Warshall's algorithm

See *Floyd-Warshall* algorithm on Wikipedia

The Floyd-Warshall algorithm is a graph analysis algorithm for finding shortest paths in a weigthed, directed graph

Warshall algorithm finds the transitive closure of a directed graph

Warshall's algorithm

We have a graph with n nodes $1, 2, \ldots, n$

We define $E_{ij} = 1$ iff there is an edge $i \rightarrow j$

$E_{ij} = 0$ if there is no edge from i to j

We define $E^1_{ij} = E_{ij}$ and

$E^{k+1}_{ij} = E^k_{ij} \vee E^k_{ik} E^k_{kj}$

Then $E^k_{ij} = 1$ iff there exists a path $i \rightarrow i_1 \cdots \rightarrow i_l \rightarrow j$ with $i_1, \ldots, i_l$ all $< k$

Warshall's algorithm

This is best implemented with a fixed array of $n \times n$ booleans

For $k = 1$ to n

$E_{ij} := E_{ij} \vee E_{ik}E_{kj}$

Floyd's algorithm

Now E_{ij} is a positive number (the *cost* or the *distance* of going from i to j; it is ∞ if there is no edge from i to j).

For $k = 1$ to n

$$E_{ij} := min(E_{ij}, E_{ik} + E_{kj})$$

Regular expression

Now E_{ij} is a regular expression, and we compute *all* possible paths from i to j. We initialize by $E_{ij} := E_{ij}$ if $i \neq j$ and $E_{ii} := \epsilon + E_{ii}$.

For $k = 1$ to n

$$E_{ij} := E_{ij} + E_{ik} E_{kk}^{*} E_{kj}$$

Regular expression

For the automata with accepting state 2 and defined by

$$1.0 = 2,\ 1.1 = 1,\ 2.0 = 2.1 = 2$$

We have $E_{11} = \epsilon + 1,\ E_{12} = 0,\ E_{21} = \emptyset,\ E_{22} = \epsilon + 0 + 1$

Regular expression

Then the first step is

$$E_{11} = \epsilon + 1 + (\epsilon + 1)(\epsilon + 1)^*(\epsilon + 1) = 1^*$$

$$E_{12} = 0 + (\epsilon + 1)(\epsilon + 1)^*0 = 1^*0$$

$$E_{21} = \emptyset + \emptyset(\epsilon + 1)^*(\epsilon + 1) = \emptyset$$

$$E_{22} = \epsilon + 0 + 1 + \emptyset(\epsilon + 1)^*0 = \epsilon + 0 + 1$$

Regular expression

The second step is

$$E_{11} = 1^* + 1^*0(\epsilon + 0 + 1)^*\emptyset = 1^*$$

$$E_{12} = 1^*0 + 1^*0(\epsilon + 0 + 1)^*(\epsilon + 0 + 1) = 1^*0(0 + 1)^*$$

$$E_{21} = \emptyset + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*\emptyset = \emptyset$$

$$E_{22} = \epsilon + 0 + 1 + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*(\epsilon + 0 + 1) = (0 + 1)^*$$

Regular expression

In this way, we have seen *two* proofs of one direction of *Kleene's Theorem*: any regular language is recognized by a regular expression

The two proofs are

by solving an equation system and using Arden's Lemma

by using Warshall's algorithm

Algebraic Laws for Regular Expressions

$$E + (F + G) = (E + F) + G,\ E + F = F + E,\ E + E = E,\ E + 0 = E$$

$$E(FG) = (EF)G,\ E0 = 0E = 0,\ E\epsilon = \epsilon E = E$$

$$E(F + G) = EF + EG,\ (F + G)E = FE + GE$$

$$\epsilon + EE^* = E^* = \epsilon + E^*E$$

Algebraic Laws for Regular Expressions

We have also

$$E^* = E^*E^* = (E^*)^*$$

$$E^* = (EE)^* + E(EE)^*$$

Algebraic Laws for Regular Expressions

How can one prove equalities between regular expressions?

In usual algebra, we can "simplify" an algebraic expression by rewriting

$$(x + y)(x + z) \rightarrow xx + yx + xz + yz$$

For regular expressions, there is no such way to prove equalities. There is not even a complete finite set of equations.

Algebraic Laws for Regular Expressions

Example: $L^* \subseteq L^*L^*$ since $\epsilon \in L^*$

Conversely if $x \in L^*L^*$ then $x = x_1x_2$ with $x_1 \in L^*$ and $x_2 \in L^*$

$x \in L^*$ is clear if $x_1 = \epsilon$ or $x_2 = \epsilon$. Otherwise

So $x_1 = u_1 \dots u_n$ with $u_i \in L$

and $x_2 = v_1 \dots v_m$ with $v_j \in L$

Then $x = x_1x_2 = u_1 \dots u_nv_1 \dots v_m$ is in L^*

Algebraic Laws for Regular Expressions

Two laws that are useful to simplify regular expressions

Shifting rule

$$E(FE)^* = (EF)^*E$$

Denesting rule

$$(E^*F)^*E^* = (E + F)^*$$

Variation of the denesting rule

One has also

$$(E^*F)^* = \epsilon + (E + F)^*F$$

and this represents the words empty or finishing with F

Algebraic Laws for Regular Expressions

Example:

$a^*b(c + da^*b)^* = a^*b(c^*da^*b)^*c^*$

by denesting

$a^*b(c^*da^*b)^*c^* = (a^*bc^*d)^*a^*bc^*$

by shifting

$(a^*bc^*d)^*a^*bc^* = (a + bc^*d)^*bc^*$

by denesting. Hence

$a^*b(c + da^*b)^* = (a + bc^*d)^*bc^*$

Algebraic Laws for Regular Expressions

Examples: $10?0? = 1 + 10 + 100$

$$(1 + 01 + 001)^*(\epsilon + 0 + 00) = ((\epsilon + 0)(\epsilon + 0)1)^*(\epsilon + 0)(\epsilon + 0)$$

is the same as

$$(\epsilon + 0)(\epsilon + 0)(1(\epsilon + 0)(\epsilon + 0))^* = (\epsilon + 0 + 00)(1 + 10 + 100)^*$$

Set of all words with no substring of more than two adjacent 0's

Proving by induction

Let Σ be $\{a, b\}$

Lemma: *For all* n *we have* $a(ba)^n = (ab)^n a$

Proof: by induction on n

Theorem: $a(ba)^* = (ab)^* a$

Similarly we can prove $(a + b)^* = (a^* b)^* a^*$

Complement of a(n ordinary) regular expression

For building the "complement" of a regular expression, or the "intersection" of two regular expressions, we can use NFA/DFA

For instance to build E such that $L(E) = \{0,1\}^* - \{0\}$ we first build a DFA for the expression 0, then the complement DFA. We can compute E from this complement DFA. We get for instance

$$\epsilon + 1(0+1)^* + 0(0+1)^+$$

Abstract States

Two notations for the derivative L/a or $a \setminus L$

Last time I have used

$L/a = \{x \in \Sigma^* \mid ax \in L\}$

I shall use now the following notation (cf. exercice 4.2.3)

$a \setminus L = \{x \in \Sigma^* \mid ax \in L\}$

and more generally if z in Σ^*

$z \setminus L = \{x \in \Sigma^* \mid zx \in L\}$

Abstract States

Example: $L = \{a^n \mid 3\ divides\ n\}$ we have

$\epsilon \setminus L = L,\ a \setminus L = \{a^{3n+2} \mid n \geq 0\}$

$aa \setminus L = \{a^{3n+1} \mid n \geq 0\},\ aaa \setminus L = L$

Although Σ^* is infinite, the number of *distinct* sets of the form $u \setminus L$ is *finite*

Another example

$\Sigma = \{0, 1\}$

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

$\epsilon \setminus L = L,\ 0 \setminus L = \{0^n 1^{n+1} \mid n \geq 0\}$

$00 \setminus L = \{0^n 1^{n+2} \mid n \geq 0\},\quad 000 \setminus L = \{0^n 1^{n+3} \mid n \geq 0\}$

$1 \setminus L = \emptyset,\ 11 \setminus L = \emptyset$

In this case there are *infinitely* many *distinct* sets of the form $u \setminus L$

Another example

$L = L(E)$ where E is $(01 + 10)^*$

Then $0 \setminus E = 1E$ and $1 \setminus E = 0E$

$0 \setminus 0E = E$ and $1 \setminus 0E = \emptyset$

$0 \setminus 1E = \emptyset$ and $1 \setminus 1E = E$

So we have only finitely many sets of the form $u \setminus L$

Abstract States

The sets $u \setminus L$ are called the *abstract states* of the language L

Myhill-Nerode theorem: *A language is regular iff its set of abstract states is* finite

This is a *characterisation* of regular sets, and a powerful way to show that a language is *not* regular

Proof of the Myhill-Nerode theorem

Assume L is such that its set of abstract states $u \setminus L$ is finite.

We define Q to be the set of all $u \setminus L$. By hypothesis Q is a finite set

We define q_0 to be $L = \epsilon \setminus L$

We define $\delta(M, a) = a \setminus M$ for $a \in \Sigma$ and $M \subseteq \Sigma^*$ an arbitrary language

In particular $\delta(u \setminus L, a) = ua \setminus L$

Remark: We have $a \setminus (u \setminus L) = ua \setminus L$ and more generally $v \setminus (u \setminus L) = uv \setminus L$

Proof of the Myhill-Nerode theorem

Accepting states: we say that M is accepting iff ϵ is in M

We write $\psi(M)$ in this case

Thus x is in M iff $\psi(x \setminus M)$ iff $x \setminus M$ is accepting

Proof of the Myhill-Nerode theorem

Define $F \subseteq Q$ to be the set of abstract states $u \setminus L$ such that $\psi(u \setminus L)$. Thus $u \setminus L \in F$ iff $u \in L$

Lemma: *We have* $\hat{\delta}(L, x) = x \setminus L$

Proof: Write $x = a_1 \ldots a_n$. We have

$\hat{\delta}(L, a_1 \ldots a_n) = \delta(\ldots \delta(\delta(L, a_1), a_2) \ldots, a_n) = a_n \setminus (\ldots (a_2 \setminus (a_1 \setminus L)) \ldots) = a_1 \ldots a_n \setminus L$

Proof of the Myhill-Nerode theorem

If $A = (Q, \Sigma, \delta, q_0, F)$ we have $x \in L(A)$ iff $\psi(x \setminus L)$ iff $x \in L$. Thus $L = L(A)$ and L is regular

Notice that Q is now a finite set of sets.

Proof of the Myhill-Nerode theorem

This proves one direction: if the set of abstract sets is finite then L is regular

Conversely assume that L is regular then $L = L(A)$ for some DFA $A = (Q, \Sigma, \delta, q_0, F)$

We have

$$u \setminus L(A) = L(Q, \Sigma, \delta, q_0.u, F)$$

Indeed v is in $u \setminus L(A)$ iff uv is in $L(A)$ iff $q_0.(uv) = (q_0.u).v$ is in F

Since Q is *finite* since there are only finitely many possibilities for $u \setminus L$

Proof of the Myhill-Nerode theorem

Hence we have shown that L is *regular* iff there are only finitely many abstract states $u \setminus L$

This is a powerful way to prove that a language is *not* regular

For instance $L = \{0^n 1^n \mid n \geqslant 0\}$ is not regular since there are infinitely many abstract states $0^k \setminus L$

We still get an automaton but with infinitely many states

Proof of the Myhill-Nerode theorem

You should compare this with the use of the "pumping Lemma" (section 4.1) that I will present next time

Proof of the Myhill-Nerode theorem

This can be used also to show that a language is regular and indicate how to build a DFA for this language

$$L = \{a^n \mid 3 \; divides \; n\}$$

We have three abstract states $q_0 = L,\ q_1 = a \setminus L,\ q_2 = aa \setminus L$ hence a DFA with 3 states

A corollary of Myhill-Nerode's Theorem

Corollary: *If L is regular then each $u \setminus L$ is regular*

Proof: Since we have

$$v \setminus (u \setminus L) = uv \setminus L$$

each abstract state of $u \setminus L$ is an abstract state of L. If L is regular it has finitely many abstract states by Myhill-Nerode's Theorem. So $u \setminus L$ has finitely many abstract states and is regular by Myhill-Nerode's Theorem.

A corollary of Myhill-Nerode's Theorem

Another direct proof of

Corollary: *If* L *is regular then each* $u \setminus L$ *is regular*

Proof: L is regular so we have some DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $L = L(A)$. Define

$$u \setminus A = (Q, \Sigma, \delta, q_0.u, F)$$

We have seen that $L(u \setminus A) = u \setminus L(A)$.

Symbolic Computation of $u \setminus L$

$a \setminus \emptyset = \emptyset$

$a \setminus \epsilon = \emptyset$

$a \setminus a = \epsilon$

$a \setminus b = \emptyset$ if $b \neq a$

$a \setminus (E_1 + E_2) = a \setminus E_1 + a \setminus E_2$

$a \setminus (E_1 E_2) = (a \setminus E_1) E_2$ if $\epsilon \notin L(E_1)$

$a \setminus (E_1 E_2) = (a \setminus E_1) E_2 + a \setminus E_2$ if $\epsilon \in L(E_1)$

$a \setminus E^* = (a \setminus E) E^*$

Symbolic Computation of $u \setminus L$

If we introduce the notation $\psi(E) = \epsilon$ if ϵ in $L(E)$ and $\psi(E) = \emptyset$ if ϵ is not in $L(E)$

$a \setminus \emptyset = \emptyset \quad a \setminus \epsilon = \emptyset \quad a \setminus a = \epsilon$

$a \setminus b = \emptyset$ if $b \neq a$

$a \setminus (E_1 + E_2) = a \setminus E_1 + a \setminus E_2$

$a \setminus (E_1 E_2) = (a \setminus E_1)E_2 + \psi(E_1)(a \setminus E_2)$

$a \setminus E^* = (a \setminus E)E^*$

Symbolic Computation of $u \setminus L$

Computation of $\psi(E)$

$\psi(\emptyset) = \psi(a) = \emptyset$

$\psi(\epsilon) = \epsilon$

$\psi(E_1 + E_2) = \psi(E_1) + \psi(E_2)$

$\psi(E_1 E_2) = \psi(E_1)\psi(E_2)$

$\psi(E^*) = \epsilon$

Symbolic Computation of $u \setminus L$

We can similarly define $\alpha(E) = \epsilon$ if $L(E) \neq \emptyset$ and $\alpha(E) = \emptyset$ if $L(E) = \emptyset$

$\alpha(\emptyset) = \emptyset$

$\alpha(\epsilon) = \alpha(a) = \epsilon$

$\alpha(E_1 + E_2) = \alpha(E_1) + \alpha(E_2)$

$\alpha(E_1 E_2) = \alpha(E_1)\alpha(E_2)$

$\alpha(E^*) = \epsilon$

The Derivatives

Let E be $(0+1)^*01(0+1)^*$

$0 \setminus E = E + 1(0+1)^*$

$1 \setminus E = E$

$01 \setminus E = (0+1)^*$

$00 \setminus E = 0 \setminus E$

We have three languages $E, E + 1(0+1)^*, (0+1)^*$

We can build then a DFA for E

The Derivatives

Other example: let E be $(01)^*0$

$0 \setminus E = (0 \setminus (01)^*)0 + 0 \setminus 0 = 1(01)^*0 + \epsilon = (10)^*$

$1 \setminus E = (1 \setminus (01)^*)0 + 1 \setminus 0 = \emptyset$

$00 \setminus E = 0 \setminus 1(01)^*0 + 0 \setminus \epsilon = \emptyset$

$01 \setminus E = 1 \setminus 1(01)^*0 + 1 \setminus \epsilon = E$

We have three languages $E, (10)^*, \emptyset$

We can build then a DFA for E

The Derivatives

A more complex example $E = 1^*0 + 0^*1$ we get a DFA with 7 states

To get the minimal DFA for $E = 01^* + 1^*0$ we have to recognize that $1^* + \epsilon = 1^*$

Closure properties

Regular languages have the following *closure properties*

closure by union

closure by intersection

closure by complement

closure by difference

closure by reversal

closure by morphism and inverse morphism

Reversal

The *reversal* of a string $a_1 \dots a_n$ is the string $a_n \dots a_1$.

We write x^R the reversal of x

Thus $\epsilon^R = \epsilon$ and $0010^R = 0100$

Lemma: $(xy)^R = y^R x^R$

Reversal

If L is a language let L^R be the set of all x^R for $x \in L$

Theorem: *If L is regular then so if L^R*

Proof 1: We have $L = L(E)$ for a regular expression E. We define E^R by induction

$$(E_1 E_2)^R = E_2^R E_1^R \qquad (E_1 + E_2)^R = E_1^R + E_2^R \qquad (E^*)^R = (E^R)^*$$

$$a^R = a \qquad \emptyset^R = \emptyset \qquad \epsilon^R = \epsilon$$

We then prove $L(E^R) = L(E)^R$ by *structural induction* on E

Reversal

Proof 2: We have $L = L(A)$ for a NFA A, we define then a ϵ-NFA A' such that $L^R = L(A')$

We have $A = (Q, \Sigma, \delta, q_0, F)$

We take $q_1 \notin Q$ and define $A' = (Q \cup \{q_1\}, \Sigma, \delta', q_1, \{q_0\})$ which is an ϵ-NFA with

$r \in \delta'(s, a)$ iff $s \in \delta(r, a)$ for $r, s \in Q$

$r \in \delta'(q_1, \epsilon)$ iff $r \in F$

Example: The reverse of the language defined by $(0 + 1)^*0$ can be defined by $0(0 + 1)^*$

Monoid

Let Σ be an alphabet

Σ^* is a *monoid*

It has a binary operation $(x, y) \longmapsto xy$ which is associative $x(yz) = (xy)z$

It has a neutral element ϵ: we have $x\epsilon = \epsilon x = x$

It is not commutative in general $ab \neq ba$

Definition of Homomorphisms

Let Σ and Θ be two alphabets.

Definition: an *homomorphism* $h : \Sigma^* \to \Theta^*$

is an application such that, for all $x, y \in \Sigma^*$

$$h(xy) = h(x)h(y) \qquad h(\epsilon) = \epsilon$$

It follows that if $h(a_1 \dots a_n) = h(a_1) \dots h(a_n)$

Notice that $h(a) \in \Theta^*$ if $a \in \Sigma$

Closure under Homomorphisms

Let $h : \Sigma^* \rightarrow \Theta^*$ be an homomorphism

Theorem: *If $L \subseteq \Sigma^*$ is regular then $h(L)$ is regular*

We define $h(E)$ if E is a regular expression

$h(\epsilon) = \epsilon,\ h(\emptyset) = \emptyset,\ h(a) = h(a)$

$h(E_1 + E_2) = h(E_1) + h(E_2)$

$h(E_1 E_2) = h(E_1)h(E_2)$

$h(E^*) = h(E)^*$

Closure under Homomorphisms

Lemma: *If E is a regular expression then $L(h(E)) = h(L(E))$*

Proof: By structural induction on E. There are 6 cases.

This implies that given a DFA A such that $L(A) = L \subseteq \Sigma^*$ one can build a DFA A' such that $L(A') = h(L)$

This DFA exists because we have a regular expression (hence a ϵ-NFA hence a DFA by the subset construction)

Not obvious how to build directly this DFA

Closure under Homomorphisms

Theorem: *If* $L \subseteq \Theta^*$ *is regular then* $h^{-1}(L)$ *is regular*

Proof: Let $A = (Q, \Theta, \delta, q_0, F)$ DFA for L we define $A' = (Q, \Sigma, \delta', q_0, F)$ with

$$\delta'(q, a) = q.h(a)$$

A' is a DFA of alphabet Σ, we prove then that $L(A') = h^{-1}(L)$

Lemma: *We have for all* x $\hat{\delta}'(q, x) = q.h(x)$

The proof uses the fact that $q.(uv) = (q.u).v$

Closure under Homomorphisms

Notice that the proof would be difficult to do directly at the level of regular expressions. For instance if

If $h(a) = \epsilon,\ h(b) = b,\ h(c) = \epsilon$ what is $h^{-1}(\{\epsilon\})$?

If $h(a) = abb,\ h(b) = c,\ h(c) = c$ we have $h(ab) \in \{ab\}\{bc\}$ but we have $h^{-1}(\{ab\}) = h^{-1}(\{bc\}) = \emptyset$

Closure under Homomorphisms

Can we prove this using Myhill-Nerode's Theorem?

We have to compute $u \setminus h^{-1}(L)$

v is in this set iff $h(uv) = h(u)h(v)$ is in L

Hence $u \setminus h^{-1}(L)$ is the same as $h^{-1}(h(u) \setminus L)$

Hence if L is regular there are only a finite number of possible values for $u \setminus h^{-1}(L)$ and hence $h^{-1}(L)$ is *regular*

Closure under Union

We have a direct construction via ϵ-NFA or variation on the product of DFA

It is interesting to notice that we have also a proof via Myhill-Nerode's Theorem

$$u \setminus (L_1 \cup L_2) = (u \setminus L_1) \cup (u \setminus L_2)$$

If L_1, L_2 are regular, we have only a finite number of possible values for $u \setminus (L_1 \cup L_2)$, hence $L_1 \cup L_2$ is regular

Closure under Intersection, Difference, Complement

The same argument works for showing that regular languages are closed under intersection, complement and differences

$$u \setminus (L_1 \cap L_2) = (u \setminus L_1) \cap (u \setminus L_2)$$

$$u \setminus L' = (u \setminus L)'$$

Application: we have another way to compute $0'$ We have also direct constructions on DFAs

Closure under Prefix

If $L \subseteq \Sigma^*$ is a language we write $Pre(L)$ the set

$\{u \in \Sigma^* \mid \exists v.\ uv \in L\}$

This is the set of *prefixes* of words that are in L

We present two proofs that $Pre(L)$ is regular if L is regular

One proof using Myhill-Nerode's Theorem, and one proof using a DFA for L

Closure under Prefix

If $(Q, \Sigma, \delta, q_0, F)$ is a DFA for L we define a DFA for $Pre(L)$ by taking

$A' = (Q, \Sigma, \delta, q_0, F')$

where $F' = \{q \in Q \mid \exists z.\ \hat{\delta}(q, z) \in F\}$

We then show that x in $L(A')$ iff $\hat{\delta}(q_0, x) \in F'$ iff there exists z such that $(q_0.x).z = q_0.(xz)$ in F iff xz in $Pre(L(A)) = Pre(L)$

Closure under Prefix

We have also a proof by using regular expression: given a regular expression E we define $p(E)$ such that $L(p(E)) = Pre(L(E))$

$$p(a) = \epsilon + a \qquad p(\epsilon) = \epsilon \qquad p(\emptyset) = \emptyset$$

$$p(E_1E_2) = p(E_1) + E_1p(E_2)$$

$$p(E_1 + E_2) = p(E_1) + p(E_2)$$

$$p(E^*) = E^*p(E)$$ ‘

Minimal automaton

If L is regular, we have seen that there is a DFA which recognizes L which has for set of states the set S of *abstract states* of L

S is the set of all $u \setminus L$

$u \setminus L$ goes to $(ua) \setminus L$

This is *the minimal* automaton which recognizes L

Minimal automaton

Let $A = (Q, \Sigma, \delta, q_0, F)$ be another DFA which recognizes L

We show that Q has more elements than S

Indeed we know that $u \setminus L$ is $(Q, \Sigma, \delta, q_0.u, F)$

Thus S has less elements than there are accessible states in Q

Minimal automaton

For example, for $L = L((0+1)^*01(0+1)^*)$ we have computed three abstract states

$$L,\ 0 \setminus L,\ 01 \setminus L = \Sigma^*$$

Hence *any* automaton which recognizes L has *at least* three states

Minimal automaton

Let Q' be the set of states accessible from q_0

If $q_0.u = q_0.v$ I claim that we have $u \setminus L = v \setminus L$

Indeed this is the set recognized by $(Q, \Sigma, \delta, q_0.u, F) = (Q, \Sigma, \delta, q_0.v, F)$

This means that we have a surjective map $\psi : Q' \to S,\ q_0.u \longmapsto u \setminus L$

Furthermore $\psi(q.a) = a \setminus \psi(q)$

This shows that connection between *any* automaton recognizing L and the minimal automaton of abstract states

Minimal automaton

Next time, I will present an algorithm for computing the minimal automaton for L *given* a DFA for L

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

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

and hence $E = F$ iff

$$\delta(E) = \delta(F)$$

$$a \setminus E = a \setminus F$$

$$b \setminus E = b \setminus F$$

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 $f : X \to Y$ 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

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

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 $w = a_1 \ldots a_l$ 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 $0^n 1^n \in L_{eq}$ and hence

$0^n 1^n = xyz$

with $|xy| \leqslant n,\ y \neq \epsilon$ and $xy^k z \in L_{eq}$ 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

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

$L = \{0^n 1^m \mid n \geqslant m\}$

$L' = \{0^n 1^m \mid n \neq m\}$

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: $L/0^k 1 = \{1^{2k-1}\}$, infinitely many abstract states.