Learning Regular ω-Languages

Dana Angluin and

Dana Fisman

Overview

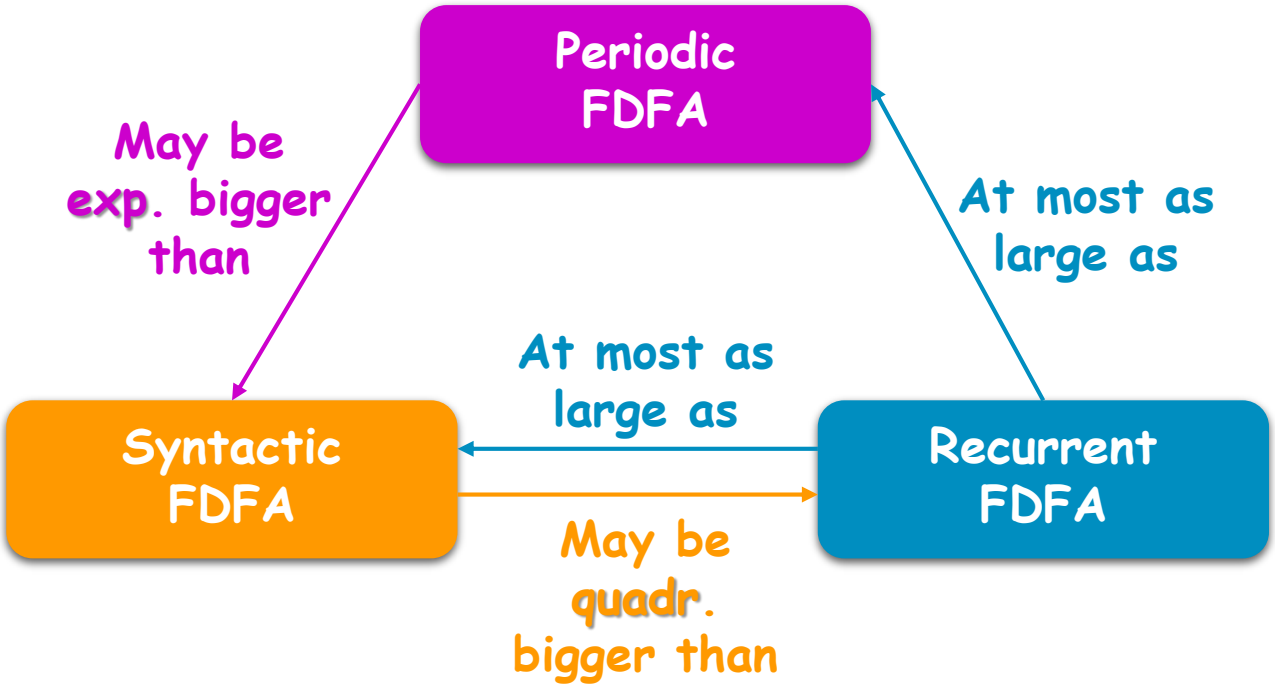
- Motivation
- Background (ω-Automata)
- Previous work on learning regular ω-languages
- Why is it **difficult** to extend **L*** [Angluin] to ω-languages?
  - L* works due to the **Myhill-Nerode Theorem**
  - **No such theorem** exists for traditional **ω-automata**
- Three non-traditional representations for ω-langs

Can actually be learnt using L* itself [CNP93]+ [F. et al. 2008]

Periodic, **Syntactic**, NEW **Recurrent**

NEW Learning Algorithm for all three.

However …



FDFA = Family of DFAs

Motivation

Motivation

D. Fisman

What are ω-Languages?

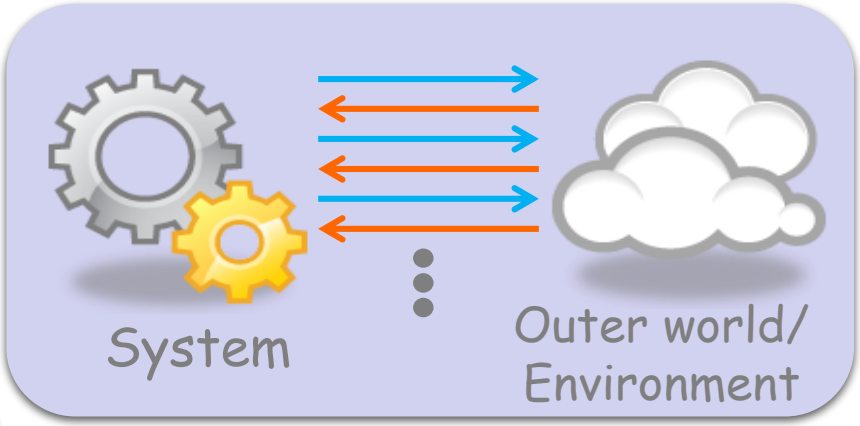
ω-Languages are sets of infinite words (also called ω-words).

What are they good for?

Regular ω-Languages are the main means to model reactive systems.

What are reactive systems ?

Systems with infinite ongoing behavior



Web browser

Cellular phone

ATM machine

Elevator

Operating system

Mail client

A possible computation is an w-word

All possible computations are an w-language

Reasoning about reactive systems

- **Models**:
  - ω-Automata
  - Temporal logic
- **Questions**:
  - Verification
  - Synthesis
- **Practice**:
  - Tools by IBM, Intel, Cadence, Synopsis
  - Users: Intel, Nokia, AMD, TI, Apple…
- **Solving Methods**:
  - Reductions to problems about ω-Automata (intersection, complementation, emptiness, etc.)

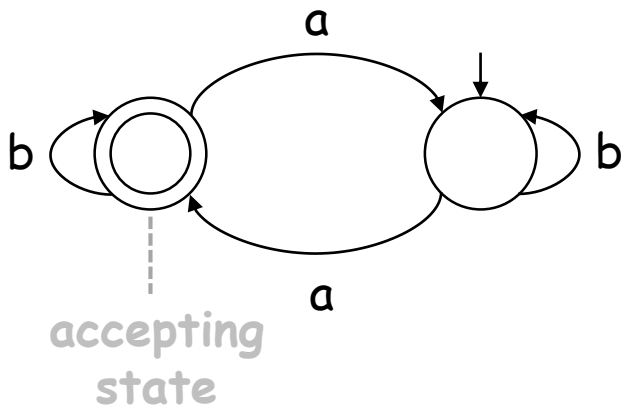
Why learn ω-languages?

- **Synthesis**:
  - One can synthesize a reactive system from a given temporal logic formula
  - However, formulating temporal logic formulas characterizing a system might be difficult
  - Whereas classifying scenarios into good and bad is much easier
- **Compositional Verification**:
  - Current practice uses L*. Thus drastically limiting the expressive power of the obtained assumptions.

ω-Automata

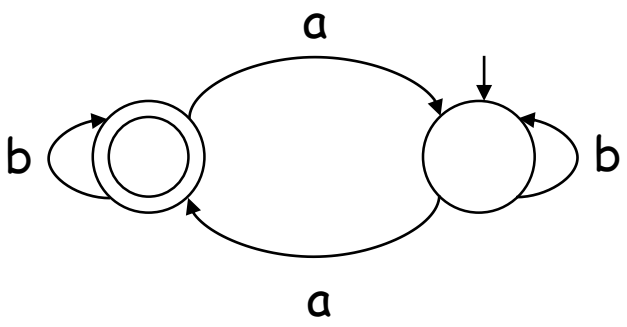
Automata

On words



A word is accepted if it the run on it ends in an accepting state.

On w-words

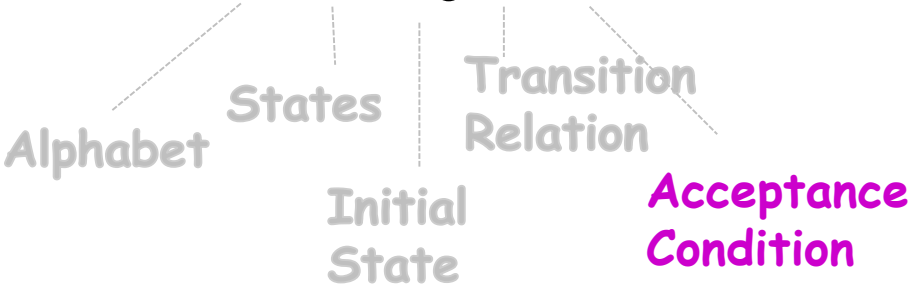


A w-word is accepted if it the run on it ...

???

ω-Automata

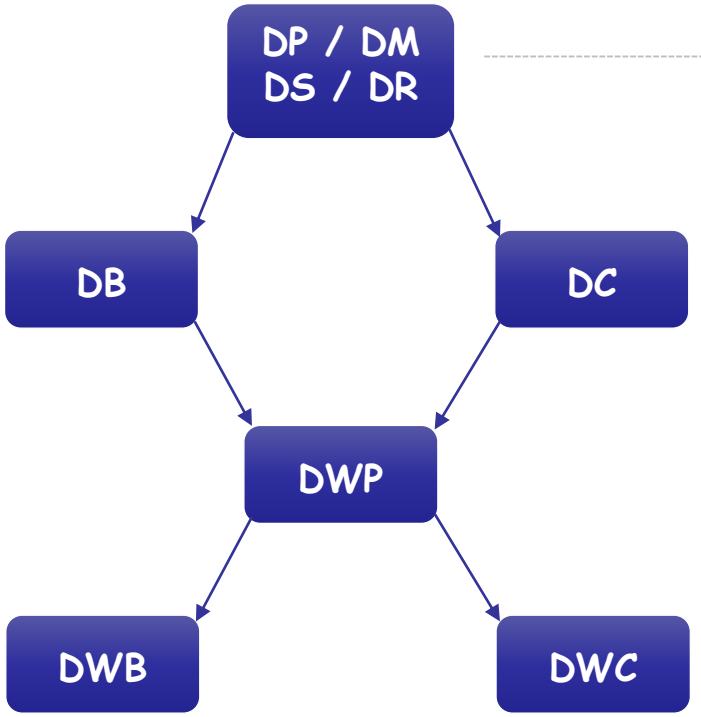
ω-automaton $(\Sigma, S, s_0, \delta, \alpha)$



- There are many ways to define acceptance condition for ω-Automata
  - Büchi
  - co-Büchi
  - Muller
  - Parity
  - Rabin
  - Streett
- Roughly speaking, all are defined using the notion of the states visited infinitely often during a run.

ω-Automata – Expressiveness

- Some acceptance criteria are equally expressive, some are strictly less expressive than others.
- Overall picture looks like this:

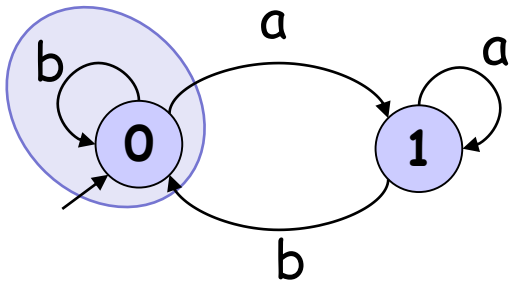


all regular ω-languages

strictly less expressive

Muller Automata (DM)

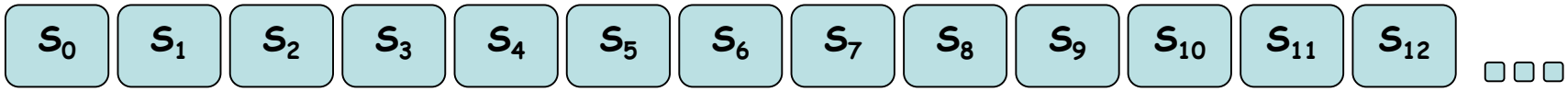
- Among the most expressive is a **Muller** aut.
- A **Muller acceptance condition** is a set of sets of states $\alpha = \{F_1, F_2, \ldots, F_k\}$
- It **accept** a **w-word**, if the **set** of states **visited infinitely often** is exactly one of the F_i 's.



$L = \{ w \mid w \text{ has finitely many } a\text{'s} \}$

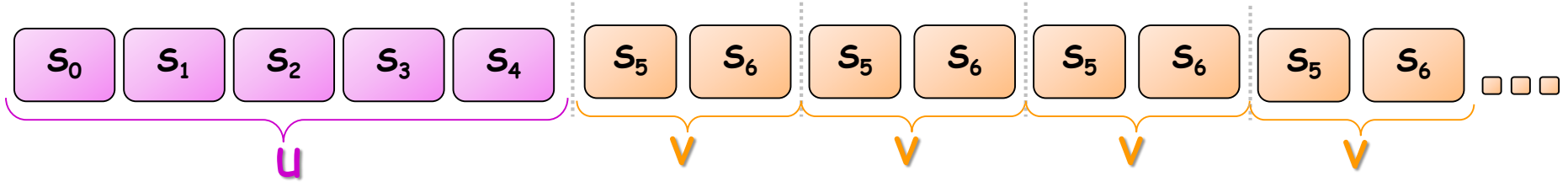
Previous Work on Learning regular ω-Languages

How to cope with ω-words?



- Some **finite** representations is needed.
- Two approaches:
  - Consider only **finite prefixes** of infinite words

  - Use **ultimately periodic words**, i.e. pairs of finite words (u,v) standing for uv^ω

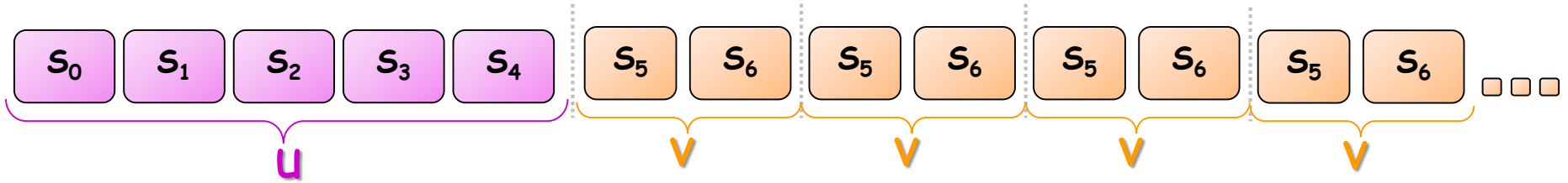


How to cope with ω-words?

D. Fisman

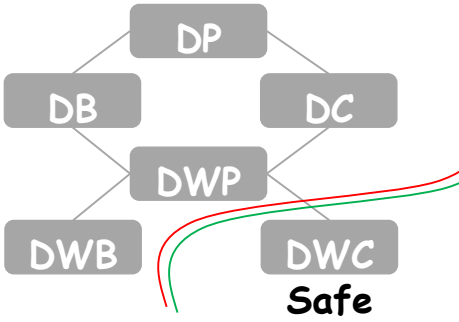
It follows from McNaughton's Theorem that two regular ω-languages are equivalent iff they agree on the set of ultimately periodic words

- Use **ultimately periodic words**, i.e. pairs of finite words (u,v) standing for uv^{ω}



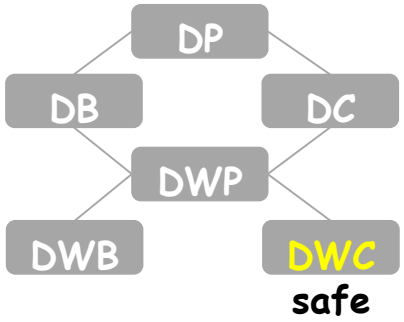
Previous Works (from prefixes)

- de la Higuera and Janodet, 2004
  - positive results for polynomially learning in the limit safe regular ω-languages from prefixes
  - negative results for learning any strictly subsuming class of ω-languages from prefixes.
- Jayasrirani et al., 2012
  - extended this to learning bi-ω-words.

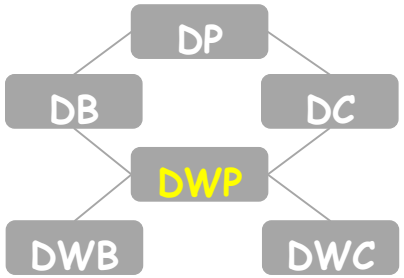


Previous Works (ultimately periodic words)

- Saoudi and Yokomori, 1993
  - Consider **ultimately periodic words**, and provide an algorithm for learning **in the limit** local ω-languages, and safe ω-languages.



- Maler and Pnueli, 1995
  - Provide an **extension to L***, using **ultimately periodic words**, for the class of ω-languages which are recognizable by weak parity automata.

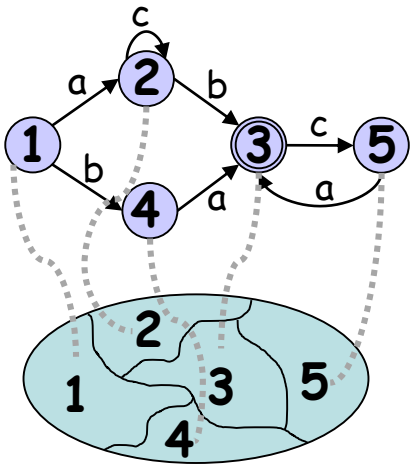


No 1-to-1 correspondence between the syntactic right congruence and ω-automata

Why is it so difficult?

L* works due to the Myhill-Nerode Theorem, stating a 1-to-1 relationship between

- states of the minimal DFA (deterministic finite automaton)
- and the syntactic right congruence ~ of a language.



But for w-langs, the syntactic right congruence is not informative enough!

The Syntactic Right Congruence

For finite words

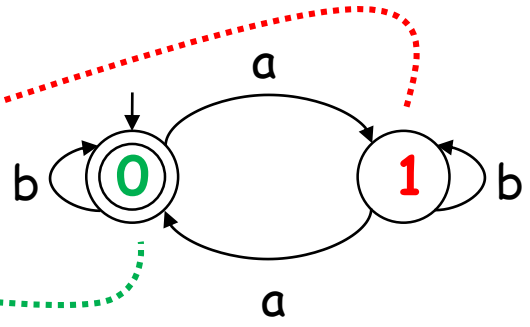
$$x \sim_L y \quad \text{iff} \quad \forall v \in \Sigma^*.\ xv \in L \Leftrightarrow yv \in L$$

Example:

L = { w | w has an even number of a's}

Then $\sim_L$ has two equivalence classes:

- words with an **odd** number of a's and
- words with an **even** number of a's



The Syntactic Right Congruence

For finite words:

$$x \sim_L y \quad \text{iff} \quad \forall v \in \Sigma^*.\ xv \in L \Leftrightarrow yv \in L$$

For ω-words:

$$x \sim_L y \quad \text{iff} \quad \forall u,v \in \Sigma^*.\ xuv^\omega \in L \Leftrightarrow yuv^\omega \in L$$

The Syntactic Right Congruence

For ω-words:

$$x \sim_L y \quad \text{iff} \quad \forall u,v \in \Sigma^*.\ xuv^\omega \in L \Leftrightarrow yuv^\omega \in L$$

Example:

L = { w | w has finitely many a's}

Then $xuv^\omega \in L$ iff uv^ω has finitely may a's.

Regardless of x.

Thus $\sim_L$ has just one equivalence class.

No 1-to-1 Correspondence !

L = { w | w has finitely many a's}

Clearly, a Muller automaton for L needs at least two states.

While $\sim_L$ has only one equivalence class, an ω-automaton for L requires at least two states

A non-traditional representation for w-languages for which the desired 1-to-1 correspondence exists

Family of Right Congruences

- In the quest for a representation for ω-langs for which a Myhill-Nerode THM exists [Maler and Staiger 1997] introduced

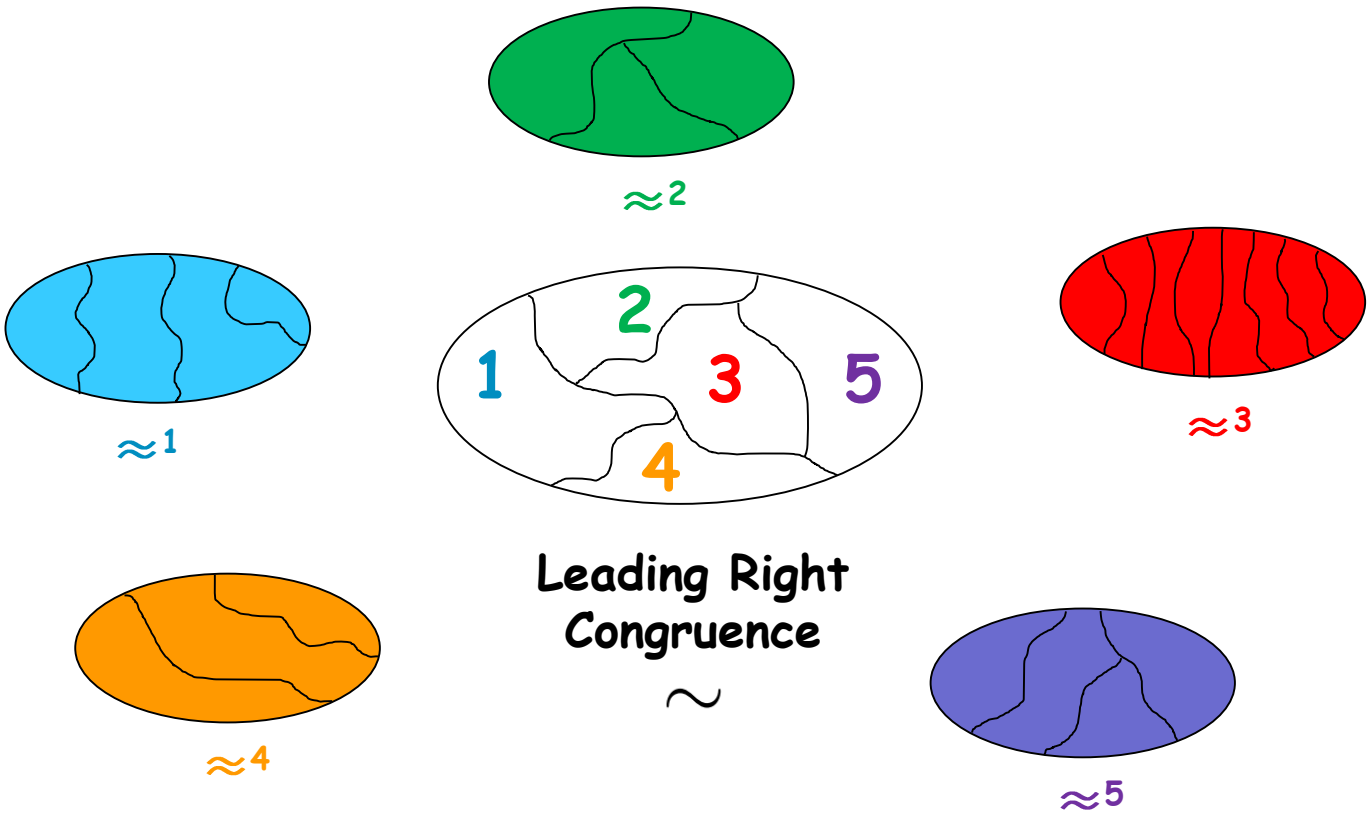
  Family of Right Congruences (FORC)

- **Idea:**
  - Consider only ultimately periodic words uv^{ω}
  - Assume they are given as pairs (u,v)
  - Have a main/leading right congruence for the prefixes
  - And for every class of possibilities a right congruence for the allowed periods

Family of Right Congruences [MS97]

$(\sim, \approx_1, \approx_2, \approx_3, \approx_4, \approx_5)$

Leading — Progress

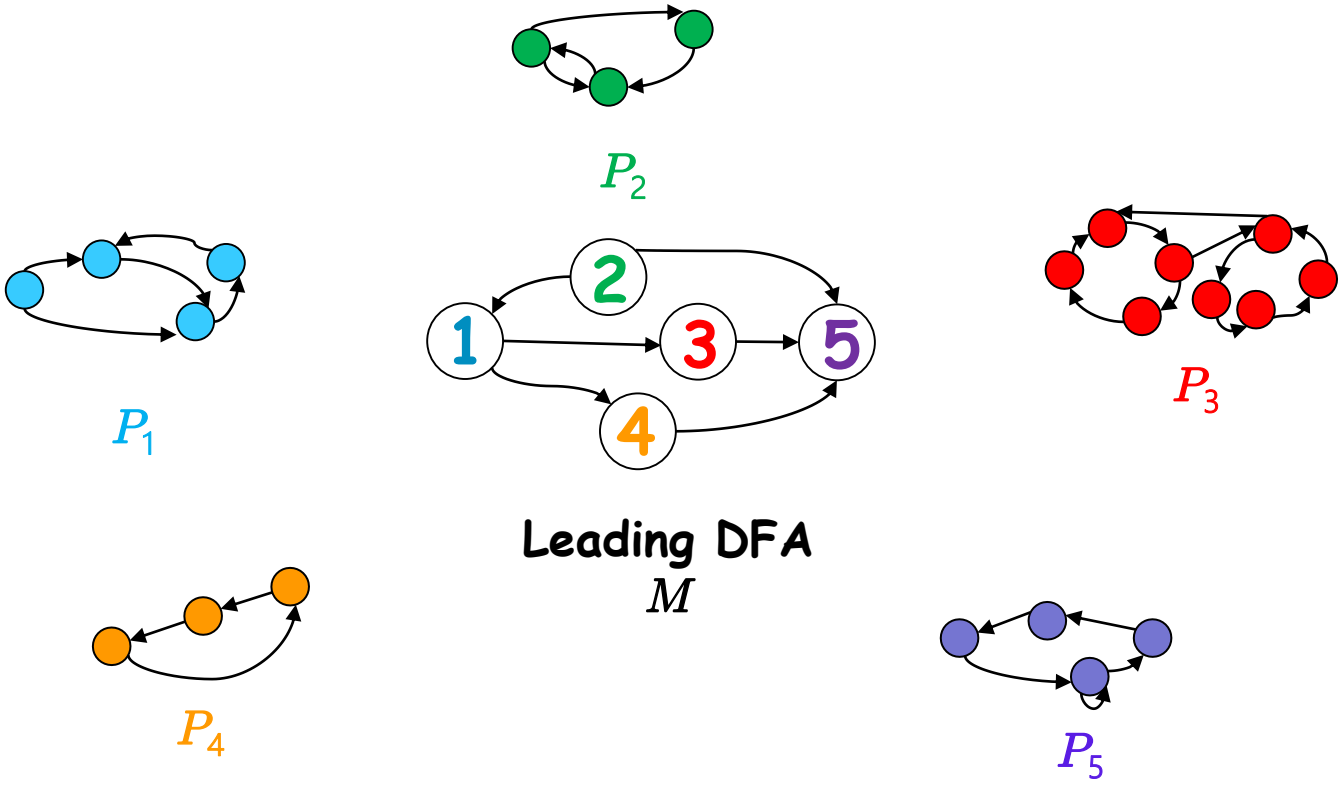


Plus some restriction (details ommited)

Family of DFAs (FDFA)

$(M, P_1, P_2, P_3, P_4, P_5)$

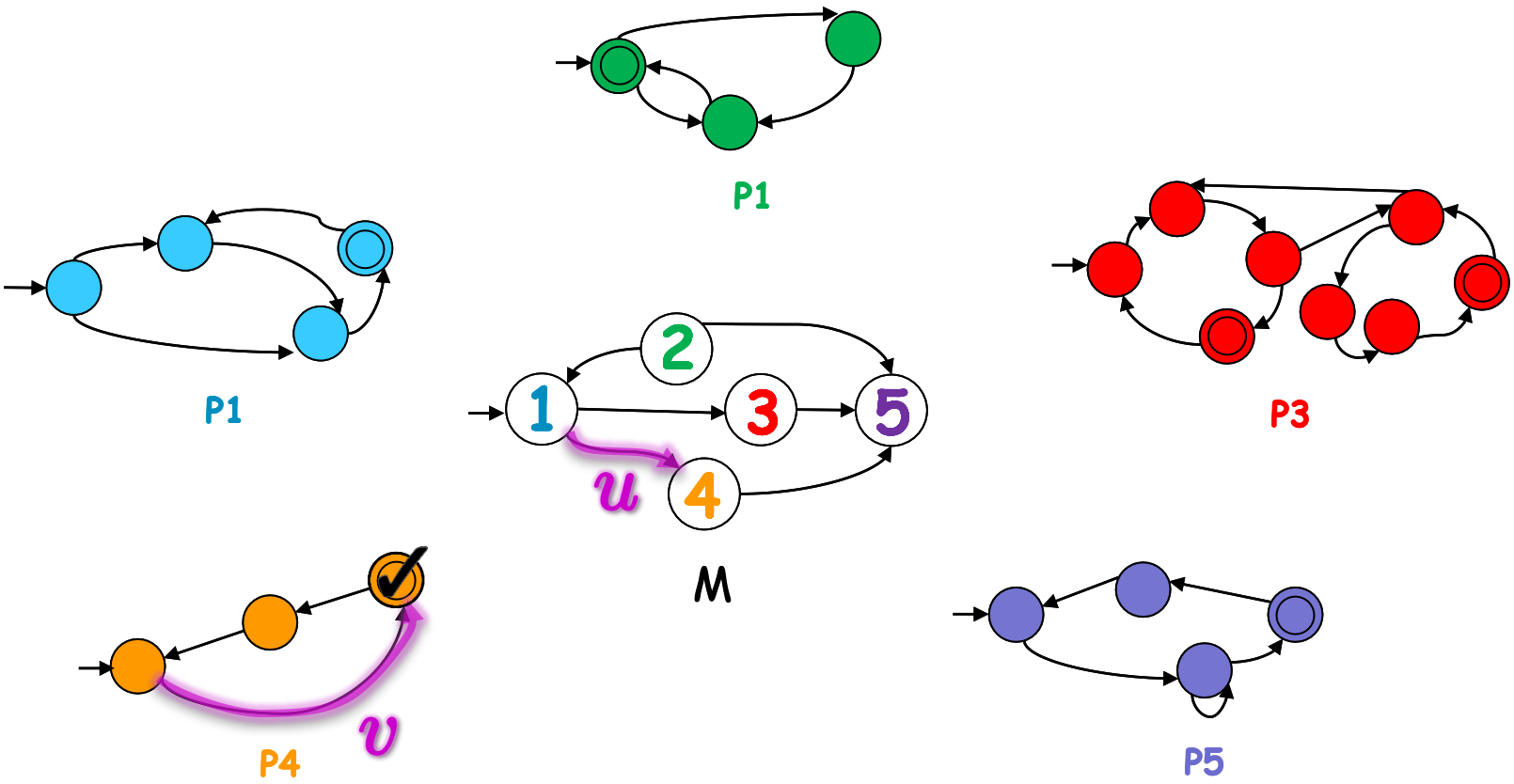
Leading — Progress



That restriction is removed

FDFA Acceptance

$$(u,v) \overset{?}{\in} [\![M, P_1, P_2, P_3, P_4, P_5]\!] \quad (\mathcal{F})$$



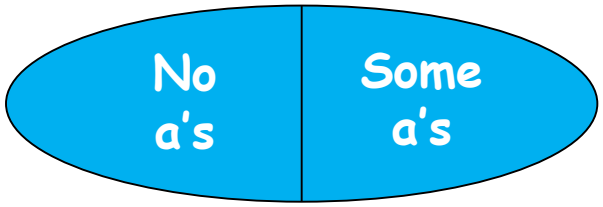
FORC and FDFA - Example

$L = \{ w \mid w \text{ has finitely many } a\text{'s} \}$

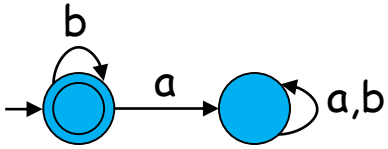
FORC

FDFA

Progress

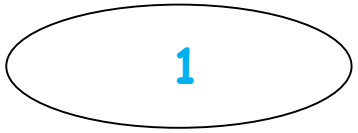


$\approx_1$

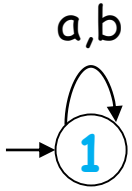


P_1

Leading



$\sim$



All prefixes are equally good

$(abba, bbb)$ ✓

$(abba, bab)$ ✗

Three Canonical FDFAs

3 canonical FDFAs:

- **Periodic** [corr. to $L_\$$ of Calbrix, Nivat & Podelski, 1993]
  - The simplest to understand
- **Syntactic** [Maler & Staiger, 1997]
  - The coarsest possible FORC
- **Recurrent**
  - Simplification of the Syntactic
  - The smallest among the three

No contradiction here. The Recurrent is not a FORC. Save up is due to the removal of that restriction.

Three Canonical FDFAs

- All three canonical FDFAs use $\sim_L$ for the leading automaton.
- It is the smallest choice possible for the leading automaton [MS97].
- The canonical FDFAs differ in the definition of the right congruence for the progress automata.

The Progress Right Congruences

Periodic

$$x \approx^{u}_{\mathsf{P}} y$$

Syntactic

$$x \approx^{u}_{\mathsf{S}} y$$

Recurrent

$$x \approx^{u}_{\mathsf{R}} y$$

The Progress Right C...

Considering periods starting at u, x and y should agree on all extensions

Periodic

$$x \approx^{u}_{\mathsf{P}} y \quad \text{iff}$$

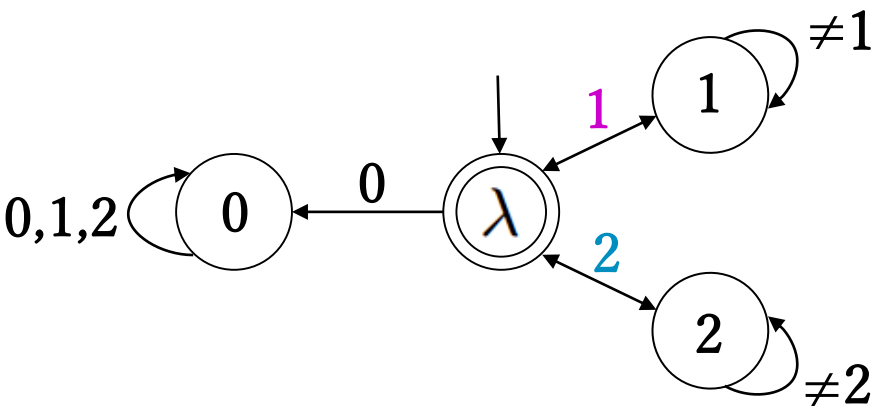
$$u(xv)^{\omega} \in L \iff u(yv)^{\omega} \in L$$

Syntactic

Recurrent

Possible Periods of λ

- Some periods are easy to find:
  - 11
  - 22
  - 100201
  - 10020122



- Some are hard:
  - 1012 — 1012 1012 1012 1012
  - 12 — 12 12 12 12 12 12

The periodic progress DFA for λ has 23 states !

The Progress Right Congruences

Periodic

Syntactic

x and y should agree on extensions that loop back to u in the leading automton

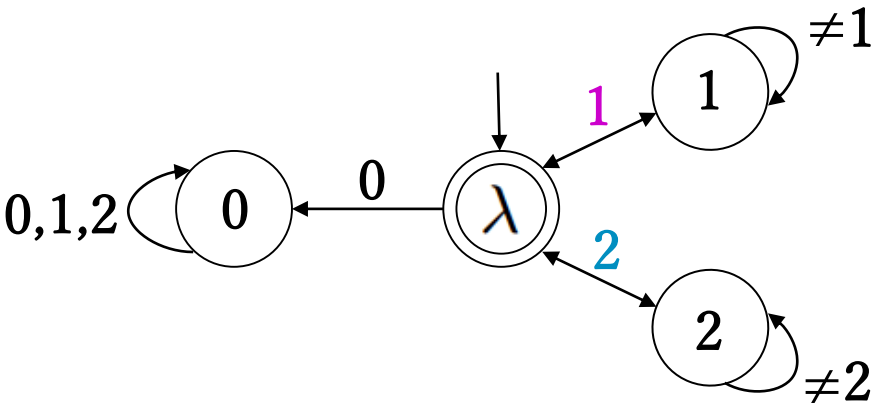
Recurrent

$$x \approx_R^u y \quad \text{iff}$$

$$\forall v \in \Sigma^*. \quad uxv \sim_L u \text{ and } u(xv)^\omega \in L \iff uyv \sim_L u \text{ and } u(yv)^\omega \in L$$

Periods of λ that loop back

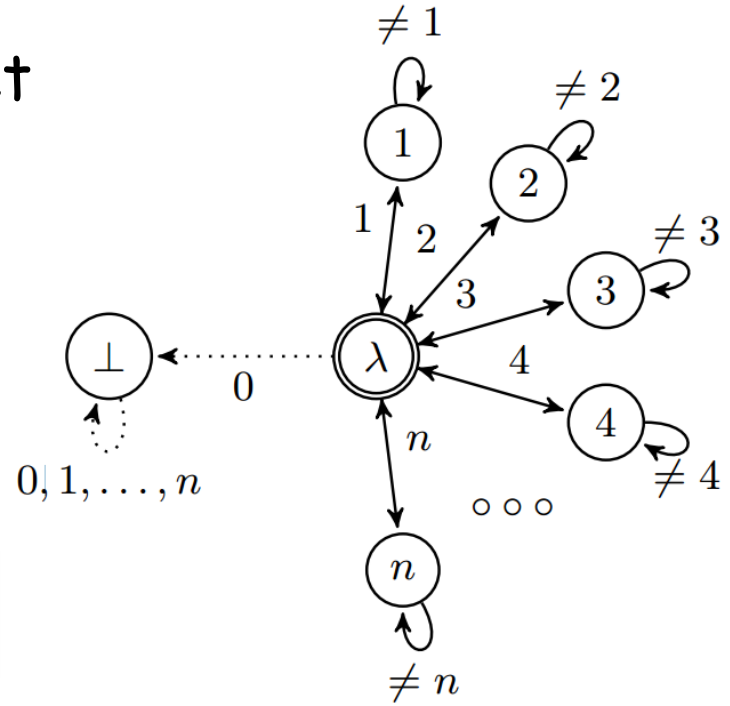
- All periods that loop back are easy to find:
  - 11
  - 22
  - 100201
  - 10020122

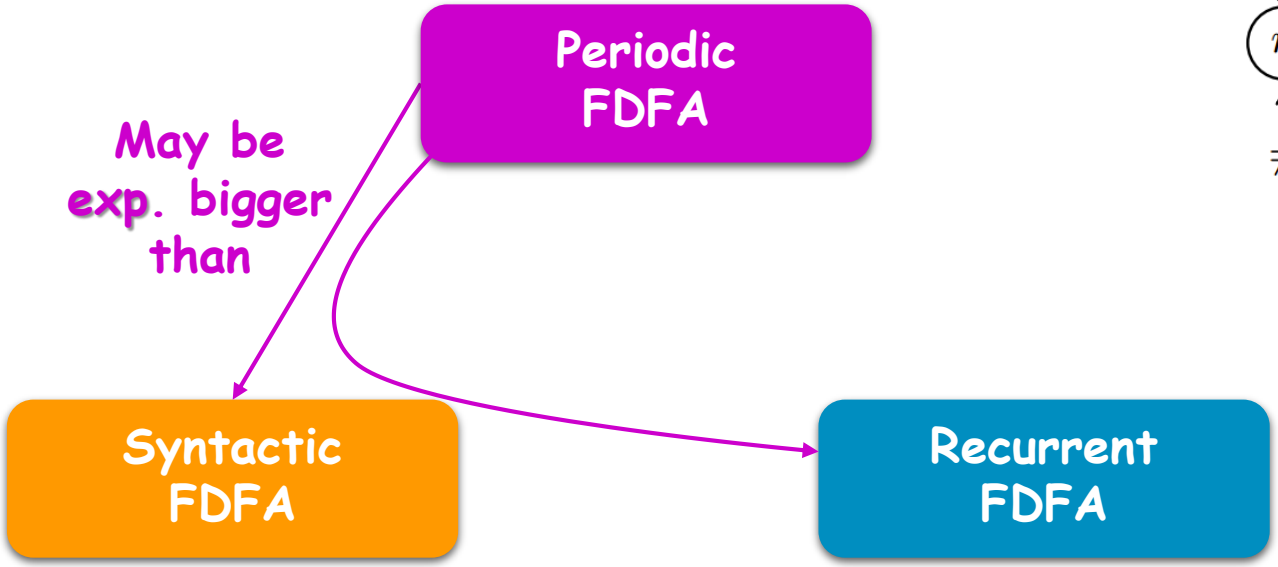


The recurrent progress DFA for λ has only 5 states !

More generally

- Generalizing to arbitrary n we get the following lower bound





Saturation (Consistency)

- But wait, wouldn't you like to get the same result for 12 as you do for 12 12 12 ?
- Well… yes

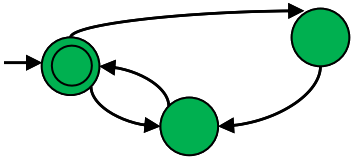
We say that a language is saturated if

$uv^\omega = xy^\omega$ implies $(u,v) \in L \Leftrightarrow (x,y) \in L$

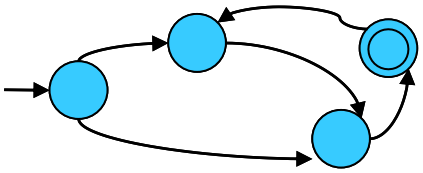
- To achieve saturation, we change the definition of acceptance.

FDFA Exact Acceptance

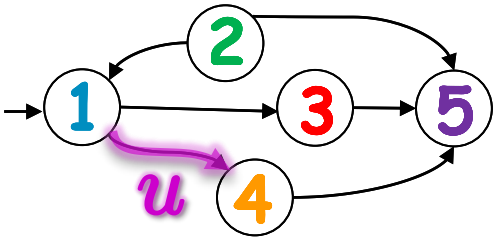
$$(u,v) \overset{?}{\in} \overbrace{[\![M, P_1, P_2, P_3, P_4, P_5]\!]}^{\mathcal{F}}$$



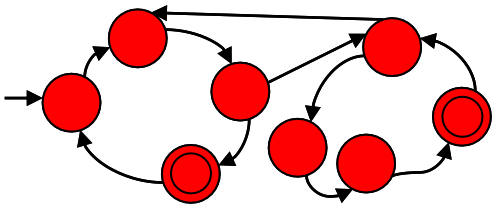
P1



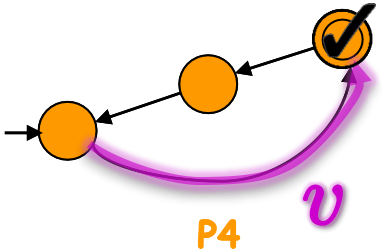
P1



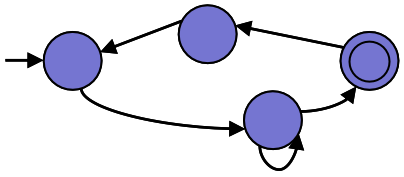
M



P3



P4

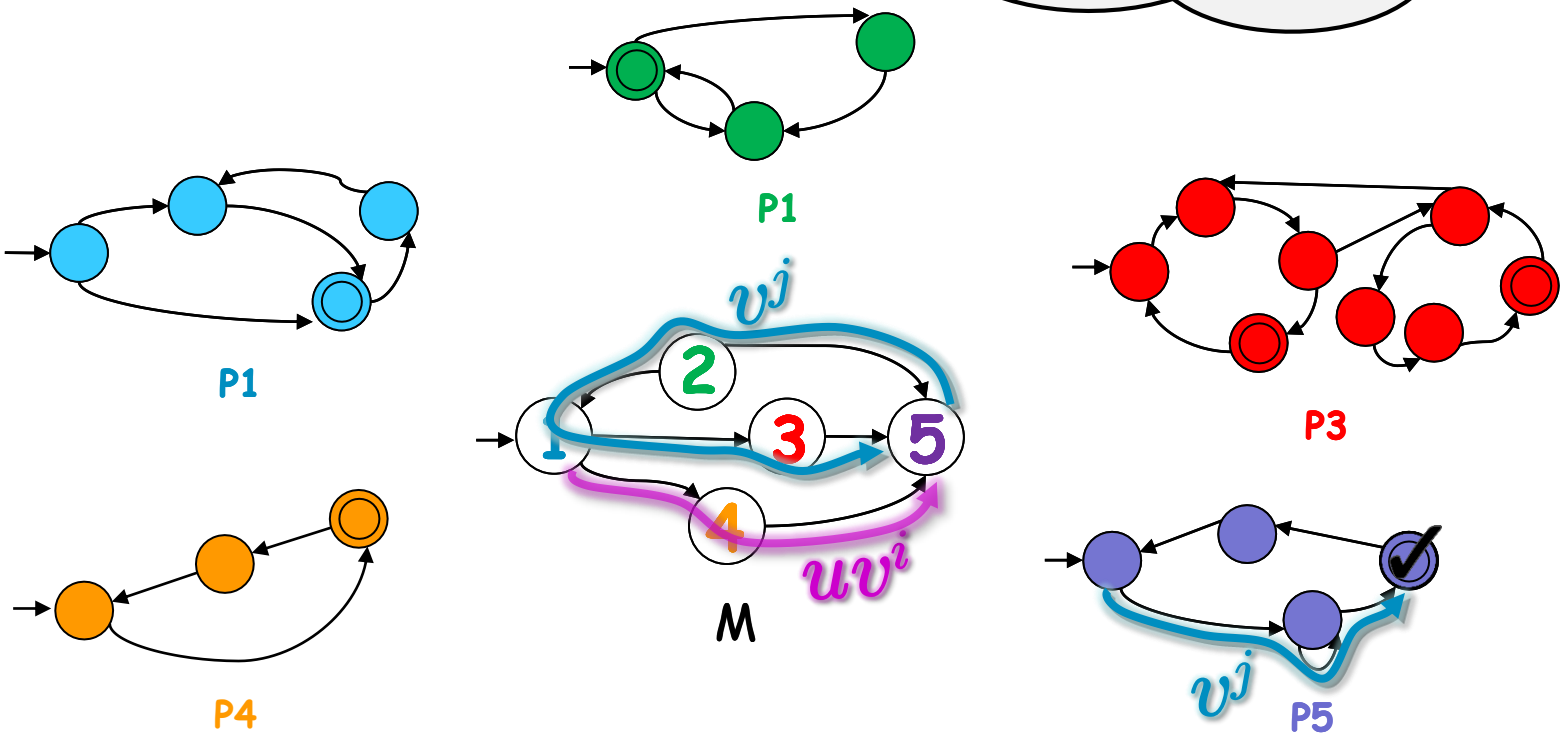


P5

FDFA Normalized Acceptance

$$(u,v) \overset{?}{\in} \overbrace{[\![M, P_1, P_2, P_3, P_4, P_5]\!]}^{\mathcal{F}}$$

Normalization seeks for the smallest repetition of the period that loops back



Syntactic/Recurrent FDFA are saturated under Normalized Acceptance.

Learning Algorithm for all 3 Canonical FDFAs

Learning FDFAs

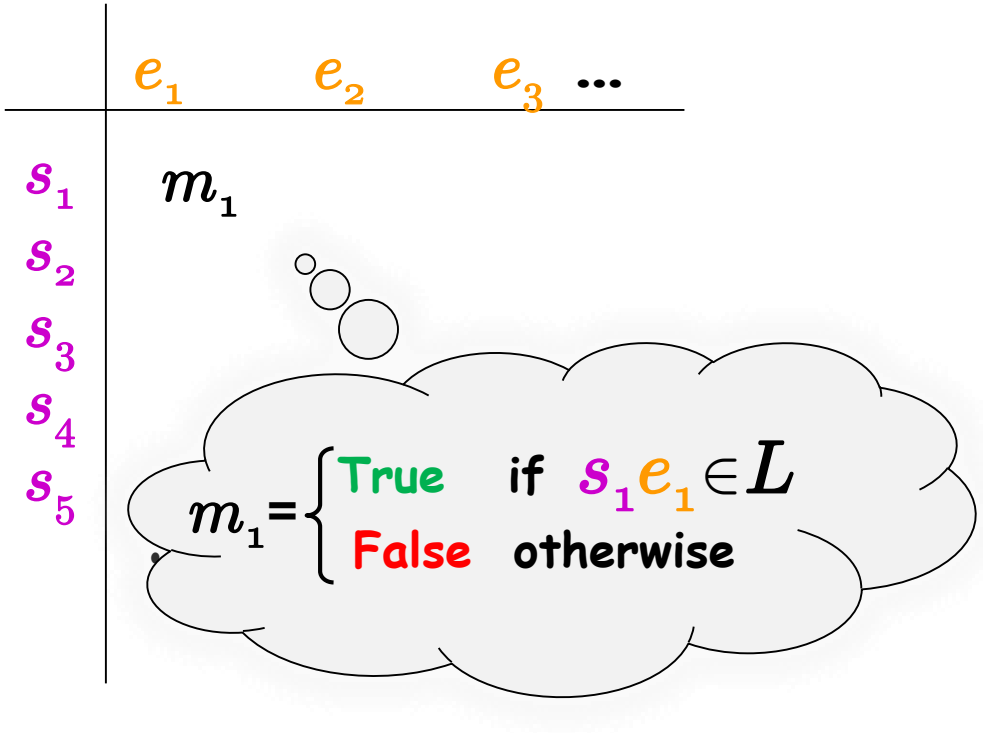
- We now have a 1-to-1 correspondence between FDFA states and right congruences
- Note that we don't have oracles for the languages of the leading/progress DFAs. We have oracles for the w-language itself.

Main Idea

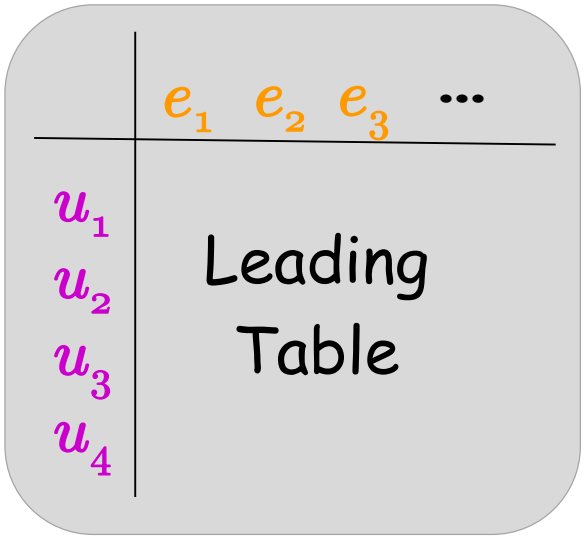
- Similar to L*
- We learn the leading and progress DFAs simultaneously
- When we get a counter example, we first try to see if we can distinguish a new state in the leading DFA
- Otherwise we try to distinguish a state in the respective progress DFA

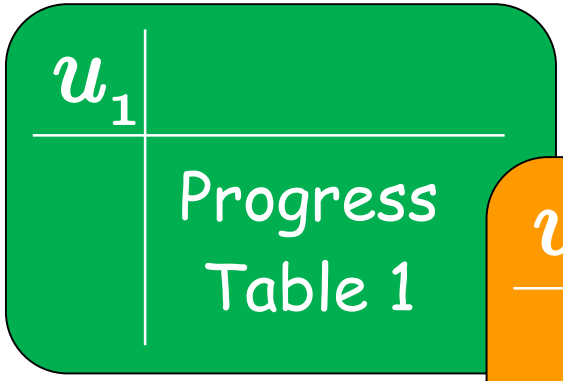
L^* Data Structure

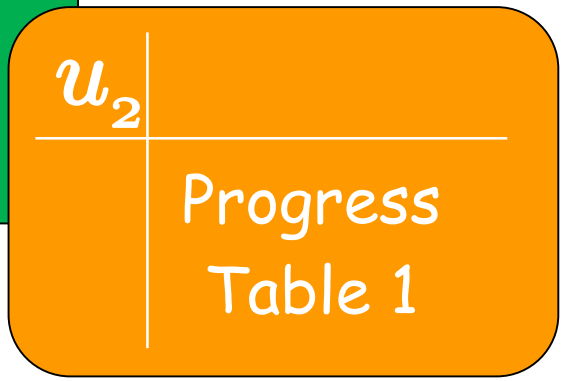
Observation Table

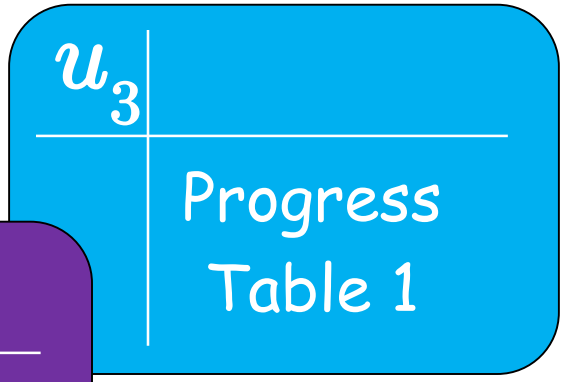


L^w Data Structures



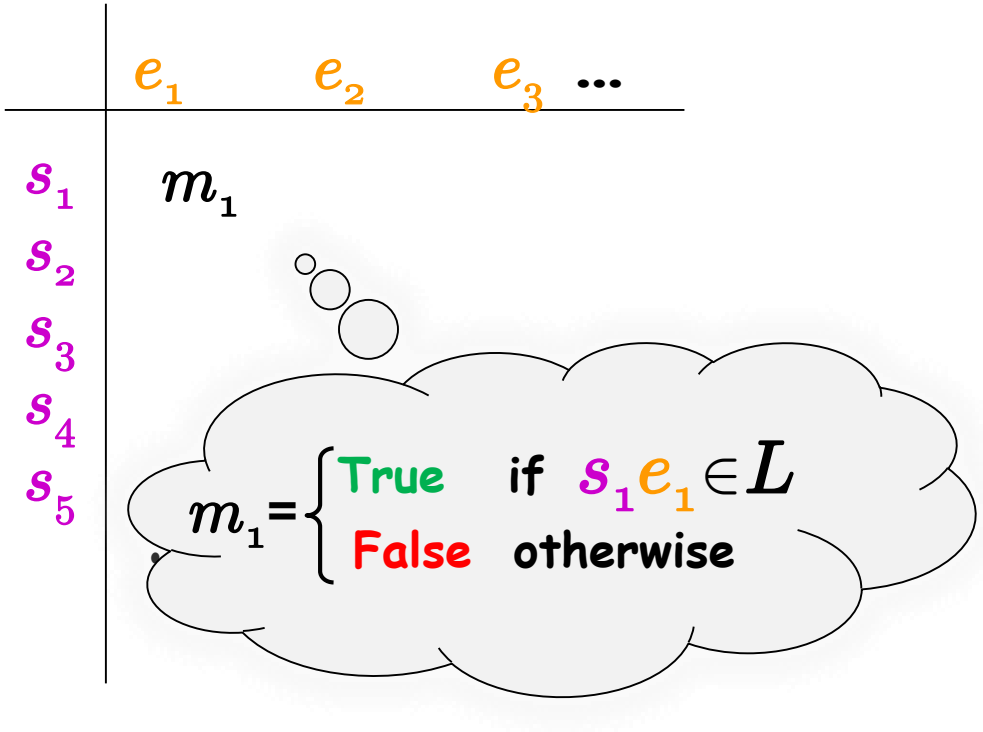






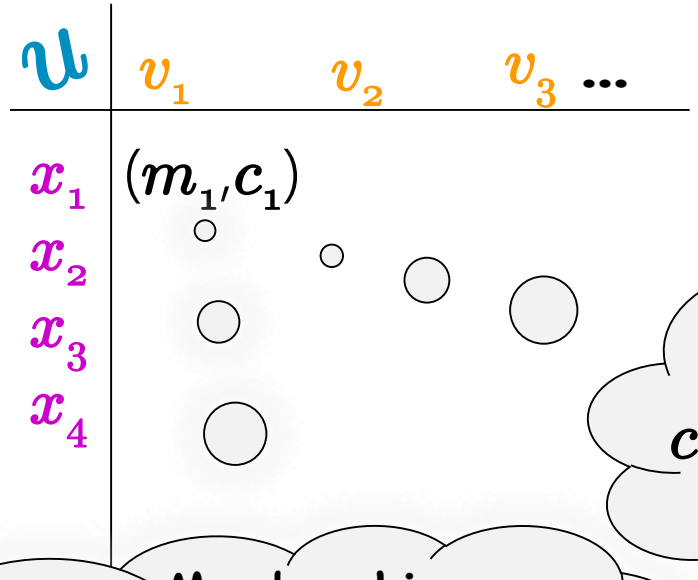
L^ω Leading Table

Same as in L* only that experiments e_i are ultimately periodic words $u_i v_i^\omega$



L^{ω} - Recurrent Progress Table

Recurrent
Observation Table



Cycle-Closing:

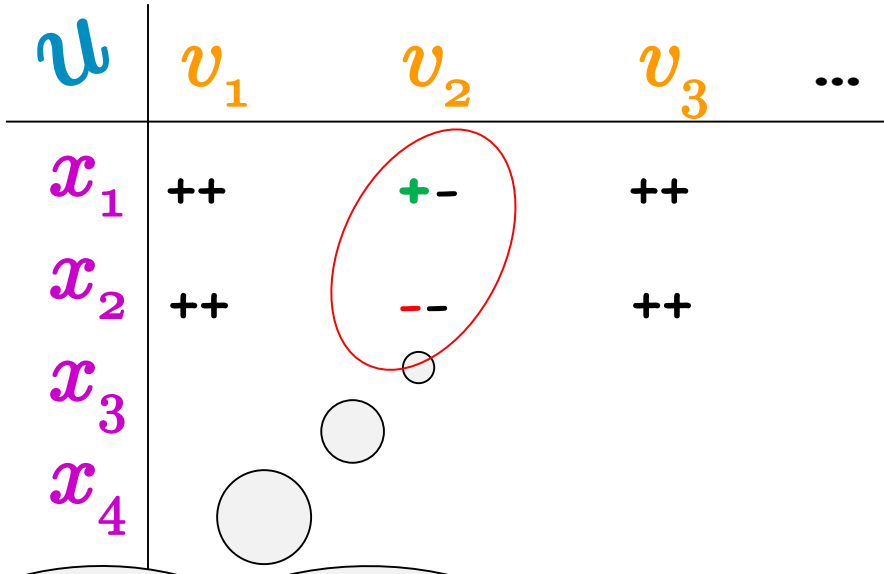
$$c_1 = \begin{cases} \text{True} & \text{if } ux_1v_1 \sim u \\ \text{False} & \text{otherwise} \end{cases}$$

Membership:

$$m_1 = \begin{cases} \text{True} & \text{if } u(x_1v_1)^{\omega} \in L \\ \text{False} & \text{otherwise} \end{cases}$$

From tables to an algorithm

- How to define when two rows should be considered distinguished ?
- Given a counter example, how do we find an experiment that distinguishes a new state ?



Difficulty:
when an experiment differentiates two strings with respect to membership, it might not close a cycle, thus should not necessarily imply the strings are in different equivalence classes …

Minimality?

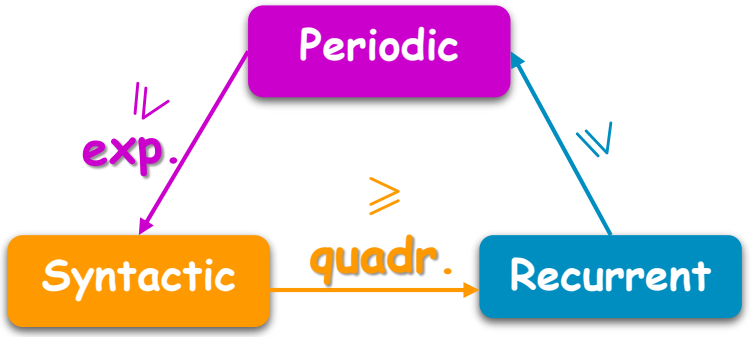
- The Recurrent FDFA for a given L is not necessarily the minimal FDFA for L.
- According to the normalized acceptance criterion a progress DFA P_u should give correct results only to extensions that close a cycle to u. On other extensions it has freedom.
- This has the flavor of minimization with unknowns, which is NP complete.
- The Recurrent FDFA chooses to treat all don't cares as rejecting.

Time Complexity

- Interestingly, [Klarlund, 1994] has shown that choosing a leading automaton which is more refined (bigger) than the syntactic right congruence, may yield an overall smaller FDFA.
- If we are given such a leading automaton, we can feed it to the learning algorithm, in which case it will yield the smaller FDFA.
- However, such a phenomenon can cause the learning algorithm for the syntactic/recurrent FDFAs to work as hard as the one for the periodic FDFA
- The time complexity for all 3 families is thus polynomial in the size of the periodic FDFA.

Recap

- Learning ω-languages is important for verification of reactive systems.
- No learning alg. for traditional ω-automata since they don't have a Myhill-Nerode theorem.
- We have provided a L*-style learning algorithm for all three canonical representations of ω-languages, based on the notion of FDFAs.
- While the size relations are



worse case time complexity depends on the largest representation.

Future Direction

- Find smaller canonical representations ?
- Find smallest FDFA ?
- Learning the leading first ?
- L*-learning for (traditional) ω-automata ?
- Other types of learning for ω-languages ?