Regular languages

Learnable representations for languages

Alexander Clark

Department of Computer Science
Royal Holloway, University of London

August 2010
ESSLLI, 2010

Overview

Regular languages

- well developed theory of inference
- explain basic representational approaches
- look at 3 specific algorithms

Algorithms work in practice:

- random DFAs are efficiently learnable
- Tenjinno competition
- but you can construct families of DFAs which are hard to learn.

Regular languages

A natural class

- Language theoretic characterisation
- Two machine models
- A grammar model

Predate language theory:
The natural numbers of languages. (Dedekind)

Regular languages

Regular expressions

Given Σ, the class of regular languages over Σ is the smallest set of languages, R, such that

- R includes the singleton languages $\{a\}$ for $a \in \Sigma$, and $\{\lambda\}$, and $\emptyset$
- If $A, B \in R$, then $A \cup B \in R$
- If $A, B \in R$, then $AB \in R$
- If $A \in R$, then $A^* \in R$

Deterministic finite automata

Definition

A DFA A is a tuple $(Q, \Sigma, q_0, F, \delta)$, where

- Q is a finite set of states,
- Σ is the alphabet, a finite set of symbols,
- $q_0 \in Q$ is the single initial state,
- $F \subseteq Q$ are the final states,
- $\delta : Q \times \Sigma \to Q$ is the transition function.

Either assume that the transition function is complete: by having a sink state, or have a partial function.

Non-deterministic finite automata

with λ-moves

Definition

A NFA A is a tuple $(Q, \Sigma, q_0, F, \delta)$, where

- Q is a finite set of states,
- Σ is the alphabet, a finite set of symbols,
- $q_0 \in Q$ is the single initial state,
- $F \subseteq Q$ are the final states,
- $\delta : Q \times (\Sigma \cup \lambda) \to 2^Q$ is the transition function.

We can determinise NFAs but the number of states may increase exponentially.

Regular grammar

Special form of Phrase-structure grammar:

- Tuple $\langle \Sigma, V, P, S \rangle$
- P is a set of productions of the form: $M \rightarrow wN$ or $M \rightarrow w$.

Conversion

Easy to convert NFA to regular grammar and back. $\delta(q, a) = q'$
$N_q \rightarrow aN_{q'}$
If $q \in F$: then $N_q \rightarrow \lambda$

Regular inference

A success story

Paradigm	Learnable class	
Positive Data	reversible languages	Angluin (1982)
Queries	regular languages	Angluin (1987)
Positive and Negative	regular languages	Oncina and Garcia (1992)
Stochastic data	acyclic PDFAs	Ron et al (1994),
	regular languages	Carrasco and Oncina (199
	regular languages	Clark and Thollard (2004)

See de la Higuera (2010) for an excellent treatment.

Outline

Congruence and residual languages

Given a language L and a string w, we define
$w^{-1}L = \{v \in \Sigma^* | wv \in L\}$

$L = \{(ab)^*\}$

- What is $a^{-1}L$?
- What is $(ab)^{-1}L$?
- What is $\lambda^{-1}L$?
- What is $b^{-1}L$?

Congruence and residual languages

Given a language L and a string w, we define
$w^{-1}L = \{v \in \Sigma^* | wv \in L\}$

$L = \{(ab)^*\}$

- What is $a^{-1}L$?
- What is $(ab)^{-1}L$?
- What is $\lambda^{-1}L$?
- What is $b^{-1}L$?

Congruence

definition

$u \equiv_L v$ iff $u^{-1}L = v^{-1}L$

An equivalence relation – a congruence.

Exercise

What are the equivalence classes of $L = \{(ab)^*\}$?

Congruence

definition

$u \equiv_L v$ iff $u^{-1}L = v^{-1}L$

An equivalence relation – a congruence.

Exercise

What are the equivalence classes of $L = \{(ab)^*\}$?

Answer

- L
- $\{a, aba, \dots\} = La$
- $\{b, bb, bab \dots\}$ – everything else

Non-regular languages

If the language is non-regular then the number of congruence classes is infinite.

Congruence classes of $\{a^n b^n | n \geq 0\}$

- $\{a^k\}$ for each $k \geq 0$
- $\{a^{i+k} b^i | i > 0\}$ for each $k \geq 0$

The Canonical DFA

The states are the congruence classes – or symbols in bijection.

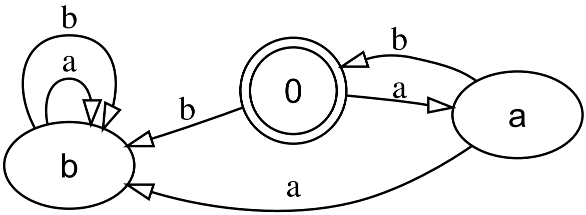
- Objectivity of the primitive elements
- Local validity of the rules given representation.

Right congruence

$u \equiv v$ implies $ua \equiv va$
Transition function $\delta([u], a) = [ua]$

Example

$L = (ab)^*$



Basic idea

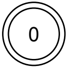
If we can tell whether $u \equiv_L v$ then we can write down the correct minimal DFA, once we have enough examples.

- Create a new state labelled with λ.
- For each state labelled with u; try all strings ua, $a \in \Sigma$
- If $ua \equiv_L v$ for some state labelled v; add transition $u \rightarrow^a v$.
- Otherwise create new state labelled ua.
- Repeat

Will terminate if language is regular.

Example

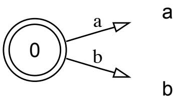
$L = (ab)^*$



Create initial state.

Example

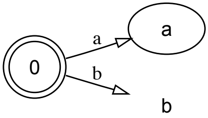
$L = (ab)^*$



Try *a* and *b*. Test *a*.

Example

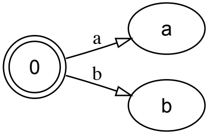
$L = (ab)^*$



Test *b*.

Example

$L = (ab)^*$

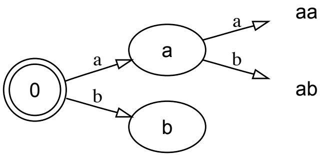


We know we have at least these three states.

Try state *a*.

Example

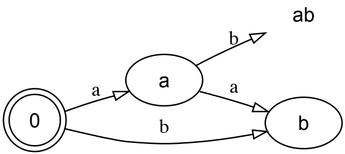
$L = (ab)^*$



$aa \equiv_L b.$

Example

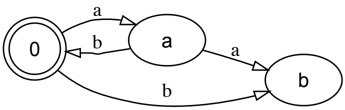
$L = (ab)^*$



$ab \equiv_L \lambda$

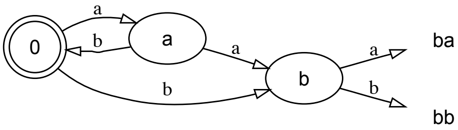
Example

$L = (ab)^*$



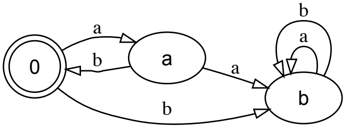
Example

$L = (ab)^*$



Example

$L = (ab)^*$



Oracle

Suppose we have an oracle: a machine for testing whether $u \equiv_L v$. A very powerful source of information.

- If language is regular we can write down an exactly correct DFA.
- If language is not-regular, we can stop after some finite time and we have a regular subset of the language.

Testing

Suppose we don't have such an oracle. We can substitute it with weaker information.

Three approaches:

- Restrict the class of languages so it is easy to tell from positive examples alone.
- Allow some other form of queries so we can test.
- Make some assumptions about the distribution of examples, so we can test probabilistically.

State merging algorithms

The Prefix Tree Acceptor

We start from a set of strings S, a finite subset of L.

- $Pr(w) = \{v | \exists u, vu = w\}$ set of prefixes of w
- $Pr(S) = \bigcup_{w \in S} Pr(w)$

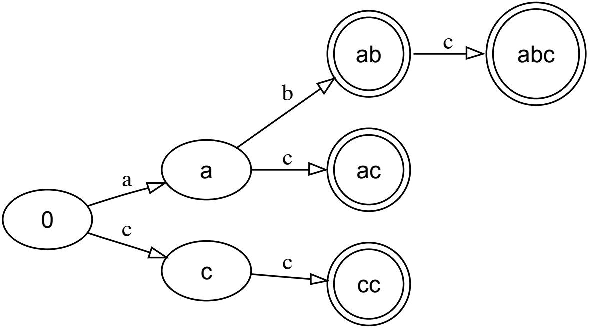
Prefix Tree Acceptor of S

- $Q = Pr(S)$
- $q_0 = \lambda$
- $F = S$
- $\delta(u, a) = ua$

PTA

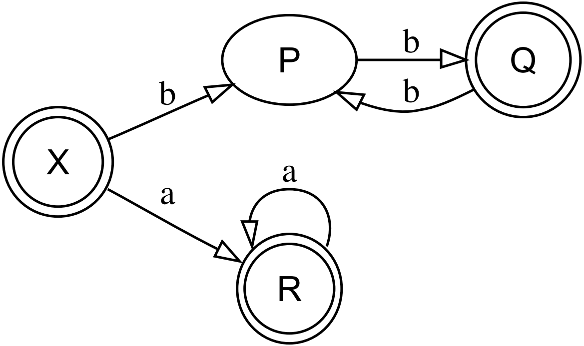
Diagram

$\{ab, abc, ac, cc\}$



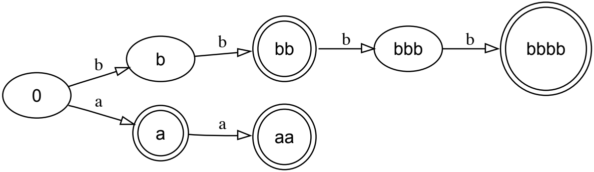
Example language

$a^* \cup (bb)^*$



Sample

$\{\lambda, a, aa, bb, bbbb\}$



PTA

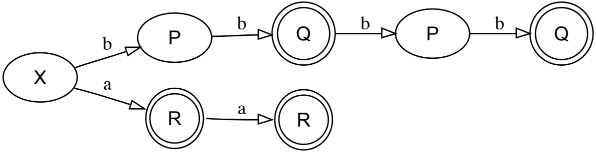
Basic properties

- Generates exactly the training sample S
- Efficient
- Deterministic Finite Automaton

Sample

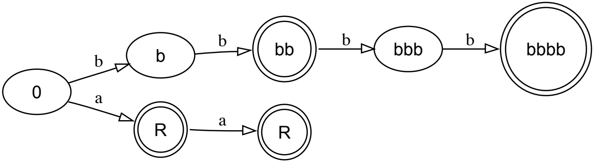
$\{\lambda, a, aa, bb, bbbb\}$

Label states of PTA with the correct states.



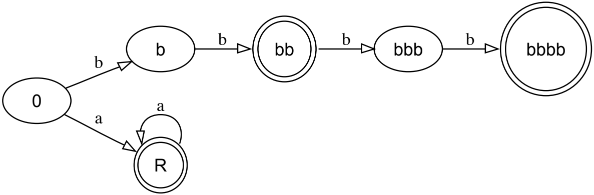
State merging

Merge two states and then determinise.



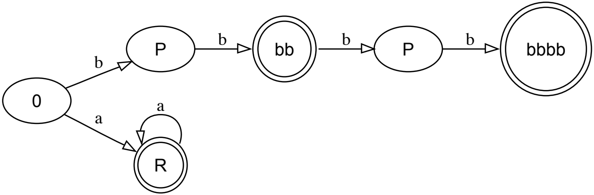
State merging

Merge two states and then determinise.



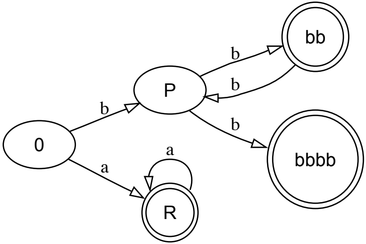
State merging

Merge two states and then determinise.



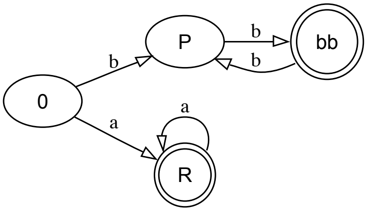
State merging

Merge two states and then determinise.



State merging

Merge two states and then determinise.



Outline

Basic representational issues

Reversible languages

LSTAR Algorithm

PDFA algorithm

General Inference Algorithms

Regular languages

Reversible languages

Definition

If $uv, u'v, uv' \in L$ then $u'v' \in L$
OR: If we have $uv, u'v \in L$ then $u \equiv_L u'$

Example

$L = \{(ab)^*\}$

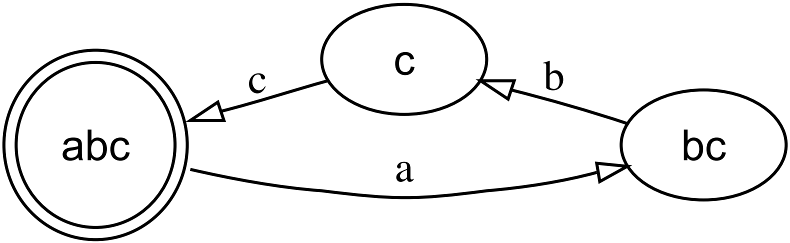
Example non-reversible

$L = \{a, aa\}$

Reversible automata

Definition

One final state.
If $p \rightarrow^a q$ and $r \rightarrow^a q$, then $p = r$
Deterministic when we go backwards.



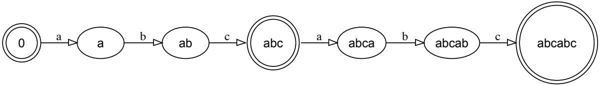
Algorithm

Angluin (1982)

- Construct PTA.
- Merge all the final states
- If it is not reversible:
  - find two states q, q' that have an arc labelled a leading into the same state,
  - merge them
  - determinise
- Repeat until it is reversible.

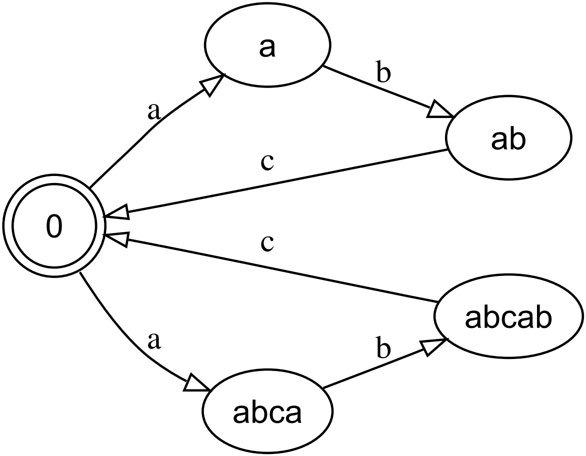
Example

$\{\lambda, abc, abcabc\}$



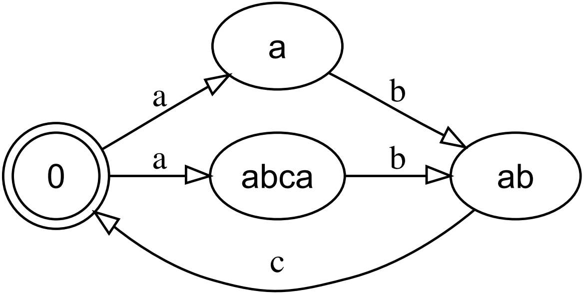
Example

$\{\lambda, abc, abcabc\}$



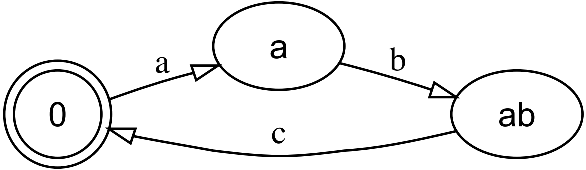
Example

$\{\lambda, abc, abcabc\}$



Example

$\{\lambda, abc, abcabc\}$



Outline

MAT model

The Minimally adequate teacher model.

Two queries

- Membership queries
- Equivalence queries: returns counterexample
  - May be undecidable in general
  - Can be approximated probabilistically

Pros and cons?

Angluin 1987

Learning regular sets from queries and counter-examples

Basic idea

test whether $u^{-1}L = v^{-1}L$

Pick a finite set X

test whether $u^{-1}L \cap X = v^{-1}L \cap X$

Angluin 1987

Learning regular sets from queries and counter-examples

Basic idea

test whether $u^{-1}L = v^{-1}L$

Pick a finite set X

test whether $u^{-1}L \cap X = v^{-1}L \cap X$

Algorithm

Maintain two sets:

- A set of prefixes which will give us the states K
- A set of experiments that we use to differentiate the states X

We start with

- $K = \{\lambda\}$
- $X = \{\lambda\}$

Observation table

Rows are prefixes, columns are suffixes

	λ	abc	$abcabc$	bc	$bcabc$	c	$cabc$	a
b								
$abcab$						■	■	
ab						■	■	
$abca$				■	■			
a				■	■			
abc	■	■	■					
λ	■	■	■					

Fill in with MQs

States

Similarity

Two strings are similar $u \sim_X v$ iff the rows are identical $u^{-1}L \cap X = v^{-1}L \cap X$

- If $u^{-1}L = v^{-1}L$ then $u^{-1}L \cap X = v^{-1}L \cap X$
- If not, then there is some q such that when $q \in X$, $u^{-1}L \cap X \neq v^{-1}L \cap X$

States will be equivalence classes of rows.

Next state

If $u \in K$, then we want to know what happens to ua

- Two sets of rows
- K
- $K\Sigma \setminus K$

Consistency

If $u \sim_X v$ then $ua \sim_X va$ for all $a \in \Sigma$

Closed

For every row in $K\Sigma$ there is a row in K

If closed and consistent then we can write down a DFA.

Making consistent

- If it is not consistent then there is $u_1, u_2 \in K$, and $a \in \Sigma$ and $e \in X$ such that
  - $u_1 \sim_X u_2$
  - $u_1 ae$ in L, $u_2 ae$ not in L

Solution: add ae to X, and then u_1 and u_2 will be different.

Make closed

Find a $s \in K$ and an $a \in \Sigma$ such that sa is not equivalent to any element of K.
Add sa to K.

Algorithm

Result: A DFA A

$K \leftarrow \{\lambda\}$;

$X \leftarrow \{\lambda\}$;

Construct observation table ;

while *true* **do**

- **while** *Table is not closed or consistent* **do**
  - **if** *Table is not consistent* **then**
    - Find u_1, u_2, a, e and add $ae \rightarrow X$;
  - **if** *Table is not closed* **then**
    - Add $s_1 a$ to S ;
- Construct A from table ;
- Query A ;
- **if** *Teacher gives counterexample t* **then**
  - Add $Pr(t)$ to S ;
- **else**
  - Halt and output A ;

Example

Angluin 1987

Target

Language over $\{0, 1\}$ with an even number of 0s and an even number of 1s.

T_1	λ
λ	1
0	0
1	0

Example

Angluin 1987

T_2	λ
λ	1
0	0
1	0
00	1
01	0

Query and receive the counterexample 11

Example

Angluin 1987

T_3	λ
λ	1
0	0
1	0
11	1
00	1
01	0
10	0
110	0
111	0

Not consistent as *row*[0] $=$ *row*[1], but 00 is different from 10.

Example

Angluin 1987

T_4	λ	0
λ	1	0
0	0	1
1	0	0
11	1	0
00	1	0
01	0	0
10	0	0
110	0	1
111	0	0

$$S = \{\lambda, 0, 1, 11\}, E = \{\lambda, 0\}.$$

Query and receive the counterexample 011

Example

Angluin 1987

T_5	λ	0
λ	1	0
0	0	1
1	0	0
11	1	0
01	0	0
011	0	1
00	1	0
10	0	0
110	0	1
111	0	0
010	0	0
0110	1	0
0111	0	0

$S = \{\lambda, 0, 1, 11, 01, 011\}$, $E = \{\lambda, 0\}$.

Not consistent – 1 and 01

Example

Angluin 1987

T_6	λ	0	1
λ	1	0	0
0	0	1	0
1	0	0	1
11	1	0	0
01	0	0	0
011	0	1	0
00	1	0	0
10	0	0	0
110	0	1	0
111	0	0	1
010	0	0	1
0110	1	0	0
0111	0	0	0

FIG. 9. $S = \{\lambda, 0, 1, 11, 01, 011\}$, $E = \{\lambda, 0, 1\}$.

Outline

PDFA

Definition

A DFA with a probability function:

- Probability of each transition: $P(a|q)$
- Probability of terminating: $P_f(q)$

$P_f(q) + \sum_{a \in \Sigma} P(a|q) = 1$

Defines $P(q \to a)$, $P(q_0 \to a)$.

Model

Error measure between Target, Hypothesis:

$$KLD(T||H) = \sum_{w \in \Sigma*} P_T(w) \log \frac{P_T(w)}{P_H(w)} \tag{1}$$

Convergence

When we have received $p(1/\epsilon, 1/\delta, n)$ examples, then with prob at least $1-\delta$ we have a hypothesis such that $KLD(T||H) < \epsilon$

Stratifying the class

Ron, Singer and Tishby (1988), JCSS

Distinguishability

Two states q_1, q_2 are μ-distinguishable if there is a string u such that $|P(q_1 \rightarrow u) - P(q_2 \rightarrow u)| \geq \mu$.

Stratifying the class

Ron, Singer and Tishby (1988), JCSS

Distinguishability

Two states q_1, q_2 are μ-distinguishable if there is a string u such that $|P(q_1 \to u) - P(q_2 \to u)| \geq \mu$.

- If this criterion holds, then given a polynomial bound on the number of samples for a residual language, we can determine with high probability whether these are similar or not.

Some technical problems

Algorithm generates the automaton incrementally: merging new states with existing states according to a stochastic criterion.

- Smoothing and a sink state
- Re-sampling
- Need an additional bound for length

Learning all PDFAs

Clark and Thollard (2004) JMLR

Theorem

The class of all PDFAs can be PAC-learned from positive data with the KLD as distance function with sample complexity polynomial in $\epsilon^{-1}, \delta^{-1}$ and

- *n* is the number of states of the automaton
- $|\Sigma|$ is the number of letters in the alphabet
- μ^{-1} is the distinguishability
- *D* is an upper bound on the expected length of strings generated from a state.

Outline

Basic representational issues

Reversible languages

LSTAR Algorithm

PDFA algorithm

General Inference Algorithms

- Regular languages

Algorithms

Classic Variants

Classic

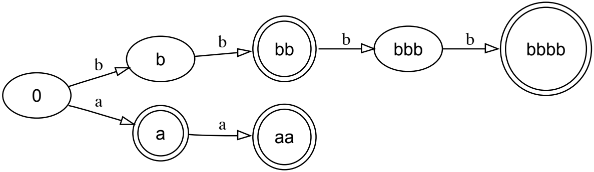
Merge states of the PTA until the automaton is deterministic.

Non-classic

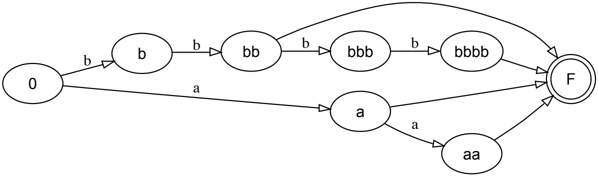
Given input data *Data*:

- $Q = Pr(Data)$.
- $[[u]] \rightarrow^{a} [[ua]]$
- If we want to merge u, v, then add transition $[[u]] \rightarrow^{\lambda} [[v]]$
- Non-deterministic representation

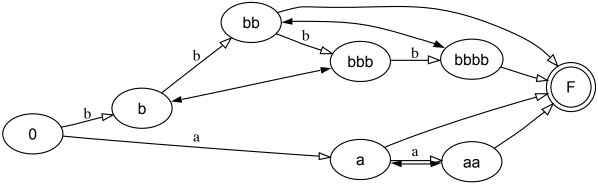
PTA example



Add terminal transitions



Add null transitions between identical states



Three types of transitions

$u \rightarrow^{a} ua$

A priori valid

Terminal transitions

$u \rightarrow^{term} F$ is valid if $u \in L$

Certain based on a limited bit of information

Equality transitions

Defeasible

Appear to be true based on information in X

Generally might turn out to be false later on.

Objective representations

Program

Given a language L

1. Define a collection of sets of strings as primitives
2. Define a derivation relation, based on algebraic properties of these sets.
3. Define a representation based on this derivation relation

Step 1

Make a representational decision

Residual languages

$u^{-1}L = \{w | uw \in L\}$

right congruence classes

$u \cong v$ iff $u^{-1}L = v^{-1}L$

$[u]_R = \{v | u \cong v\}$

Primitives

Finite set of strings K; $\lambda \in K$

$Q = \{[u]_R | u \in K\}$

$q_0 = [\lambda]_R$

Step 1

Make a representational decision

Residual languages

$u^{-1}L = \{w | uw \in L\}$

right congruence classes

$u \cong v$ iff $u^{-1}L = v^{-1}L$

$[u]_R = \{v | u \cong v\}$

Primitives

Finite set of strings K; $\lambda \in K$

$Q = \{[u]_R | u \in K\}$

$q_0 = [\lambda]_R$

Let's call the elements of Q "states"

Example

$L = \{(ab)^*\}$

- $[\lambda]_R = L$
- $[a]_R = aL$
- $[b]_R = \{b, bb, bab, \ldots\}$ $(u^{-1}L = \emptyset)$

Define the goal

Basic property

If $u \in L$ then $[u]_R \subseteq L$
Let $F \subseteq Q$ be $\{[u] \in Q | u \in L\}$

- A representation defines a function from $\Sigma^* \to \{0, 1\}$
  $f(w) = 1$ iff $w \in L$
- Define a function from $\Sigma^* \to Q$
  We want $\delta(w) = q$ iff $w \in q$

Step 2

Recursive derivation

Algebraic property of the primitives

It is a right congruence:

$u \cong v \Rightarrow uw \cong vw$

$[u]_R \circ w = [uw]_R$

Derivation of δ

Left to right derivation:

- $\delta(\lambda) = [\lambda]_R = q_0$
- If $\delta(u) = q = [v]$ then $\delta(ua) = [va]_R$ if $[va]_R \in Q$

Language defined

If $\delta(w) = q$ and $q \in F$ then $w \in \hat{L}(K, L)$

DFA

A long route to a familiar destination

- This is just a deterministic automaton
- If $\delta(w) \in q$ then $w \in q$
- We will always undergenerate: $L \subseteq \hat{L}(K, L)$
- As K increases the language increases

Language class

If K is finite, then $\hat{L}(K, L)$ is regular
If L is regular and K is big enough then $\hat{L}(L, K) = L$
(Myhill-Nerode theorem)

Positive and negative results

How do we reconcile the positive results with the negative ones?

- Distributions of examples are helpful in this case
- Very sparse distributions are hard – we need some frequent examples.

Key question

Are the distributions that describe the child's PLD helpful or unhelpful?

Summary of regular learning

Derived DFA starting from a representational assumption.

Minimal DFAs are learnable because there is a bijection between the representational primitives and some objectively defined sets of strings of the language.

Define a set of primitives	Right congruence classes
Derivation relation	Transition function
Language class	Regular languages
Inference algorithms	Testing right congruence

Probabilistic data can substitute for queries.