Abstract States of a Language

Robert M. Keller
Harvey Mudd College
April 2013

Languages

- Recall that a language over a finite alphabet Σ is any set of strings from elements of the alphabet.

- The set of all such strings is designated Σ^*.

- So L is a language means $L \subseteq \Sigma^*$.

States of a Machine

- You might think of a state as something that captures aspects of previous behavior.
- This is true. However, more importantly:

- A state determines the possible **future** behaviors.

States of a Language

- We are used to thinking in terms of states of a *machine*.

- But languages also have a kind of states, which we call *abstract states*.

States of a Language

- Suppose $L \subseteq \Sigma^*$.

- Define a binary relation $\equiv_L$ on Σ^* as follows:
  - $x \equiv_L y$ means

    $\forall z \in \Sigma^* \; (xz \in L \leftrightarrow yz \in L)$

Example of $\equiv_L$

- Suppose L is $\{(01)^n \mid n \in N\}$, where N is the set of natural numbers.
- Thus $L = \{\varepsilon, 01, 0101, 010101, \ldots\}$

- Consider 01 vs. 0101
- Is $01 \equiv_L 0101$?
- Yes, provided

$$\forall z \in \Sigma^* \, (01z \in L \leftrightarrow 0101z \in L)$$

Example of $\equiv_L$, where $L = \{(01)^n \mid n \in N\}$

- ? $\forall z \in \Sigma^* \, (01z \in L \leftrightarrow 0101z \in L)$?
- If z is of the form $(01)^m$ for some m, then both 01z and 0101z are in L.
- If z is *not* of that form, then *neither* is in L.
- Therefore $01 \equiv_L 0101$.

Example of $\equiv_L$, where $L = \{(01)^n \mid n \in N\}$

- Now consider 00 and 11.
- Is it true that 00 $\equiv_L$ 11?
- ? $\forall z \in \Sigma^* \,(00z \in L \leftrightarrow 11z \in L)$?
- Neither 00 nor 11 $\in$ L. Moreover, there is no string z such that either 00z $\in$ L or 11z $\in$ L.
- Since the definition is based on $\leftrightarrow$, we do have 00 $\equiv_L$ 11.

Example of $\equiv_L$, where $L = \{(01)^n \mid n \in N\}$

- Finally consider 01 and 010.
- Is it true that $01 \equiv_L 010$?
- ? $\forall z \in \Sigma^* (01z \in L \leftrightarrow 010z \in L)$?
- Consider $z = \varepsilon$:
  $01\varepsilon = 01 \in L$ but $010\varepsilon = 010 \notin L$
- Therefore $\neg(01 \equiv_L 010)$.

Distinguishing Strings

- A string z such $xz \in L$ but $yz \notin L$ is said to *distinguish* x from y.

- So *not* $x \equiv_L y$ if there is *some* string that distinguishes x from y. In this case x and y are called "distinguishable".

- Equivalently, $x \equiv_L y$ if there is *no* string that distinguishes x from y. Such x and y are called "indistinguishable".

Exercise: Classify these pairs $\equiv_L$ or not (find a distinguishing string, if possible)

(L is $\{(01)^n \mid n \in N\}$)

- 01 vs. 010101
- ε vs. 0
- ε vs. 01
- 101 vs. 0101

Exercise:

- Is it true, for any L, and $x, y \in L$ that $x \equiv_L y$?

Properties of $\equiv_L$

- $\forall x \in \Sigma^* \; x \equiv_L x$ (reflexive)
  $\forall z \in \Sigma^* \; (xz \in L \leftrightarrow xz \in L)$

- $\forall x,y \in \Sigma^* \; x \equiv_L y \rightarrow y \equiv_L x$ (symmetric)
  If $\forall z \in \Sigma^* \; (xz \in L \leftrightarrow yz \in L)$
  then $\forall z \in \Sigma^* \; (yz \in L \leftrightarrow xz \in L)$

Properties of $\equiv_L$

- $\forall x,y,z \in \Sigma^* \ (x \equiv_L y \wedge y \equiv_L z) \rightarrow x \equiv_L z$ (transitive)

  If $\forall w \in \Sigma^* \ (xw \in L \leftrightarrow yw \in L)$
  and $\forall w \in \Sigma^* \ (yw \in L \leftrightarrow zw \in L)$
  then $\forall w \in \Sigma^* \ (xw \in L \leftrightarrow zw \in L)$

- These can all be shown using natural deduction.

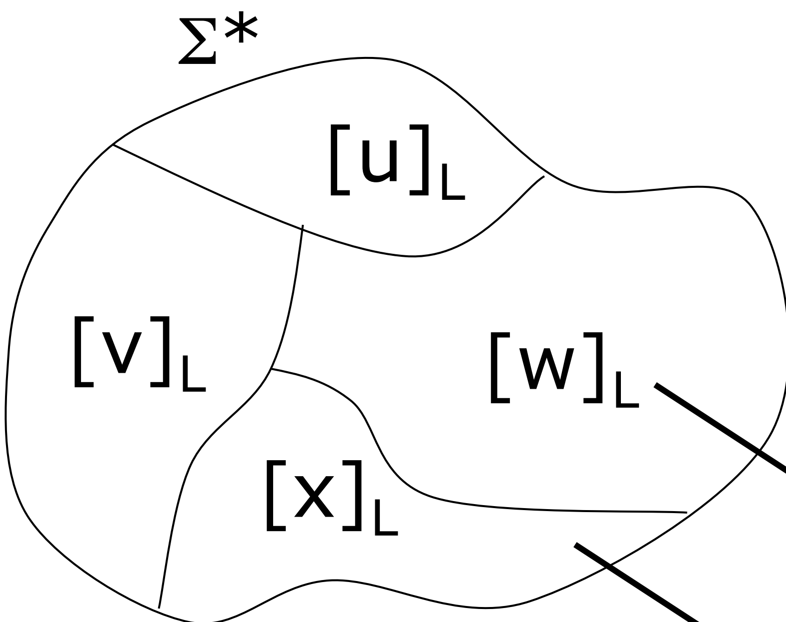
Properties of $\equiv_L$

- As $\equiv_L$ is reflexive, symmetric, and transitive, it is an **equivalence relation**.
- A known property of an equivalence relation on any set is that it **induces a partition** on that set.
- The elements of the partition are sets $[x]_L$ such that $[x]_L = \{y \in \Sigma^* \mid x \equiv_L y\}$.
- These sets are called **equivalence classes**.
- Note that $[x]_L = [y]_L$ iff $x \equiv_L y$.

Properties of a Partition on a Set

- A partition of S is any set of subsets of S, such that:
  - The subsets are pairwise mutually exclusive.
  - The union of the subsets is S.

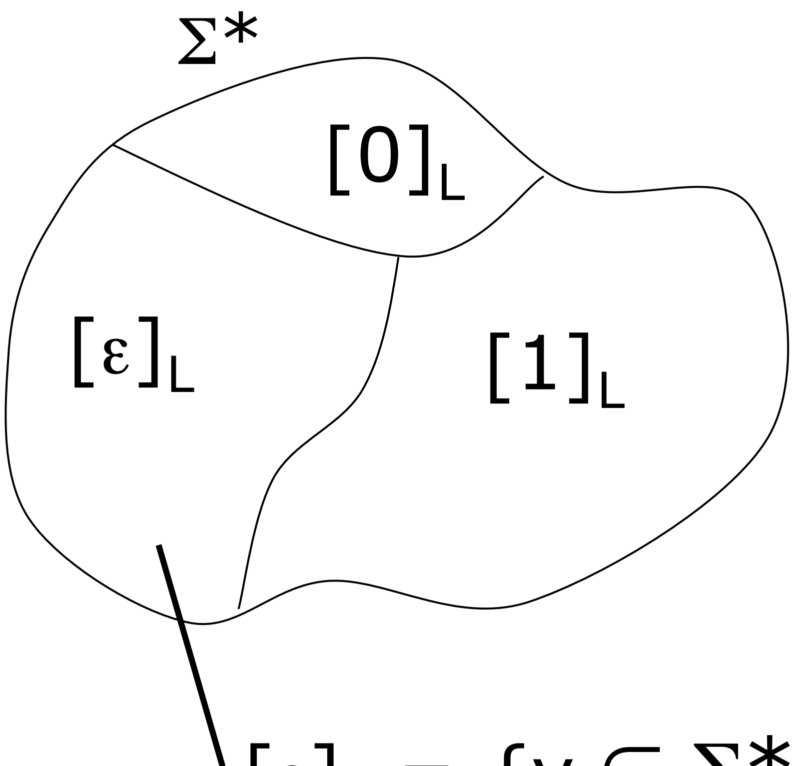
Showing how $\equiv_L$ partitions Σ^*



$[w]_L = \{y \in \Sigma^* \mid w \equiv_L y\}$

$[x]_L = \{y \in \Sigma^* \mid x \equiv_L y\}$

Partition for $L = \{(01)^n \mid n \in N\}$



$[\varepsilon]_L = \{\varepsilon, 01, 0101, \ldots\}$

$[0]_L = \{\varepsilon, 010, 01010, \ldots\}$

$[1]_L = \{1, 00, 10, 001, \ldots\}$

$[\varepsilon]_L = \{y \in \Sigma^* \mid \varepsilon \equiv_L y\}$

Terminology

- The “rank” of an equivalence relation is the number of equivalence classes.

- The rank can be finite or infinite.

Properties of $\equiv_L$

- $\equiv_L$ is not *just* an equivalence relation. It has the following additional property:
- $\forall \sigma \in \Sigma \; (x \equiv_L y \rightarrow x\sigma \equiv_L y\sigma)$
- This property is called **right-congruence** ("right" because x and y are "multiplied" on the right.)

Proof that $\equiv_L$ is a right congruence

- To show:
  $x \equiv_L y \rightarrow \forall \sigma \in \Sigma \ (x\sigma \equiv_L y\sigma)$
  - Suppose $x \equiv_L y$ and let $\sigma \in \Sigma$, to show $x\sigma \equiv_L y\sigma$.
  - From $x \equiv_L y$, the definition of $\equiv_L$ provides
    $\forall w \in \Sigma^* \ (xw \in L \leftrightarrow yw \in L)$. (++)
  - The definition of $\equiv_L$ has $x\sigma \equiv_L y\sigma$ iff
    $\forall z \in \Sigma^* \ ((xv)z \in L \leftrightarrow (yv)z \in L)$. (+++)
  - Since concatenation is associative, (+++) is the same as $\forall z \in \Sigma^* \ (x(vz) \in L \leftrightarrow y(vz) \in L)$, which follows directly from (++).

Some questions can be answered just knowing that $\equiv_L$ is a right congruence.

(L is $\{(01)^n \mid n \in N\}$)

a. ε vs. 01 Yes, by direct reasoning

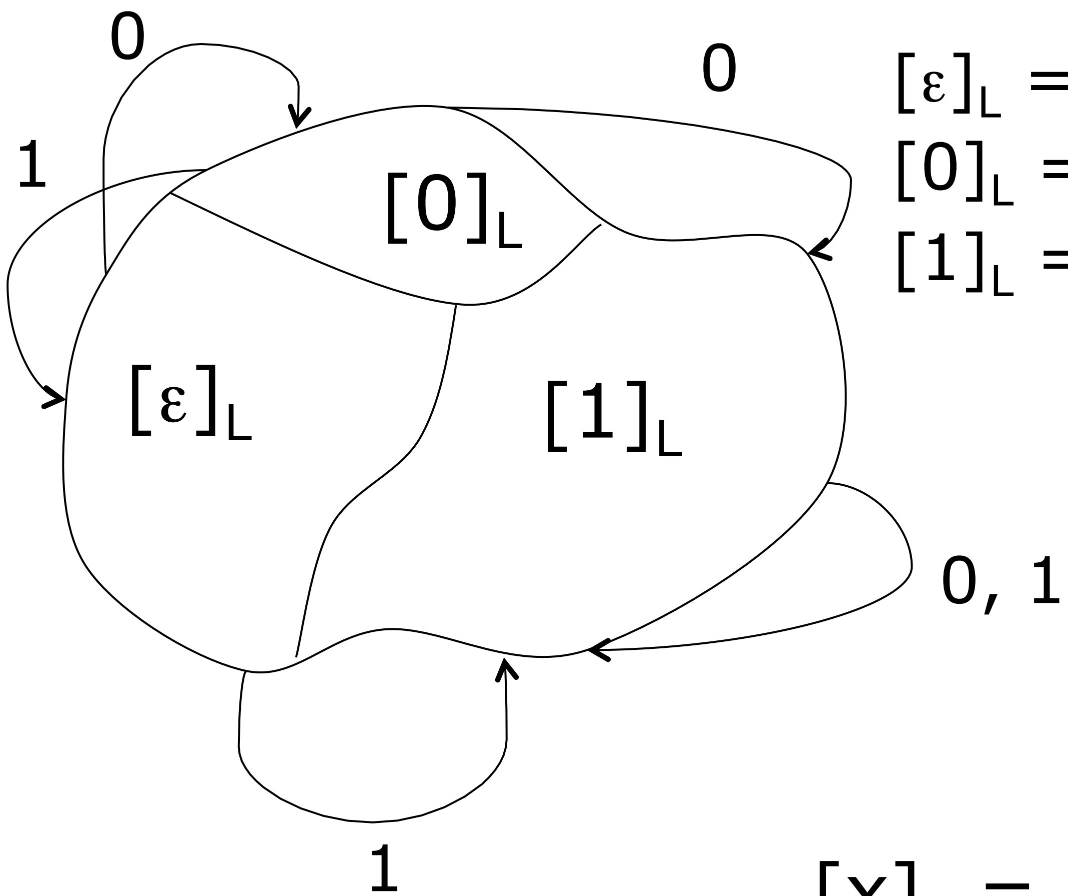
b. 01 vs. 0101 Yes, by right congruence & a.

c. ε vs. 1 No, obviously (not $\equiv$).

Equivalents of $\equiv_L$ being a right congruence

- $\forall \sigma \in \Sigma \ (x \equiv_L y \rightarrow x\sigma \equiv_L y\sigma)$ definition
- $\forall w \in \Sigma^* \ (x \equiv_L y \rightarrow xw \equiv_L yw)$ using induction
- $\forall w \in \Sigma^* \ (x \equiv_L y \leftrightarrow xw \equiv_L yw)$
  from above, because other direction trivial

Right Congruence $L = \{(01)^n \mid n \in N\}$



$[\varepsilon]_L = \{\varepsilon, 01, 0101, \ldots\}$

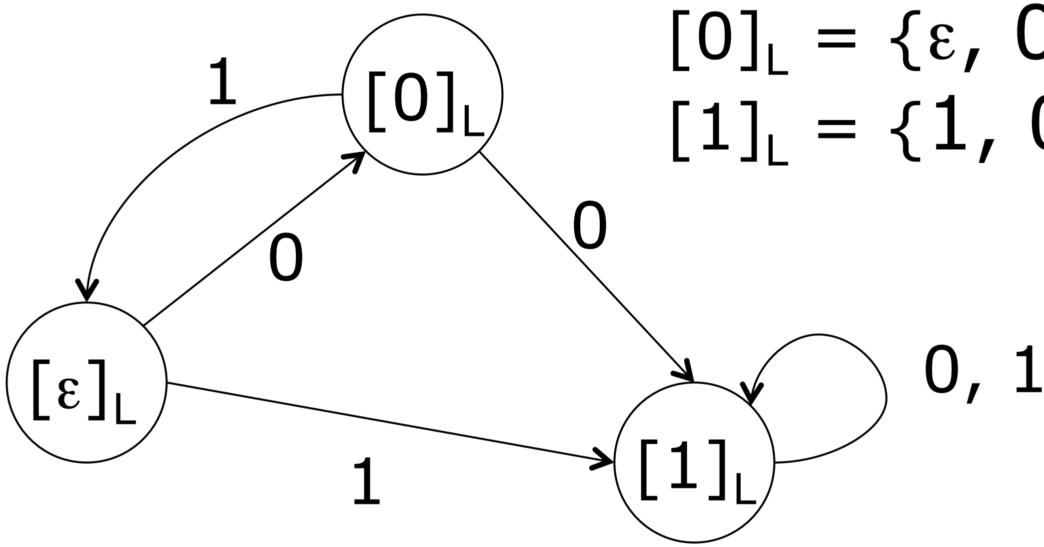
$[0]_L = \{\varepsilon, 010, 01010, \ldots\}$

$[1]_L = \{1, 00, 10, 001, \ldots\}$

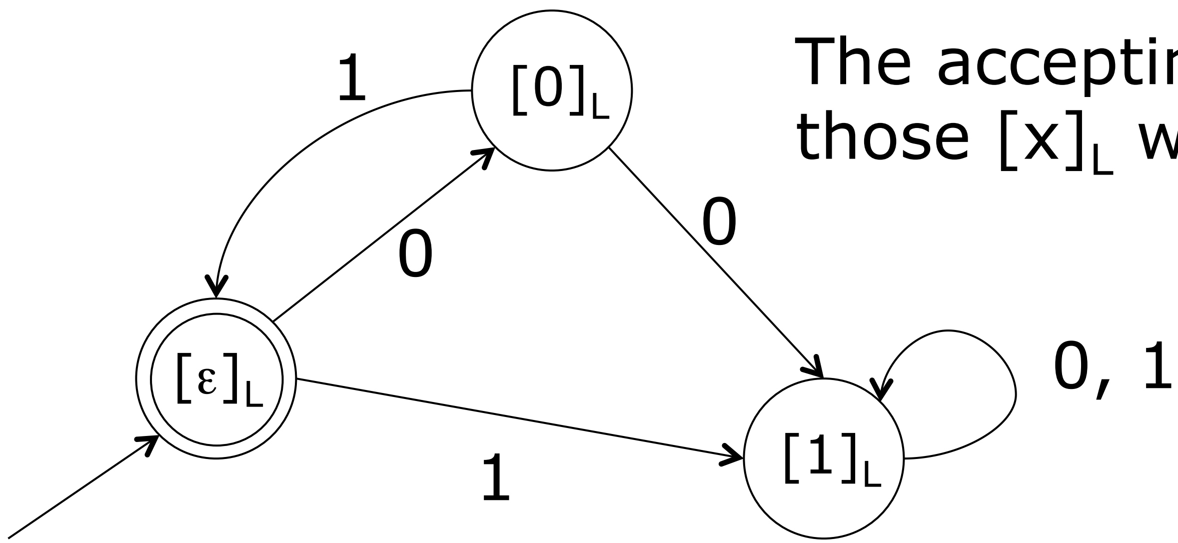
$[x]_L = \{y \in \Sigma^* \mid x \equiv_L y\}$

Abstract States of L *are* the Equivalence Classes of $\equiv_L$

$[\varepsilon]_L = \{\varepsilon, 01, 0101, \ldots\}$
$[0]_L = \{\varepsilon, 010, 01010, \ldots\}$
$[1]_L = \{1, 00, 10, 001, \ldots\}$



If L is regular, the abstract states of L determine a DFA accepting L.



The initial state is $[\varepsilon]_L$.

The accepting states are those $[x]_L$ where $x \in L$.

Conversely, if $\equiv_L$ has a **finite rank** partition, then L is regular.

- The following are the same:
  - The abstract states of L.
  - The equivalence classes of $\equiv_L$.
  - The states of a minimal DFA for L.

What does “minimal” mean?

- A machine is minimal iff no two different states are equivalent.

- In general, two states q, q′ are **equivalent** iff

  $$\forall w \in \Sigma^* \, (\delta(q, x) \in F \leftrightarrow \delta(q', x) \in F)$$

where δ is the **extended** state-transition function.

Extended State-Transition Function

- The **basic** state-transition function is

  $\delta: Q \times \Sigma \rightarrow Q$

- The **extended** function is

  $\delta: Q \times \Sigma^* \rightarrow Q$

  where $\delta(q, \varepsilon) = q$, and

$\forall x \in \Sigma^* \; \delta(q, \sigma x) = \delta(\delta(q, \sigma), x)$

(recursive definition, where the inner δ is basic).

Myhill-Nerode Theorem

- $L \subseteq \Sigma^*$ is a regular language iff
- L is the *union* of equivalence classes of a right congruence relation on L of **finite rank**.

- We need union here, because there may be more than one class corresponding to accepting states.

Notes on Myhill-Nerode

- It is proved by the comments leading up to the statement.

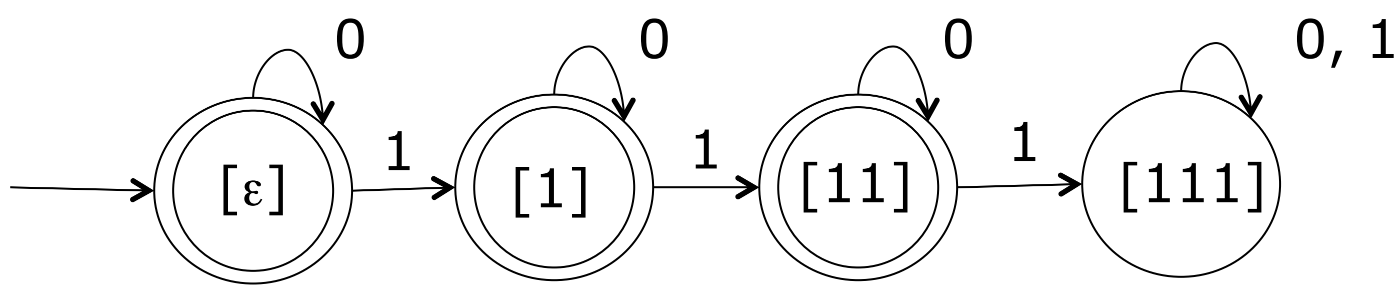
- If there is a right congruence relation as described, then:
  - The states of the DFA are the equivalence classes [x], where $x \in \Sigma^*$.
  - The transitions are defined by $\delta([x], \sigma) = [x\sigma]$
  - The initial state is $[\varepsilon]$.
  - The accepting states are [x] where $x \in L$.

Notes on Myhill-Nerode

- The transition function $\delta([x], \sigma) = [x\sigma]$ is well-defined, since if
  $[x] = [y]$, then
  $x \equiv y$, and thus for any $\sigma \in \Sigma$,
  $x\sigma \equiv y\sigma$, hence
  $[x\sigma] = [y\sigma]$.
- The initial state is $[\varepsilon]$.
- The accepting states are $[x]$ where $x \in L$.

Case where a union of more than one class is needed.

- Consider $L = \{x \in \{0, 1\}^* \mid x$ has at most 2 1's$\}$.
- The equivalence classes are: $[\varepsilon]$, $[1]$, $[11]$, $[111]$ and $L = [\varepsilon] \cup [1] \cup [11]$
- The DFA is



This DFA is minimal.

Non-Regularity Summarized

- A language L is regular iff $\equiv_L$ has finite rank.
- Thus L is not regular iff $\equiv_L$ has infinite rank.
- L is not regular iff there exists an *infinite* set of *mutually-distinguishable* strings.
- **In order to show non-regularity, we don’t have to exhibit the abstract states. We only need to infer there is such a set.**

Example of Non-Regularity

- Consider $L = \{0^n 1^n \mid n \in N\}$
- We claim that

  if $m \neq n$, then ***not*** $0^m \equiv_L 0^n$.

- We need only exhibit a z such that $0^m z \in L$ and $0^n z \notin L$.
- Such a distinguishing z is 1^m.
- As there is an infinite set of strings $\{0^n \mid n \in N\}$ that are pairwise distinguishable, L is not regular.

Example of Non-Regularity

- Consider
  $L = \{x \in \{0, 1\}^* \mid \#_0(x) = \#_1(x)\}$, where
  $\#_\sigma(x)$ means the number of σ in x.
- We claim that
  if $m \neq n$, then ***not*** $0^m \equiv_L 0^n$.

- 1^m distinguishes these strings.
- The rest of the argument on the previous slide then applies here as well.

Example of Non-Regularity

- Consider $L = \{ww \mid w \in \{0, 1\}^*\}$
- We claim that

  if $m \neq n$, then ***not*** $10^m \equiv_L 10^n$.

- We need only find an z such that $10^m z \in L$ and $10^n z \notin L$.
- Such a distinguishing z is 10^m.
- As there is an infinite set of strings $\{10^n \mid n \in N\}$ that are pairwise distinguishable, L is not regular.

Exercise: Regular or Non-Regular?

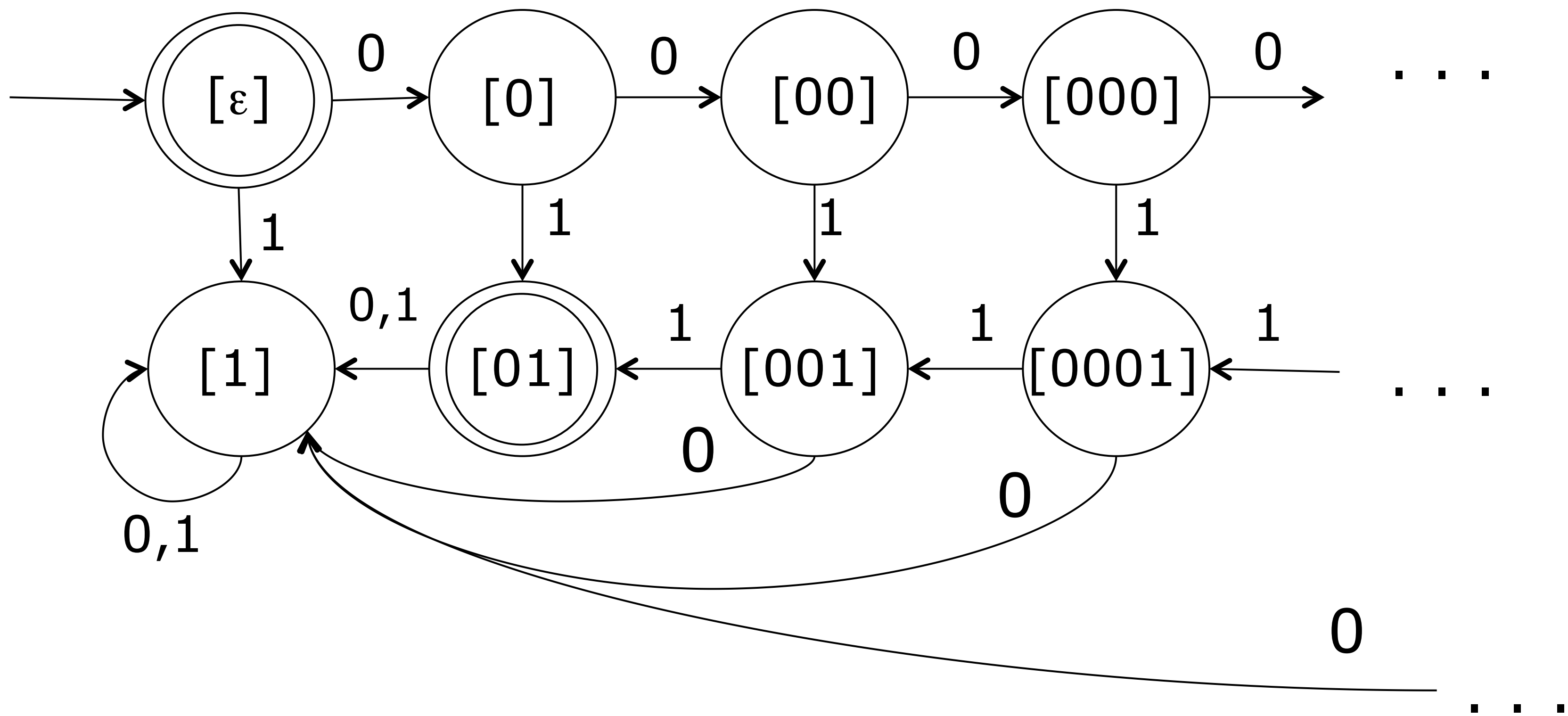
- $\{ww^R \mid w \in \{0, 1\}^*\}$ (even-length palindromes w^R is the reverse of w)
- $\{ww \mid w \in \{1\}^*\}$
- $\{1^n \mid n \text{ is even}\}$
- $\{1^n \mid n \text{ is a perfect square}\}$
- $\{1^n \mid n \text{ is prime}\}$
- $\{1^n \mid n \text{ is an even prime}\}$
- $\{w \in \{`(`, `)'\}^* \mid w \text{ is a well-balanced parenthesis string}\}$ i.e. (), (()), (()()), …

Can a state diagram be constructed for non-regular languages (if we wanted to)?

- We can construct a diagram that suggests the form of the machine, but it cannot be completed. The equivalence classes are abstract states.

- If we allow *recursion* in diagrams, certain non-regular languages can be constructed. (We will show this when discussing context-free grammars.)

Infinite Diagram for $\{0^n1^n \mid n \in N\}$



Derivatives of Languages and Regular Expressions

Robert M. Keller
Harvey Mudd College
April 2013

Derivative of a Language

- Let $L \subseteq \Sigma^*$ be a language.
- Let $x \in \Sigma^*$.

- Define $L/x = \{y \mid xy \in L\}$.

- L/x is called the **derivative** of L wrt x.

- Informally, L/x consists of members of L with any initial x “lopped off”. If x cannot be lopped of from a member, then it is not included.

Examples

- $\{01, 011\}/0 = \{1, 11\}$
- $\{01, 11\}/0 = \{1\}$
- $\{(01)^n \mid n \in N\}/0 = \{1(01)^n \mid n \in N\}$
- $\{(01)^n \mid n \in N\}/1 = \varnothing$
- $\{0^n 1^n \mid n \in N\}/0^m = \{0^{(n-m)}1^n \mid n \geq m\}$

Use of Derivatives

- Derivatives provide a method for contacting the abstract states of a language.

Claim

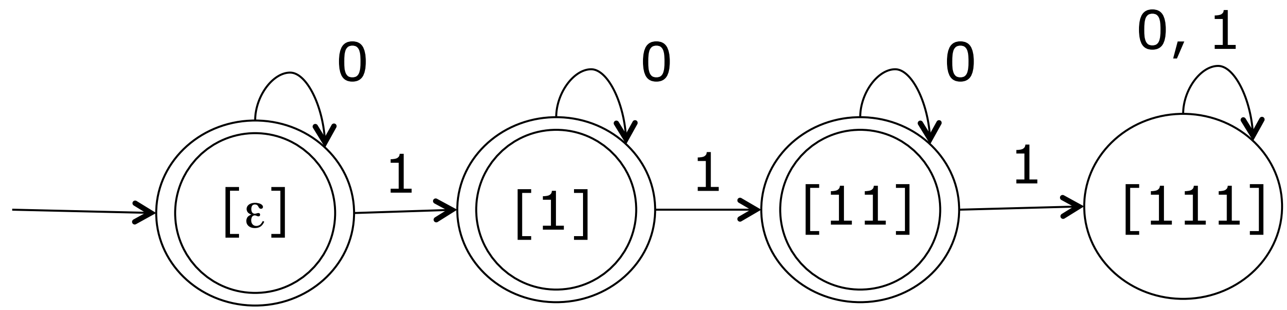
- If $L \subseteq \Sigma^*$ is regular then the set of derivatives $\{L/x \mid x \in \Sigma^*\}$ is finite.
- Justification: Consider the minimal DFA D accepting L.
- With each state q of D, define

  $$L(q) = \{x \mid \delta(q, x) \in F\},$$

  where F is the set of accepting states.
- So $L = L(q_0)$.
- Moreover, for each state q

  $$L(\delta(q, x)) = L(q)/x$$

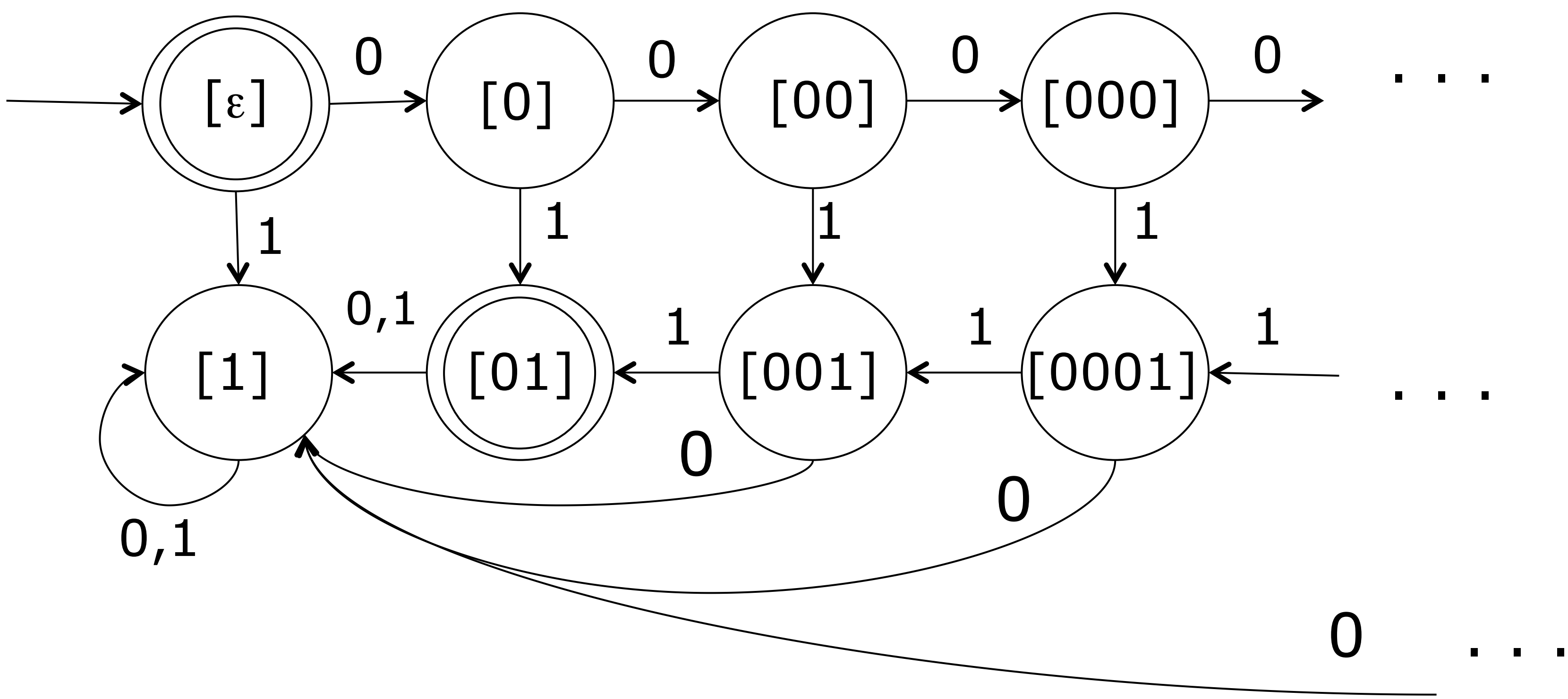
Example



- Consider again L = {x ∈ {0, 1}* x has at most 2 1's.
- The DFA was constructed earlier and is repeated above.
- $L([\varepsilon]) = L = L(\delta([\varepsilon], \varepsilon))$
- $L([1]) = L/1 = L(\delta([\varepsilon], 1))$
- $L([11]) = L/11 = L(\delta([\varepsilon], 11))$
- $L([111]) = \emptyset = L/111 = L(\delta([\varepsilon], 111))$ etc.

Derivatives of L = $\{0^n1^n \mid n \in N\}$

$L/\varepsilon = \{0^n1^n \mid n \in N\}$
$L/0^m = \{0^{n-m}1^n \mid n \geq m\}$
$L/0^m1^k = \{0^n1^{n-k} \mid n \geq k > 0\}$
$L/1 =$ everything else



Derivatives of Regular Expressions

- An amazing result is that the derivative concept can be extended to regular expressions, giving us a way to construct a DFA from a regular expression without going to an NFA first.

- For any regular expression R over Σ and any $\sigma \in \Sigma$ we can construct R/σ such that $L(R/\sigma) = L(R)/\sigma$.

Derivative Rules for Regular Expressions

- $\varnothing/\sigma = \varnothing$
- $\varepsilon/\sigma = \varnothing$
- $\sigma/\sigma = \varepsilon$
- $\sigma'/\sigma = \varnothing$ if $\sigma' \neq \sigma$
- $(R \cup S)/\sigma = (R/\sigma) \cup (S/\sigma)$
- $(RS)/\sigma = (R/\sigma)S$, if $\varepsilon \notin L(R)$
- $(RS)/\sigma = (R/\sigma)S \cup (S/\sigma)$, if $\varepsilon \in L(R)$
- $(R^*)/\sigma = (R/\sigma)R^*$

Note on the condition $\varepsilon \in L(R)$

- This condition is determinable *without* constructing a DFA, using the following rules:
  - not $\varepsilon \in L(\varnothing)$
  - $\varepsilon \in L(\varepsilon)$
  - not $\varepsilon \in L(\sigma)$
  - $\varepsilon \in L(R \cup S)$ iff $\varepsilon \in L(R)$ or $\varepsilon \in L(S)$
  - $\varepsilon \in L(RS)$ iff $\varepsilon \in L(R)$ and $\varepsilon \in L(S)$
  - $\varepsilon \in L(R^*)$

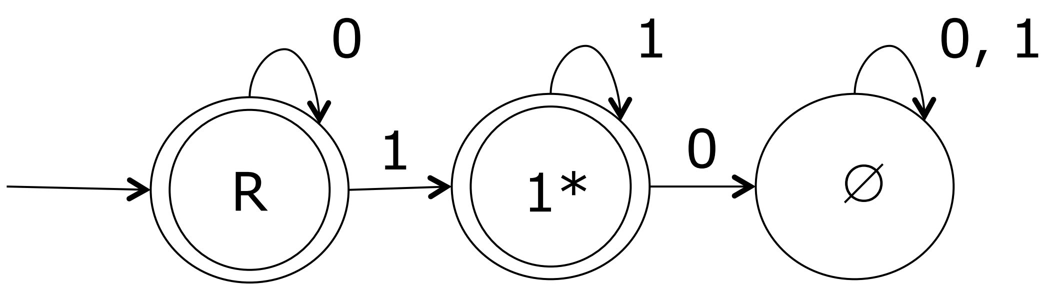
Examples of Derivatives of Res (apply the rules recursively)

- $0/0 = \varepsilon$
- $1/0 = \varnothing$
- $(0\cup1)/0 = \varepsilon$
- $01/0 = 1$
- $01/1 = \varnothing$
- $0^*/0 = 0^*$
- $1^*/0 = \varnothing$
- $(0\cup1)^*/0 = ((0\cup1)/0)(0\cup1)^* = (0/0\cup1/0)(0\cup1)^*$
  $= (\varepsilon \cup \varnothing)(0\cup1)^* = \varepsilon(0\cup1)^* = (0\cup1)^*$
- $(01)^*/0 = 1(01)^*$
- $(01)^*/1 = \varnothing$
- $(0\cup01)^*/0 = (\varepsilon\cup1)(0\cup01)^*$
- $(01\cup10)^*/0 = 1(0\cup01)^*$

Example: Using derivatives to construct a DFA

- Suppose $R = 0^* \cup 0^*11^*$. Then
- $R/0 = 0^* \cup 0^*11^* = R$
- $R/1 = 1^*$
- $R/10 = 1^*/0 = \emptyset$
- $R/11 = 1^*/1 = 1^* = R/1$
- At this point we have closure, so can diagram the DFA.
- R is the initial state.
- The accepting states are those having ε as an element.

DFA for 0* ∪ 0*11* constructed using derivatives of regular expressions

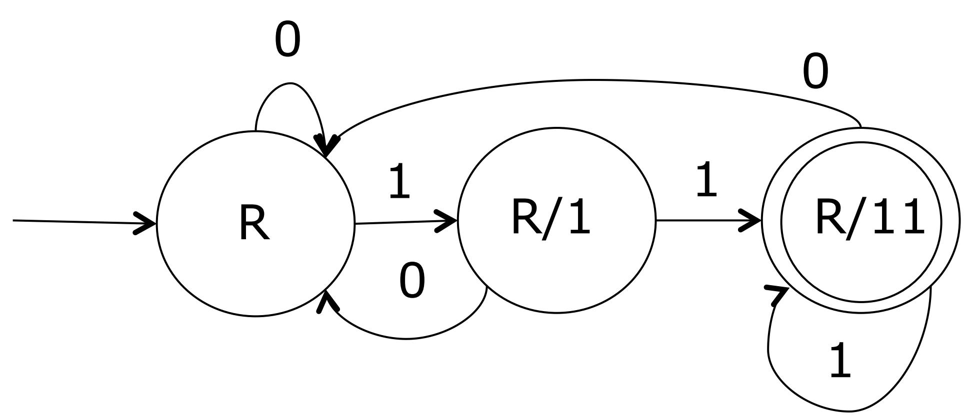


Caution

- Closure can sometimes be tricky to detect.
- Two regular expressions might be equivalent, but not identical.
- The same state is ideally used for both.

Example: Using derivatives to construct DFA

- Suppose $R = (0 \cup 1)^*11$.
- $R/0 = (0 \cup 1)^*11 = R$
- $R/1 = (0 \cup 1)^*11 \cup 1$ (note: 2nd case of rule)
- $R/10 = (R/1)/0 = R$
- $R/11 = (R/1)/1 = (0 \cup 1)^*11 \cup 1 \cup \varepsilon$
- $R/110 = R$
- $R/111 = R/11$



Regular Expression Identities

- $R \cup S = S \cup R$
- $R\ (S\ T) = (R\ S)\ T$
- $(R \cup S)T = RT \cup ST$
- $T(R \cup S) = TR \cup TS$
- $RR^* = R^*R$
- If $\varepsilon \in L(R)$ then $RR^* = R^*$
- $\varepsilon \cup R^* = R^*$
- $\varepsilon \cup RR^* = R^*$
- $(\varepsilon \cup R \cup R^2 \cup \ldots R^{n-1})\ (R^n)^* = R^*$, for every n
- $\varepsilon R = R$
- $R\varepsilon = R$
- $(R^*)^* = R^*$
- $R^*R^* = R^*$
- $R \cup R^* = R^*$
- $(R \cup S)^* = (R^*S)^*$
- $(R \cup S)^* = (RS^*)^*$
- $R(SR)^* = (RS)^*R$
- $\varnothing^* = \varepsilon$

How to Prove RE Identities

- Pretend the regular expression variables are letters of an alphabet.

- Prove the corresponding DFA for the left- and right-hand sides equivalent.

Example: $R(SR)^* = (RS)^*R$

- DFA's for $r(sr)^*$ and $(rs)^*r$

Arden’s Rule

- The smallest **solution** for X of the RE **equation**
  $$X = B \cup AX$$
  is $X = A^*B$.

If $\varepsilon \notin L(A)$ then the solution is unique.

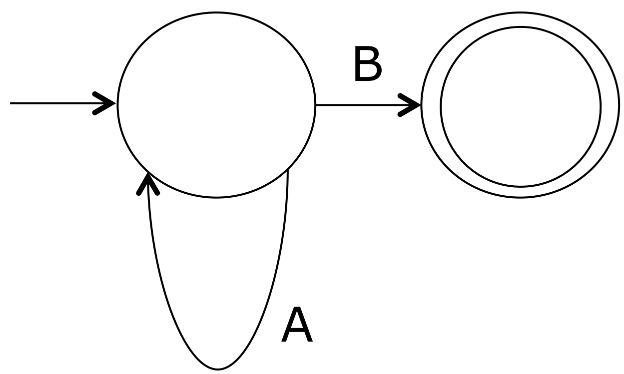
- Intuition: $X = B \cup AX = B \cup A(B \cup AX)$
  $= B \cup AB \cup A^2X = B \cup AB \cup A^2(B \cup AX)$
  $= B \cup AB \cup A^2B \cup A^3X = \ldots \subseteq A^*B$

- Arden’s rule is implicit in the derivation of a regular expression from a DFA

Arden's Rule Illustrated

$$X = B \cup AX$$

NFA accepting X



$$X = A^*B$$

This is an example of a "fixed point" theorem.

Alternate Conversion of DFA to RE

- Use Arden’s Rule + Gaussian Elimination Pattern

Non-Identities

- $(R \cup S)^* = R^* \cup S^*$

Regular Expressions in Software Tools

Utility	Regular Expression Type
vi	Basic
sed	Basic
grep	Basic
csplit	Basic
dbx	Basic
dbxtool	Basic
more	Basic
ed	Basic
expr	Basic
lex	Basic
pg	Basic
nl	Basic
rdist	Basic
awk	Extended
nawk	Extended
egrep	Extended
EMACS	EMACS Regular Expressions
PERL	PERL Regular Expressions

cf. http://www.grymoire.com/Unix/Regular.html

Example, using program egrep

- egrep =
  “Extended Global Regular Expressions Print”

- (This is a Unix program. Windows users can use it in Cygwin.)

- This program identifies **lines** in a file that **contain** a match a regular expression on the command line, e.g. to match lines containing the literal letter “x”

```
egrep x  /usr/share/dict/propernames
```

A sample file: /usr/share/dict/propernames

```
$ head /usr/share/dict/propernames
Aaron
Adam
Adlai
Adrian
Agatha
Ahmed
Ahmet
Aimee
Amy
Ami

# head is first 10
```

```
$ egrep x  /usr/share/dict/propernames
Alex
Alexander
Alexis
Axel
Felix
Lex
Marnix
Max
Rex
Roxana
Roxane
Roxanne
Roxie
```

Match lines containing “aa”

```
$ egrep aa /usr/share/dict/propernames
Isaac
Lievaart
Maarten
Raanan
Saad
Sjaak
```

Use | for Union

- Because | is significant to the operating system (for “piping”), regular expressions using it must be *quoted*.

```
$ egrep 'az|za' /usr/share/dict/propernames
Elizabeth
Hazel
Kazuhiro
Liza
Ozan
Suzan
Suzanne
```

Anchor Characters ^ and $

Pattern	Matches
^A	"A" at the beginning of a line
A$	"A" at the end of a line
A^	"A^" anywhere on a line
$A	"$A" anywhere on a line
^^	"^" at the beginning of a line
$$	"$" at the end of a line

Example: Lines ending in “ay”

```
$ egrep ay$  /usr/share/dict/propernames
Clay
Fay
Jay
Kay
Lindsay
Murray
Ray
Sanjay
Vijay
```

Wild Card .

- . by itself matches any single character except end-of-line

Example: 4-letter lines beginning with “A”

```
$ egrep ^A...$  /usr/share/dict/propernames
Adam
Alan
Alex
Amir
Amos
Andy
Anna
Anne
Arne
Axel
```

Use [ ] to enumerate several characters

```
$egrep 'of[aeiou]' /usr/share/dict/propernames
Christofer
Hirofumi
Sofia
Sofoklis

# Here we want ‘of’, but only if followed by vowel
```

Matching Ranges of Characters

[A-Z]
[a-z]
[0-9]
[A-Za-z0-9_]

Iterators

? matches 0 or 1 instances of what comes before

+ matches 1 or more instances

* matches 0 or more instances

For further info

http://en.wikipedia.org/wiki/Regular_expression

http://www.grymoire.com/Unix/Regular.html

http://www.regular-expressions.info/reference.html

$man egrep

$man regex