Equivalence relation and partitions

An *equivalence relation* on a set X is a relation which is reflexive, symmetric and transitive

A *partition* of a set X is a set P of *cells* or *blocks* that are subsets of X such that

1. If $C \in P$ then $C \neq \emptyset$

2. If $C_1, C_2 \in P$ and $C_1 \neq C_2$ then $C_1 \cap C_2 = \emptyset$

3. If $a \in X$ there exists $C \in P$ such that $a \in C$

Equivalence relation and partitions

If R is an equivalence relation on X, we define the *equivalence class* of $a \in X$ to be the set $[a] = \{b \in X \mid R(a, b)\}$

Lemma: $[a] = [b]$ *iff* $R(a, b)$

Theorem: *The set of all equivalence classes form a partition of* X

We write X/R this set of equivalence classes

Example: X is the set of all integers, and $R(x, y)$ is the relation “3 divides $x - y$”. Then X/R has 3 elements

Equivalence Relations

Example: on $X = \{1,2,3,4,5,6,7,8,9,10\}$ the relation $x \equiv y$ defined by

$$3 \ \ divides \ \ x - y$$

is an equivalence relation

We can now form $X/\equiv$ which is the set of all *equivalence classes*

$$[1] = [4] = \{1,4,7,10\} \qquad [2] = [8] = \{2,5,8\} \qquad [3] = \{3,6,9\}$$

This set is the *quotient* of X by the relation $\equiv$

Equivalence relations on states

$A = (Q, \Sigma, \delta, q_0, F)$ is a *DFA*

If R is an equivalence relation on Q, we say that R is *compatible* with A iff

(1) $R(q_1, q_2)$ implies $R(\delta(q_1, a), \delta(q_2, a))$ for all $a \in \Sigma$

(2) $R(q_1, q_2)$ and $q_1 \in F$ implies $q_2 \in F$

Equivalence relations on states

(1) means that if q_1, q_2 are in the same block then so are $q_1.a$ and $q_2.a$ and this for all a in Σ

(2) says that F can be written as an union of some blocks

We can then define $\delta/R([q], a) = [q.a]$ and $[q] \in F/R$ iff $q \in F$

Equivalence relations on states

Theorem: *If R is a compatible equivalence relation on A then we can consider the DFA $A/R = (Q/R, \Sigma, \delta/R.[q_0], F/R)$ and we have $L(A/R) = L(A)$*

Proof: By induction on x we have

$$\hat{\delta}([q], x) = [\hat{\delta}(q, x)]$$

and then $[q_0].x \in F/R$ iff $q_0.x \in F$

Notice that A/R has *fewer* states than A

Equivalence of States

Theorem: *Let* $A = (Q, \Sigma, \delta, q_0, F)$ *be a DFA. The relation* $R(q_1, q_2)$ *defined by for all* $w \in \Sigma^*$ *we have* $q_1.w \in F$ *iff* $q_2.w \in F$ *is a* compatible *equivalence relation*

It is essential for this Theorem that A is a *DFA*

We say simply that q_1 and q_2 are *equivalent* if we have $R(q_1, q_2)$

Corollary: *We have* $L(A) = L(A/R)$

We shall prove that, *provided* all the states of A are accessible, the DFA A/R depends only of $L(A)$ and not of A

Algorithm for Computing R

Let us compute R for

	a	b
→0	1	2
∗1	3	4
∗2	4	3
3	5	5
4	5	5
∗5	5	5

We know $\{0,1\},\{0,2\},\{0,5\},\{1,3\},\{2,3\},\{3,5\},\{4,1\},\{4,2\},\{4,5\}$ are non equivalent pairs

Algorithm for Computing R

	a	b
$\{0,3\}$	$\{1,5\}$	$\{2,5\}$
$\{1,5\}$	$\{3,5\}$	$\{4,5\}$
$\{2,5\}$	$\{4,5\}$	$\{3,5\}$

Thus, $\{0,3\},\{1,5\},\{2,5\}$ are non equivalent

It is convenient to build at the same time a triangular table of pair of states, as indicated in the text book

Algorithm for Computing R

Let us compute R for

	a
$\rightarrow$0	1
$*$1	2
2	3
3	4
$*$4	5
5	0

We have $\{0,3\} \rightarrow \{1,4\} \rightarrow \{2,5\} \rightarrow \{0,3\}$ and $\{0,2\} \rightarrow \{1,3\}$

We get $\{\{0,3\},\{1,4\},\{2,5\}\}$

Functional program for R

```
equal (e1,f1) (e2,f2) =
 (e1 == e2 && f1 == f2) || (e1 == f2 && f1 == e2)

data Answer = Equiv | L String | R String
   deriving (Eq,Show)
```

Functional program for R

```
equiv (cs,delta,final) e1 e2 = eq e1 e2 ([],"")
 where
  eq q1 q2 (l,s) = if q1 == q2 then Equal else
   case (final q1,final q2) of
    (True,False) -> L  (reverse s)
    (False,True) -> R (reverse s)
    _ -> if or (map (equal (q1,q2)) l) then Equal
          else combine
                 (map (\ c ->
                     eq (delta q1 c) (delta q2 c)
                        ((q1,q2):l,c:s)) cs)
```

Functional program for R

```
combine (Equal:bs) = combine bs
combine (a:_) = a
combine [] = Equal
```

Functional program for R

```
data Q = A | B | C | D  deriving (Eq,Show)

delta A '0' = B   delta A '1' = A
delta B '0' = B   delta B '1' = C
delta C '0' = C   delta C '1' = D
delta D _ = D

final C = True    final D = True
final _ = False

test1 = equiv ("01",delta,final) C D
```

The Quotient Construction

We are now going to show that A/R does not depend on A, but only on $L = L(A)$, provided all states in A are accessible

This will show that the minimal DFA for a regular language is unique (up to renaming of the states)

The Quotient Construction

Give L we define $u \equiv_L v$ iff $u \setminus L = v \setminus L$

Another formulation of the Myhill-Nerode theorem is

Theorem: *A language* $L \subseteq \Sigma^*$ *is* regular *iff* $\equiv_L$ *has only a* finite *number of equivalence classes*

Notice that $u \equiv_L v$ iff for all w we have $uw \in L$ iff $vw \in L$

Myhill-Nerode Theorem

If L is a regular language and $L = L(A)$ where $A = (Q, \Sigma, \delta, q_0, F)$ and all states in Q are accessible and S is the set of abstract states of L we know that the map

$f : Q \rightarrow S$

$q_0.u \longmapsto u \setminus L$

is well-defined and *surjective*

In particular $|Q| \geqslant |S|$

Myhill-Nerode Theorem

Assume $q_1 = q_0.u_1,\ q_2 = q_0.u_2$

We have $f(q_1) = f(q_2)$ iff $u_1 \setminus L = u_2 \setminus L$ iff for all $w \in \Sigma^*$

$q_1.w \in F \leftrightarrow q_2.w \in F$

which is precisely the equivalence for building the *minimal automaton*

Thus $|Q/\equiv| = |S|$

The Subset Construction

Theorem: *A DFA that recognizes* $L = L((0+1)^*01(0+1)^*)$ *has at least* 3 *states.*

We build a minimal DFA for this languages. It has 3 states. Hence *all* DFA that recognizes the same language has at least 3 states!

We can also show that $\equiv_L$ has at least 3 equivalence classes

The algorithm for the quotient construction we have shown is $O(n^2)$ where n number of states. Hopcroft has given a $O(n\ log\ n)$ algorithm for this (using partition instead of equivalence relation)

Accessible states

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

A state $q \in Q$ is *accessible* iff there exists $x \in \Sigma^*$ such that $q = q_0.x$

Let Q_0 be the set of accessible states, $Q_0 = \{q_0.x \mid x \in \Sigma^*\}$

Theorem: *We have $q.a \in Q_0$ if $q \in Q_0$ and $q_0 \in Q_0$. Hence we can consider the automaton $A_0 = (Q_0, \Sigma, \delta, q_0, F \cap Q_0)$. We have $L(A) = L(A_0)$*

In particular $L(A) = \emptyset$ if $F \cap Q_0 = \emptyset$.

Accessible states

Actually we have $L(A) = \emptyset$ *iff* $F \cap Q_0 = \emptyset$ since if $q.x \in F$ then $q.x \in F \cap Q_0$

Implementation in a functional language: we consider automata on a finite collection of characters given by a list cs

An automaton is given by a parameter type a with a transition function and an initial state

Accessible states

```
import List(union)

isIn as a = or (map ((==) a) as)
isSup as bs = and (map (isIn as) bs)

closure :: Eq a => [Char] -> (a -> Char -> a) -> [a] -> [a]

closure cs delta qs =
 let qs’ = qs >>= (\ q -> map (delta q) cs)
 in if isSup qs qs’ then qs
     else closure cs delta (union qs qs’)
```

Accessible states

```
accessible :: Eq a => [Char] -> (a -> Char -> a) -> a -> [a]

accessible cs delta q = closure cs delta [q]

-- test emptyness on an automaton

notEmpty :: Eq a => ([Char],a-> Char -> a,a,a->Bool) -> Bool

notEmpty (cs,delta,q0,final) = or (map final (accessible cs delta q0))
```

Accessible states

```
data Q = A | B | C | D | E
 deriving (Eq,Show)

delta A '0' = A      delta A '1' = B
delta B '0' = A      delta B '1' = B
delta C _ = D
delta D '0' = E      delta D '1' = C
delta E '0' = D      delta E '1' = C

as = accessible "01" delta A

test = notEmpty ("01",delta,A,(==) C)
```

Accessible states

Optimisation

```
import List(union)

isIn as a = or (map ((==) a) as)
isSup as bs = and (map (isIn as) bs)

Closure :: Eq a => [Char] -> (a -> Char -> a) -> [a] -> [a]
```

Accessible states

```
closure cs delta qs =  clos ([],qs)
 where
  clos (qs1,qs2) =
    if qs2 == [] then qs1
      else let qs = union qs1 qs2
               qs’ = qs2 >>= (\ q -> map (delta q) cs)
               qs’’ = filter (\ q -> not (isIn qs q)) qs’
           in clos (qs,qs’’)
```

Automatic Theorem Proving

If $\Sigma = \{a, b\}$ we have

$E = \psi(E) + a(a \setminus E) + b(b \setminus E)$

and hence $E = F$ iff

$\psi(E) = \psi(F)$

$a \setminus E = a \setminus F$

$b \setminus E = b \setminus F$

Automatic Theorem Proving

Given $E = (a^2 + a^3)^*$ what is the automaton of abstract states of E?

This gives an automatic way to prove that any number $\geqslant 2$ is a sum of 2s and 3s

One can prove automatically $a(ba)^* = (ab)^*a$ or $a^*(b + ab^*) \neq b + aa^*b^*$

One finds a counterexample to $(a + b)^* = a^* + b^*$

The Pigeonhole Principle

An important reasoning technique (see Wikipedia)

"If you have more pigeon than pigeonholes then there is at least one pigeonhole with two pigeons"

If $f : X \rightarrow Y$ and $|X| > |Y|$ then f is not injective and there exist two distinct elements with the same image

The Pigeonhole Principle

Often used to show the existence of an object without building this object explicitely

Example: in a room with at least 13 people, at least two of them are born the same month (maybe of different years). We know the existence of these two people, maybe without being able to know exactly who they are.

The Pigeonhole Principle

Example: In London, there are at least two people with the same number of hairs on their heads (assuming no one has more than 1000000 hairs on his head)

For a nice discussion, see

http://www.cs.utexas.edu/users/EWD/transcriptions/EWD09xx/EWD980.html

Other formulation: if we have a bag of numbers, the maximum value is greater than the average value

How to prove that a language is not regular?

In a NFA with N states, any path

$$q_0 \stackrel{a_1}{\rightarrow} q_1 \stackrel{a_2}{\rightarrow} q_2 \rightarrow \ldots q_{n-1} \stackrel{a_n}{\rightarrow} q_n$$

contains a loop as soon as $n \geqslant N$

Indeed, we should have $i < j$ with $q_i = q_j$. We apply the Pigeonhole Principle.

This works for NFA as well as for DFA

How to prove that a language is not regular?

Let Σ be $\{a, b\}$

Let L be the language $\{a^n b^n \mid n \geqslant 0\}$

We show that L is *not* regular

We assume that $L = L(A)$ for a NFA A and we derive a contradiction

How to prove that a language is not regular?

Let N be the number of states of A

Let $k \geqslant N$ and $w = a^k b^k \in L$

So there is an accepting path $q_0 \xrightarrow{w} q \in F$ and since we have only N states we know that there is a loop "at the beginning": we can write $w = xyz$ with $|xy| \leqslant N$ and

$$q_0 \xrightarrow{x} s \xrightarrow{y} s \xrightarrow{z} q \in F$$

How to prove that a language is not regular?

z is of the form $a^{k-m}b^k$ with $m = |xy|$

We have then an accepting path for xz

$$q_0 \xrightarrow{x} s \xrightarrow{z} q \in F$$

and since y has to be of the form $a^l,\ l > 0$ then xz is of the form $a^{k-l}b^k$

Since $a^{k-l}b^k \notin L$ we have a contradiction: xz cannot have an accepting path.

The Pumping Lemma

Theorem: *If* L *is a regular language, there exists* n *such that if* $w \in L$ *and* $n \leqslant |w|$ *then we can write* $w = xyz$ *with* $y \neq \epsilon$ *and* $|xy| \leqslant n$ *and for all* $k \geqslant 0$ *we have* $xy^k z \in L$.

The Pumping Lemma

Proof: We have a NFA A such that $L = L(A)$. Let n be the number of states of A. Any path in A of length $\geqslant n$ has a loop. We can consider that $w = a_1 \dots a_l$ defines a path with a loop

$$q_0 \xrightarrow{x} q \xrightarrow{y} q \xrightarrow{z} q_l$$

with q_l in F and $y \neq \epsilon$ and $|xy| \leqslant n$ such that $w = xyz \in L(A)$ Then we have

$$q_0 \xrightarrow{x} q \xrightarrow{y^k} q \xrightarrow{z} q_l$$

for each k and hence xy^kz in L

The pumping lemma

For instance $L_{eq} \subseteq \{0,1\}^*$ set of words with an equal number of 0 and 1 is not regular.

Otherwise, we have n as given by the pumping lemma.

We have $0^n1^n \in L_{eq}$ and hence

$0^n1^n = xyz$

with $|xy| \leqslant n,\ \ y \neq \epsilon$ and $xy^kz \in L_{eq}$ for all k.

But then we have $y = 0^q$ for some $q > 0$ and we have a contradiction for $k \neq 1$

The pumping lemma

Let L be the language of *palindromes* words x such that $x = x^R$ then L is *not* regular

Otherwise, we have n as given by the pumping lemma.

We have $0^n10^n \in L$ and hence

$0^n10^n = xyz$

with $|xy| \leqslant n, \ y \neq \epsilon$ and $xy^kz \in L$ for all k.

But then we have $y = 0^q$ for some $q > 0$ and we have a contradiction for $k \neq 1$

The pumping lemma

Another proof that $L_{eq} \subseteq \{0,1\}^*$ is not regular is the following.

Assume L_{eq} to be regular then $L_{eq} \cap L(0^*1^*)$ would be regular, but this is

$\{0^n 1^n \mid n \geqslant 0\}$

which we have seen is *not* regular.

Hence L_{eq} is not regular.

How to prove that a language is not regular?

Let L be the language $\{a^n b^n \mid n \geqslant 0\}$

Theorem: L *is not regular*

However there is a simple machine with infinitely many states that recognizes L

The Pumping Lemma is connected to the "finite memory" of FA

How to prove that a language is not regular?

For the examples

$L = \{0^n1^m \mid n \geqslant m\}$

$L' = \{0^n1^m \mid n \neq m\}$

the Pumping Lemma does not seem to work

We can use the closure properties of regular languages

The Pumping Lemma is not a Necessary Condition

If $L = \{b^k c^k \mid k \geqslant 0\}$ then L is *not* regular

If we consider $L_1 = a^+ L \cup (b+c)^*$ then L_1 is *not* regular: if L_1 is regular then so is $a^+ L$ (by intersection with the complement of $(b+c)^*$) and then so is L (by image under the morphism $f(a) = \epsilon,\ f(b) = b,\ f(c) = c$)

However *the Pumping Lemma applies to* L_1 with $n = 1$

This shows that, contrary to Myhill-Nerode's Theorem, the Pumping Lemma is not a necessary condition for a language to be regular

Applying the Pumping Lemma

$L = \{0^n 1^{2n} \mid n \geqslant 0\}$ is not regular

Proof: Assume that L is regular. By the Pumping Lemma there exists N such that if $w \in L$ and $N \leqslant |w|$ then we can write $w = xyz$ with $|xy| \leqslant N$ and $y \neq \epsilon$ and $xy^k z \in L$ for all k.

Take $w = 0^N 1^{2N}$. We have $N \leqslant |w|$ and $w \in L$. So we can write $w = xyz$ with $|xy| \leqslant N$ and $y \neq \epsilon$ and $xy^k z \in L$ for all k. Since $w = 0^N 1^{2N}$ and $y \neq \epsilon$ we have $y = 0^p$ for some $p > 0$. But then $xy \notin L$, contradiction. So L is not regular. Q.E.D.

Other proof with Myhill-Nerode: $0^k 1 \setminus L = \{1^{2k-1}\}$, infinitely many abstract states.