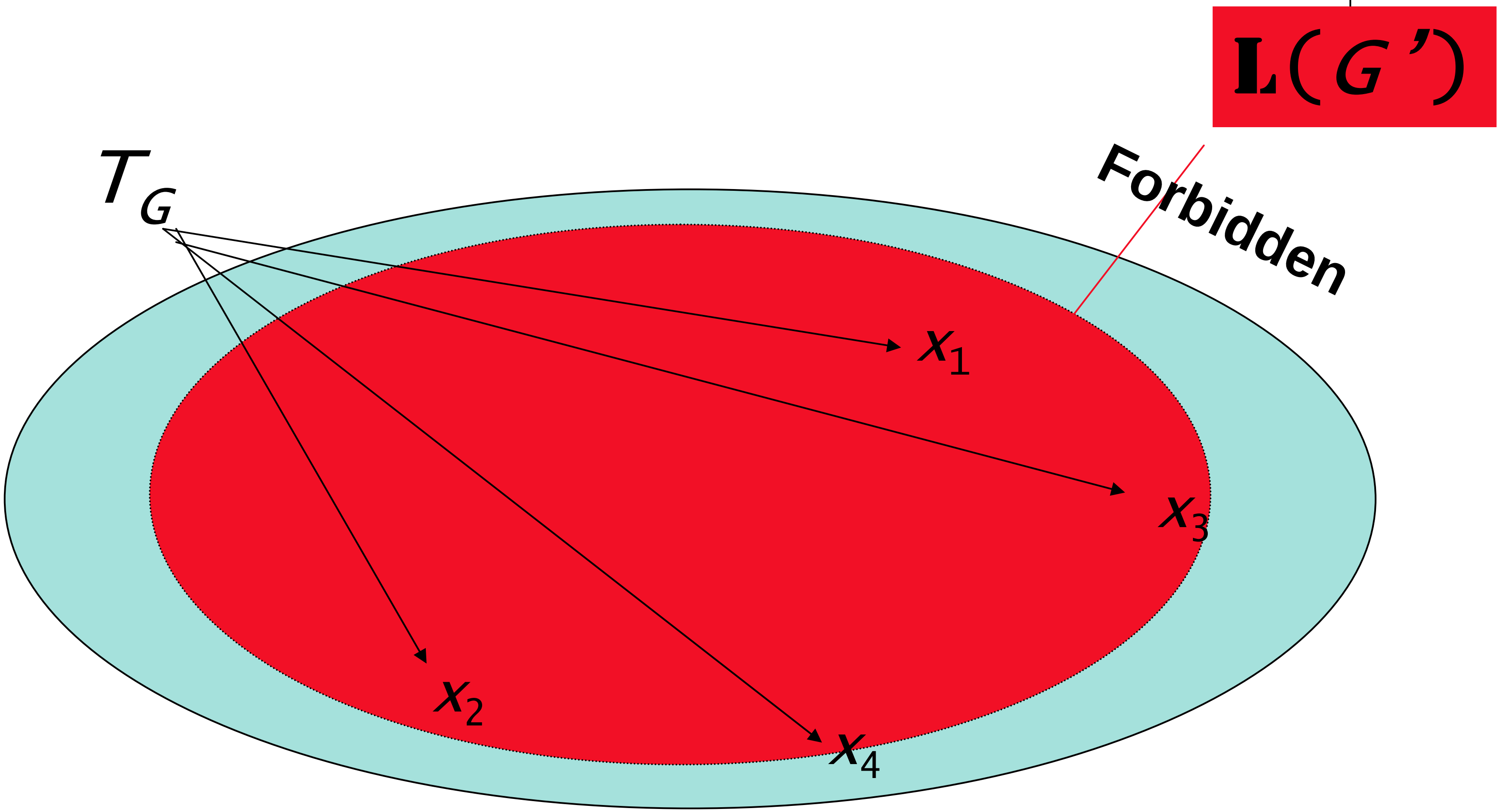
Finite elasticity

$\mathcal{L}$ has *finite elasticity* if it does not have infinite elasticity

Theorem (Wright)

If $\mathcal{L}$ ($\mathcal{G}$) has finite elasticity and is mincons, then $\mathcal{G}$ is learnable.

Tell tale sets



Theorem (Angluin)

$\mathcal{G}$ is learnable *iff* there is a computable partial function ψ: $\mathcal{G} \times \mathbb{N} \rightarrow \Sigma^*$ such that:

1) $\forall n \in \mathbb{N}$, $\psi(G,n)$ is defined *iff* $G \in \mathcal{G}$ and $\mathbf{L}(G) \neq \emptyset$
2) $\forall G \in \mathcal{G}$, $T_M = \{\psi(G,n): n \in \mathbb{N}\}$ is a finite subset of $\mathbf{L}(G)$ called a *tell-tale* subset
3) $\forall G,G' \in \mathcal{M}$, if $T_M \subseteq \mathbf{L}(G')$ then $\mathbf{L}(G') \not\subset \mathbf{L}(G)$

Proposition (Kapur 91)

A language L in $\mathcal{L}$ has a *tell-tale subset* *iff* L is not an accumulation point.

(for mincons)

Summarizing

- Many alternative ways of proving that identification in the limit is not feasible
- Methodological-philosophical discussion
- We still need practical solutions

3 Learning k-testable languages

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

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 is

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

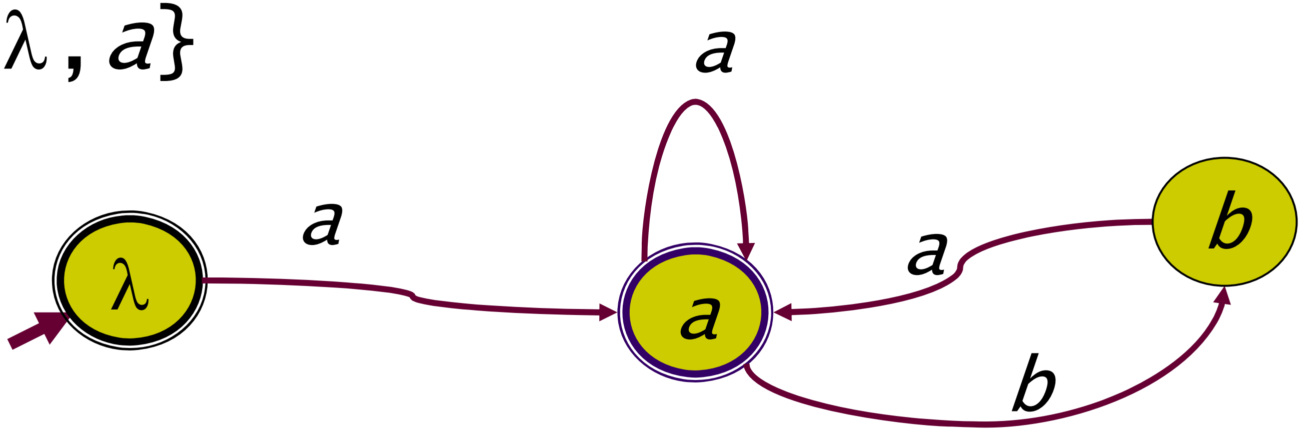
- Strings (of length at least k) have to use a good prefix and a good suffix of length k-1, and all sub-strings have to belong to T. Strings of length less than k should be in C
- Or: $\Sigma^k - T$ defines the prohibited segments
- Key idea: use a window of size k

An example (2-testable)

$I=\{a\}$

$F=\{a\}$

$T=\{aa,\ ab,\ ba\}$
$C=\{\lambda, a\}$



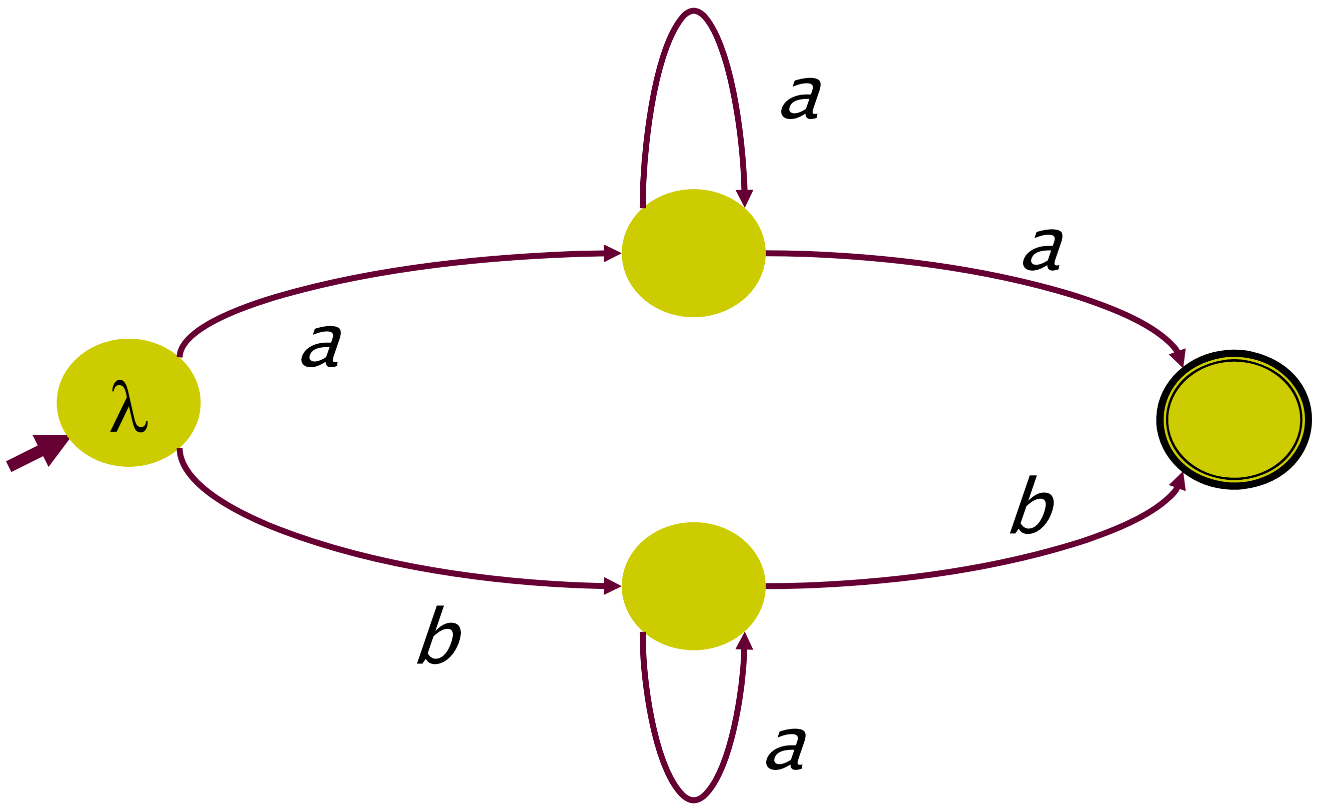
Window language

- By sliding a window of size 2 over a string we can parse
- *ababaaababababaaaab* OK
- *aaabbaaaababab* not OK

The hierarchy of *k-TSS* languages

- $k\text{-}TSS(\Sigma)=\{L\subseteq\Sigma^*: L \text{ is } k\text{-}TSS\}$
- All finite languages are in $k\text{-}TSS(\Sigma)$ if k is large enough!
- $k\text{-}TSS(\Sigma) \subset [k+1]\text{-}TSS(\Sigma)$
- $(ba^k)^* \in [k+1]\text{-}TSS(\Sigma)$
- $(ba^k)^* \notin k\text{-}TSS(\Sigma)$

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 $Z_k = (\Sigma(S), \mathcal{I}(S), \mathcal{F}(S), \mathcal{T}(S), \mathcal{C}(S))$ and

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

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

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

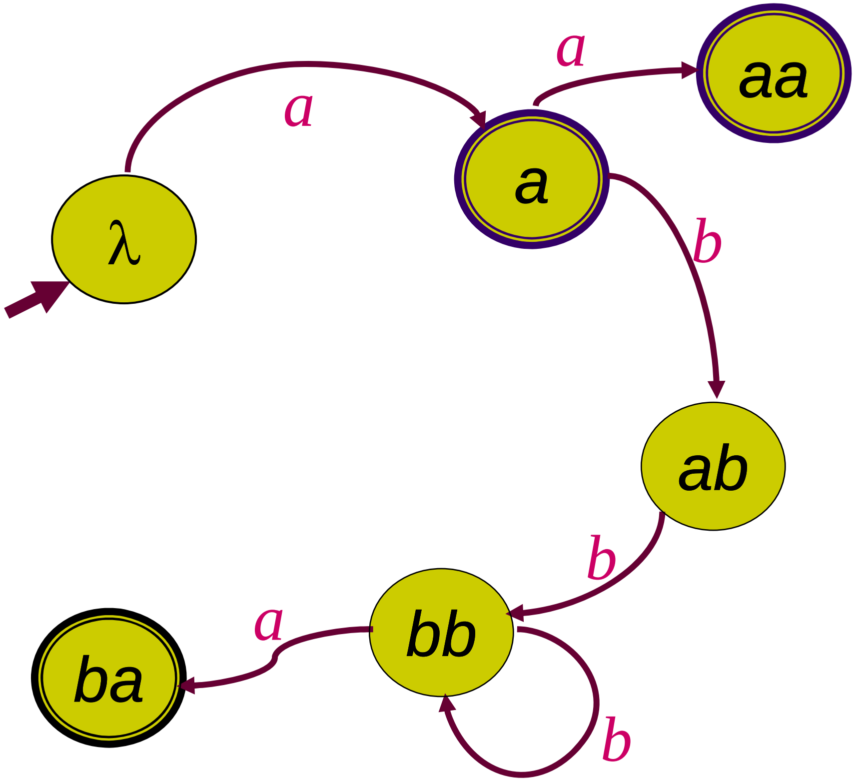
S={*a*, *aa*, *abba*, *abbbba*}

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)

- $\mathbf{L}(\mathbf{a}_{k+1\text{-}TSS}(S)) \subseteq \mathbf{L}(\mathbf{a}_{k\text{-}TSS}(S))$
  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)
- $\forall k > \max_{x \in S} |x|$, $\mathbf{L}(\mathbf{a}_{k\text{-}TSS}(S)) = S$
  - Because for a large k, $T_k(S) = \emptyset$

4 Learning *k*-reversible languages from text

D. Angluin. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29(3):741–765, 1982

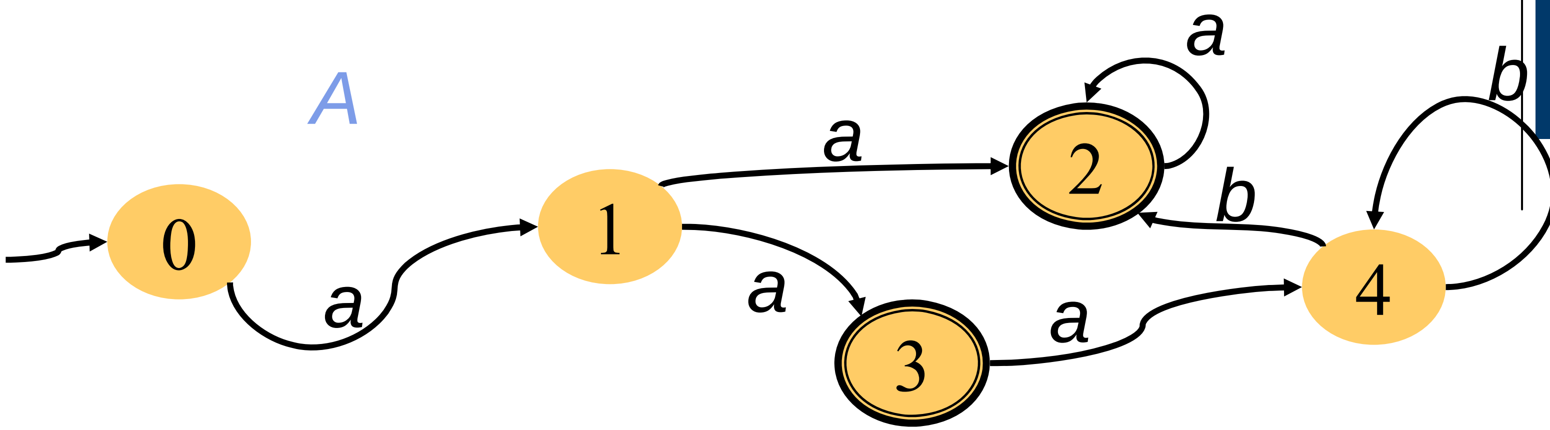
The *k*-reversible languages

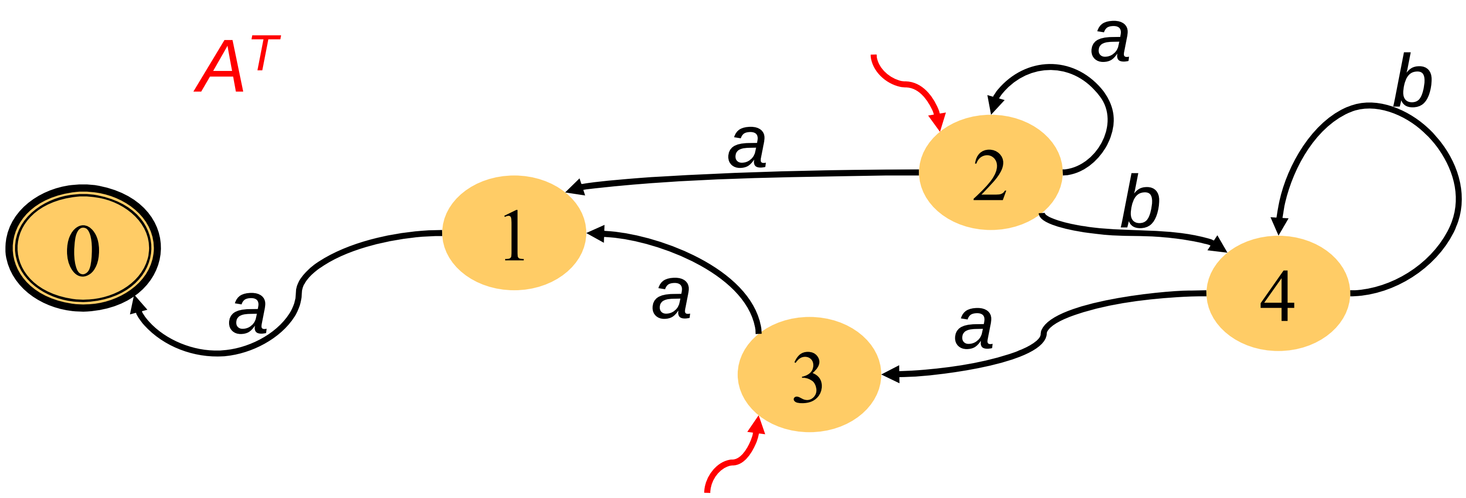
- The class was proposed by Angluin (1982)
- The class is identifiable in the limit from text
- The class is composed by regular languages that can be accepted by a *DFA* such that its reverse is *deterministic with a look-ahead of k*

Let $A=(\Sigma, Q, \delta, I, F)$ be a NFA,

we denote by $A^T=(\Sigma, Q, \delta^T, F, I)$ the reversal automaton with:

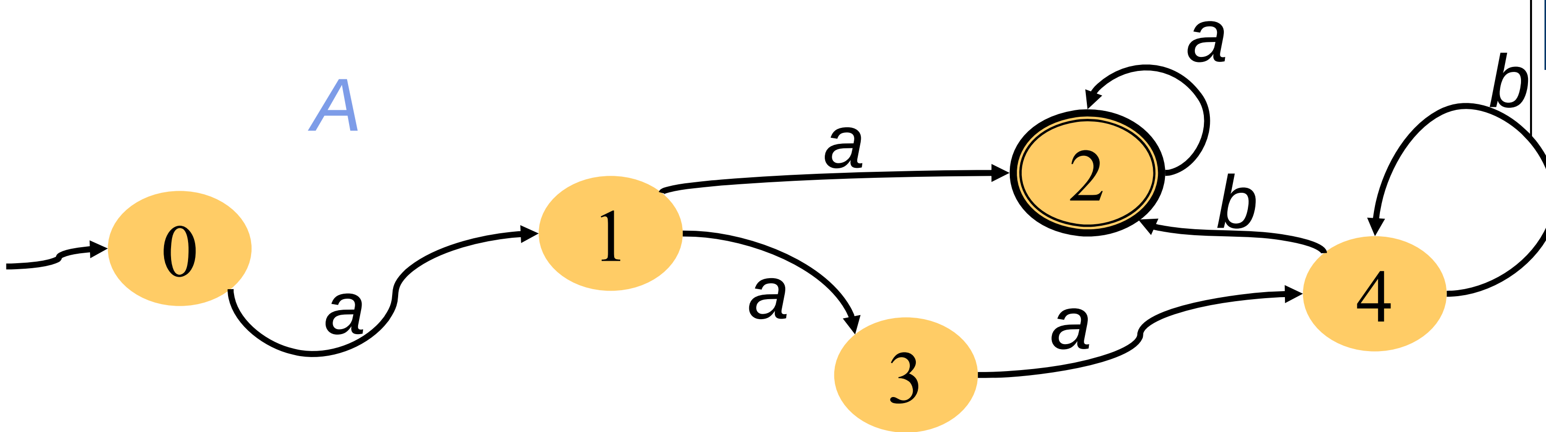
$$\delta^T(q,a)=\{q'\in Q: q\in\delta(q',a)\}$$


A
a
0
1
2
3
4
b


A^T
a
0
1
2
3
4
b

Some definitions

- u is a *k-successor* of q if $|u|=k$ and $\delta(q,u)\neq\emptyset$
- u is a *k-predecessor* of q if $|u|=k$ and $\delta^T(q,u^T)\neq\emptyset$
- λ is 0-successor and 0-predecessor of any state



- *aa* is a *2*-successor of 0 and 1 but not of 3
- *a* is a 1-successor of 3
- *aa* is a *2*-predecessor of 3 but not of 1

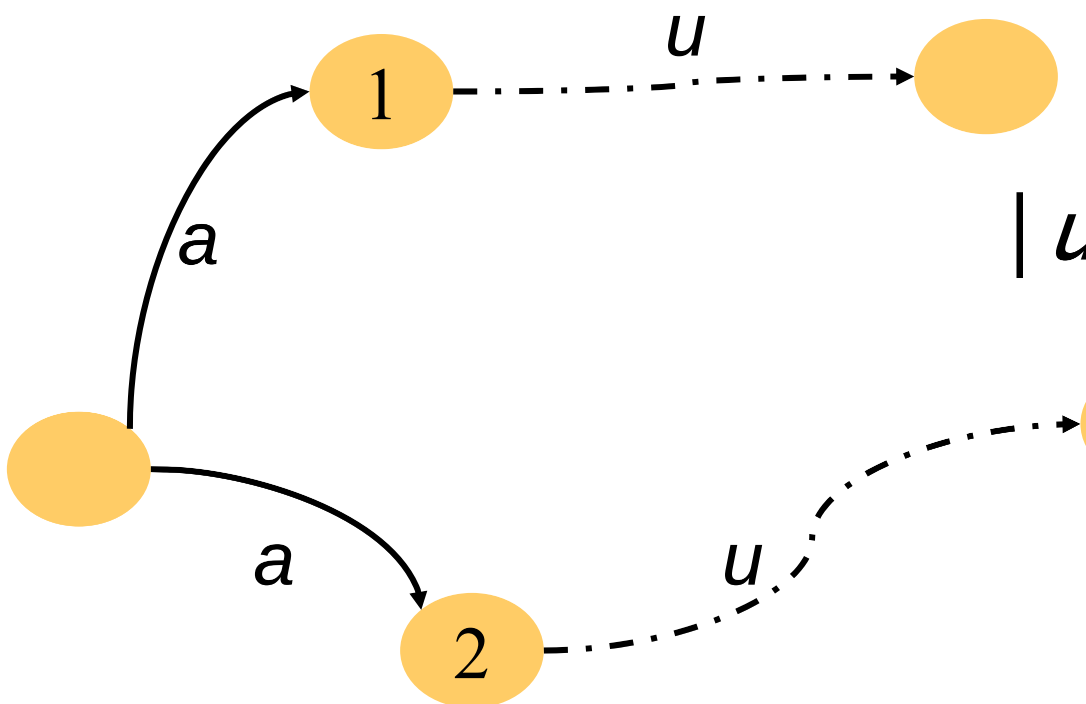
A NFA is deterministic with look-ahead k if $\forall q,q' \in Q$: $q \neq q'$

$$(q,q' \in I) \vee (q,q' \in \delta(q'',a))$$

$$\Rightarrow$$

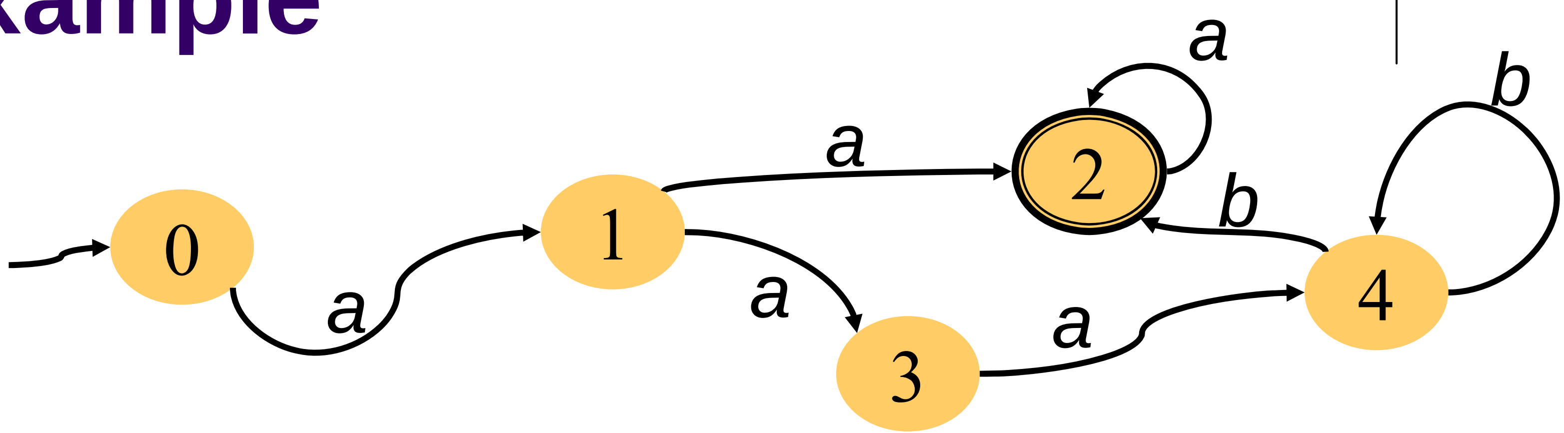
$$\left.\begin{array}{l}(u \text{ is a } k\text{-successor of } q) \wedge \\ (v \text{ is a } k\text{-successor of } q')\end{array}\right\} \Rightarrow \quad u \neq v$$

Prohibited:



$$|u| = k$$

Example

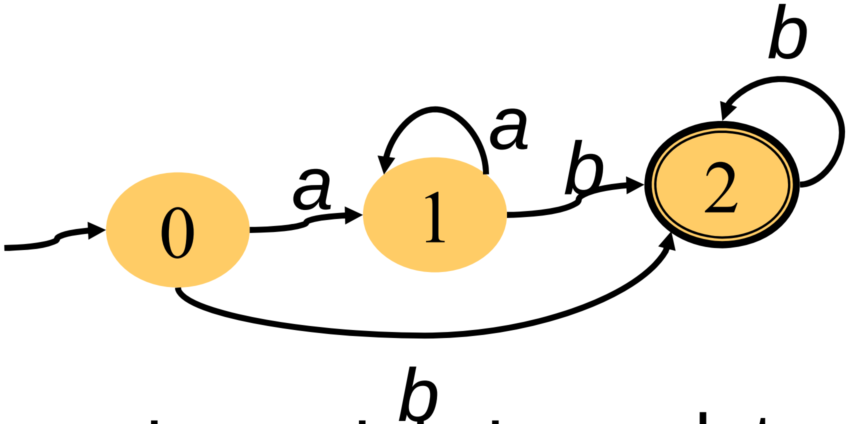


This automaton is not deterministic with look-ahead 1 but is deterministic with look-ahead 2

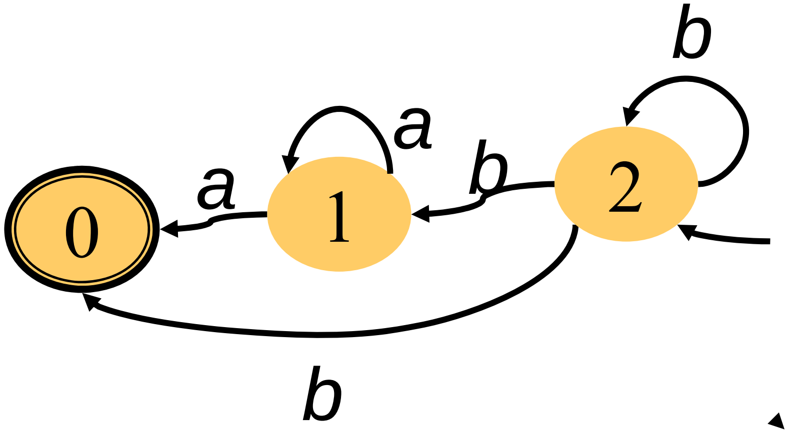
K-reversible automata

- A is k-reversible if A is deterministic and A^T is deterministic with look-ahead k

- Example



deterministic



deterministic with look-ahead 1

Notations

- $\mathcal{RL}(\Sigma, k)$ is the set of all k reversible languages over alphabet Σ
- $\mathcal{RL}(\Sigma)$ is the set of all k-reversible languages over alphabet Σ (ie for all values of k)
- $\mathbf{a}_{k\text{-}RL}$ is the learning algorithm we describe

Properties

- There are some regular languages that are not in $\mathcal{RL}(\Sigma)$
- $\mathcal{RL}(\Sigma,k) \subset \mathcal{RL}(\Sigma,k\text{-}1)$

check

Violation of *k*-reversibility

Two states q, q' violate the k-reversibility condition if

- they violate the deterministic condition: $q,q' \in \delta(q'',a)$

or

- they violate the look-ahead condition:
  - $q,q' \in F$, $\exists u \in \Sigma^k$: u is k-predecessor of both q and q'
  - $\exists u \in \Sigma^k$, $\delta(q,a)=\delta(q',a)$ and u is k-predecessor of both q and q'

Learning k-reversible automata

- Key idea: the order in which the merges are performed does not matter!
- Just merge states that do not comply with the conditions for *k*-reversibility

K-RL algorithm ($\mathbf{a}_{k\text{-}RL}$)

Data: $k \in \mathbb{N}$, S sample of a k-RL language L

A_0=PTA(S)

$\pi = \{\{q\} : q \in Q\}$

While $\exists B, B' \in \pi$ k-reversibility violators do

$\pi = \pi - B - B' \cup \{B \cup B'\}$

$A = A_0 / \pi$

K-RL Algorithm ($\mathbf{a}_{k\text{-}RL}$)

Data: $k \in \mathbb{N}$, S sample of a k-RL language L

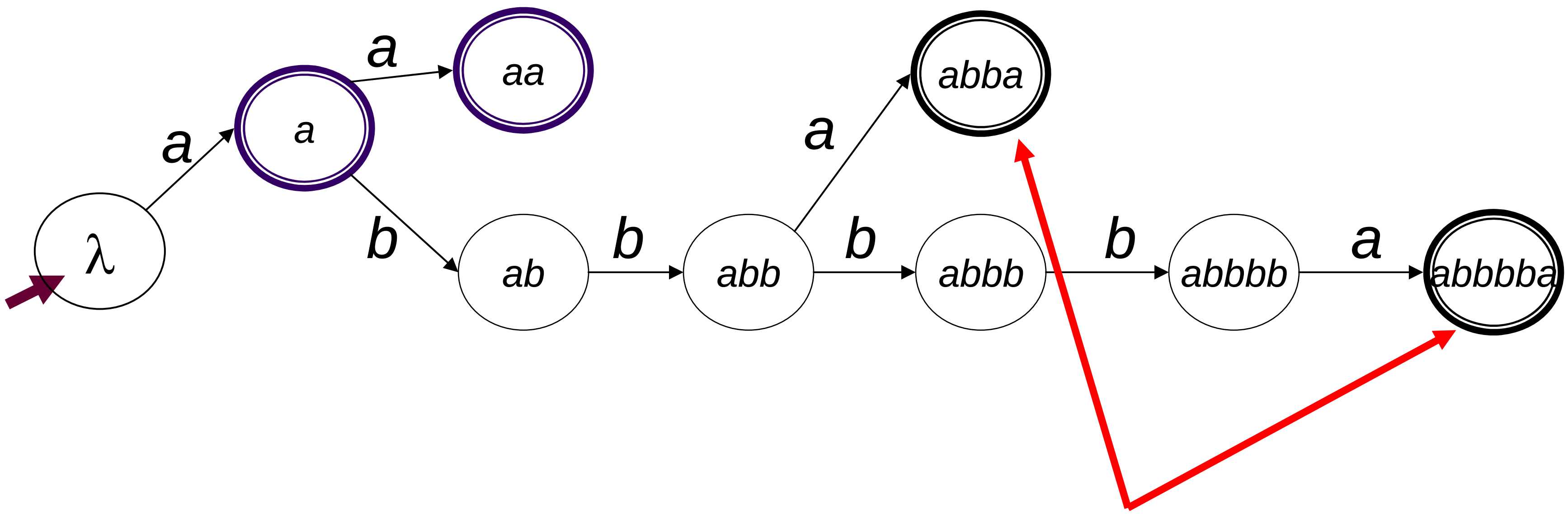
A=PTA(S)

While $\exists q,q'$ k-reversibility violators do

A=merge(A,q,q')

k=2

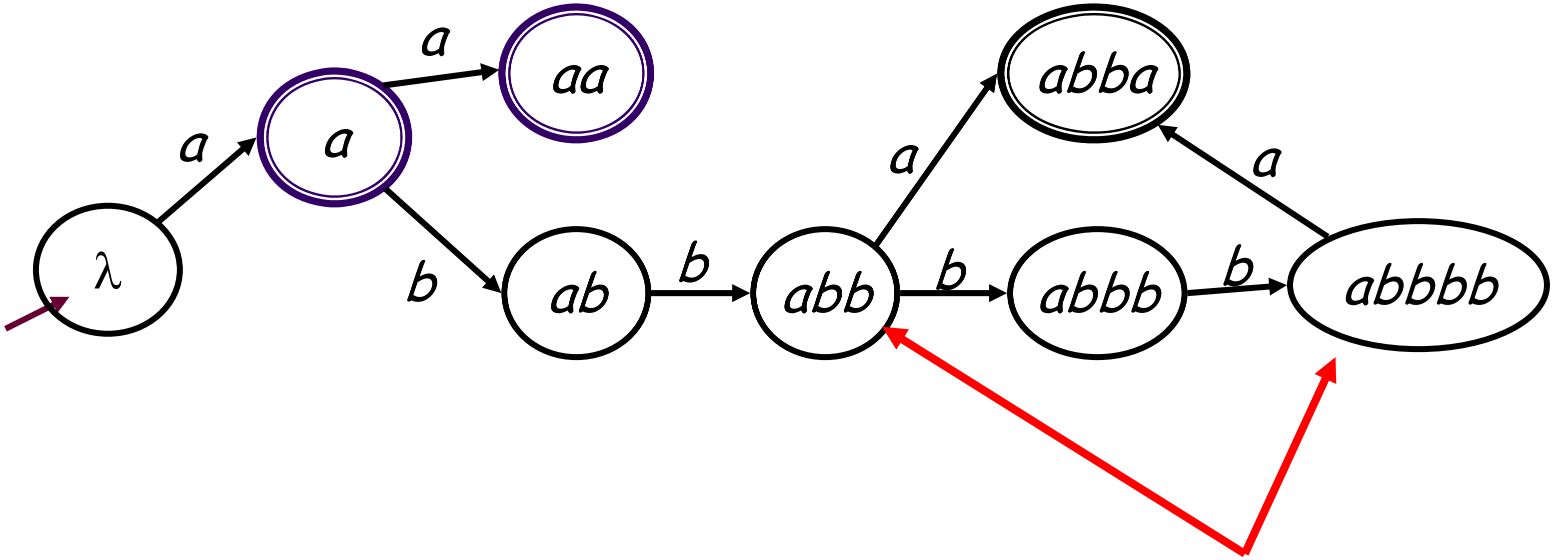
Let *S*={*a*, *aa*, *abba*, *abbbba*}



Violators, for *u*= *ba*

k=2

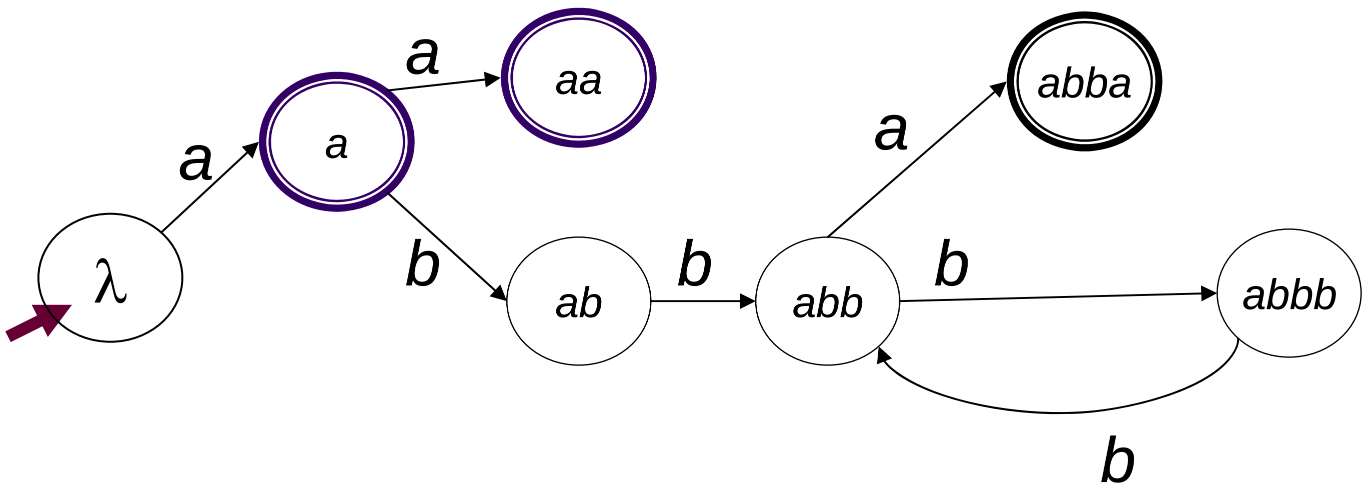
S={a, aa, $abba$, $abbbba$}



Violators, for u= bb

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

k=2



Suppose *k*=1. Then now *a*, *aa* and *abba* violate.

Properties (1)

- $\forall k \geq 0$, $\forall S$, $\mathbf{a}_{k\text{-}RL}(S)$ is a k-reversible language
- $\mathbb{L}(\mathbf{a}_{k\text{-}RL}(S))$ is the smallest k-reversible language that contains S
- The class $\mathcal{RL}(\Sigma, k)$ is identifiable in the limit from text

Properties (2)

- Any regular language is k-reversible iff

$$(u_1v)^{-1}L \cap (u_2v)^{-1}L \neq \emptyset \text{ and } |v|=k$$

$$\Rightarrow$$

$$(u_1v)^{-1}L=(u_2v)^{-1}L$$

(if two strings are prefixes of a string of length at least k, then the strings are Nerode-equivalent)

Properties (3)

- $\mathbf{L}(\mathbf{a}_{k\text{-}RL}(S)) \subset \mathbf{L}(\mathbf{a}_{(k\text{-}1)\text{-}RL}(S))$
- $\mathcal{RL}(\Sigma, k) \subset \mathcal{RL}(\Sigma, k\text{-}1)$ check!

Properties (4)

The time complexity is $O(k\|S\|^3)$

The space complexity is $O(\|S\|)$

Properties (4)
Polynomial aspects

- Polynomial characteristic sets
- Polynomial update time
- But not necessarily a polynomial number of mind changes

Extensions

- Sakakibara built an extension for context-free grammars whose tree language is *k*-reversible
- Marion & Besombes propose an extension to tree languages
- Different authors propose to learn these automata and then estimate the probabilities as an alternative to learning stochastic automata

Exercises

- Build a language L that is not k-reversible, $\forall k \geq 0$
- Prove that the class of all k-reversible languages is not learnable from text
- Run $\mathbf{a}_{k\text{-}RL}$ on $S=\{aa, aba, abb, abaaba, baaba\}$ for $k=0,1,2,3$

Solution (idea)

- $L_k=\{a^i: i\leq k\}$
- Then for each k: L_k is k-reversible but not k-1 reversible.
- And $\bigcup L_k = a^*$
- So there is an accumulation point…

6 Conclusions

- Window languages

Check

Exercise (1)

Check

- Let $J_n=\{w\in\Sigma^*: |w|\leq n\}$
- And $\mathcal{J}=\cup\{J_n\}$
- Find an algorithm that identifies $\mathcal{J}$ in the limit from text
- Prove that this algorithm works in polynomial update time
- Prove that it admits a polynomial locking sequence (characteristic set)
- Prove that the algorithm does not meet Yokomori's conditions

Exercise (2)

- Let $B_{n,w}=\{u\in\Sigma^*: d_{edit}(u,w)\leq n\}$
- And $\mathcal{B}=\cup\{B_{n,w}\}$
- Find an algorithm that identifies $\mathcal{B}$ in the limit from text.
- Does your algorithm meet Yokomori's conditions?

Check