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 $x = x_1x_2 = u_1 \dots u_n v_1 \dots v_m$ 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

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

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

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

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

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

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

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

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

Lemma: *We have* $L.u = u \setminus L$

Proof: By induction on u. This holds for $u = \epsilon$ and if it holds for v and $u = av$ then

$$L.(av) = (a \setminus L).v = v \setminus (a \setminus L) = av \setminus L$$

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

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 $0^k \setminus L$

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 $\delta(E) = \epsilon$ if ϵ in $L(E)$ and $\delta(E) = \emptyset$ 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 $b \neq a$

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

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

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

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

Closure properties

Regular languages have remarkable *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 $a_1 \ldots a_n$ is the string $a_n \ldots a_1$.

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

$(E_1E_2)^R = E_2^R E_1^R$ $\quad$ $(E_1 + E_2)^R = E_1^R + E_2^R$ $\quad$ $(E^*)^R = (E^R)^*$

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

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* $h : \Sigma^* \rightarrow \Theta^*$

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 $h : \Sigma^* \to \Theta^*$ 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'')
```