Search algorithm

Clever algorithm even for a single word

Example: find "abac" in "abaababac"

See Knuth-Morris-Pratt and String searching algorithm on wikipedia

Subset construction

We have defined for a DFA

$L(A) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}$

and for A NFA

$L(A) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$

For any NFA A we can build a DFA A_D such that $L(A) = L(A_D)$

Regular languages

Given an alphabet Σ, a language $L \subseteq \Sigma^*$ is *regular* iff there exists a DFA A such that $L = L(A)$

Theorem: *A language L is regular iff there exists a NFA N such that $L = L(N)$*

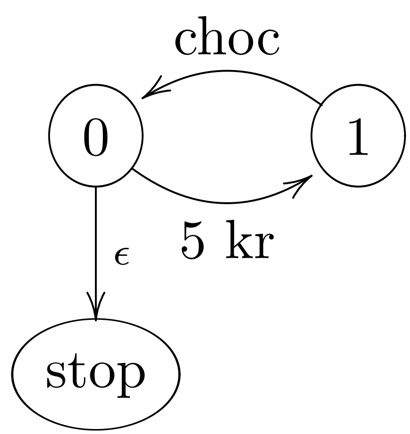
Proof: If L is regular then $L = L(A)$ for some DFA A. To any DFA A we can associate a NFA N_A such that $L(A) = L(N_A)$. If $A = (Q, \Sigma, \delta, q_0, F)$ we simply take $N_A = (Q, \Sigma, \delta', q_0, F)$ with $\delta'(q, a) = \{\delta(q, a)\}$. Notice that $\delta' \in Q \times \Sigma \to Pow(Q)$.

In the other direction, if $L = L(N)$ for some NFA N then, the power set construction gives a DFA A such that $L(N) = L(A)$. We have then $L = L(A)$ and so L is regular. Q.E.D.

Automata with ϵ-Transitions

Another extension of the notion of automata that is useful but adds no more power

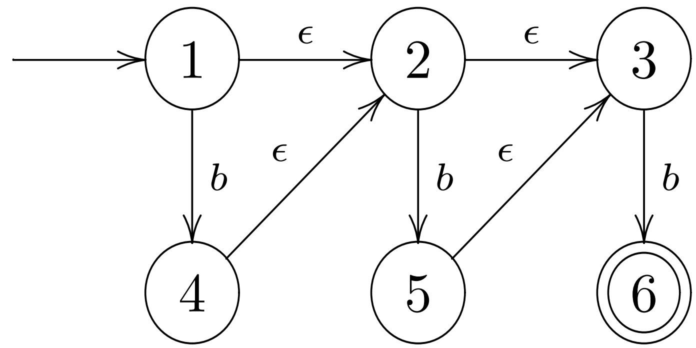
Intuitively an ϵ-transition occurs when one can go from one state to another without reading any input symbol



A vending machine that may decide to stop

Automata with ϵ-Transitions

$\Sigma = \{b\}$



ϵ-NFA accepting {b,bb,bbb}

The machine can jump by itself from the state 1 to the state 2

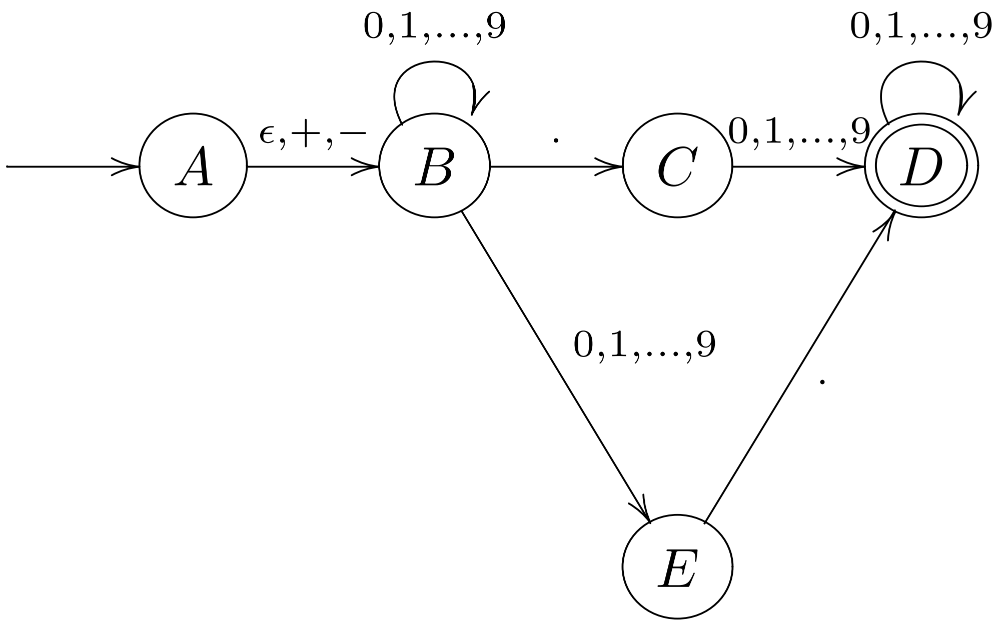
Automata with ϵ-Transitions

Example: decimal numbers consisting of

1. An optional + or - sign
2. A string of digits
3. A decimal point, and
4. Another string of digits. Either this string, or the string (2) can be empty, but at least one of them is nonempty.

Automata with ϵ-Transitions

A possible ϵ-NFA for this language is



Notice the crucial use of ϵ transition to represent the "optional" choice of the sign + or -

Automata with ϵ-Transitions

Definition A ϵ-NFA consists of

1. a finite set of *states* (often denoted Q)
2. a finite set Σ of *symbols* (alphabet)
3. a *transition function* that takes as argument a state and an element of $\Sigma \cup \{\epsilon\}$ and returns a set of states (often denoted δ); this set can be empty
4. a *start state*
5. a set of *final* or *accepting* states (often denoted F)

We have $F \subseteq Q$ and $\delta \in Q \times (\Sigma \cup \{\epsilon\}) \to Pow(Q)$

Automata with ϵ-Transitions

For the example of decimal numbers the transition table is

	+,-	.	0,1,. . .,9	ϵ
A	$\{B\}$	$\emptyset$	$\emptyset$	$\{B\}$
B	$\emptyset$	$\{C\}$	$\{B, E\}$	$\emptyset$
C	$\emptyset$	$\emptyset$	$\{D\}$	$\emptyset$
D	$\emptyset$	$\emptyset$	$\{D\}$	$\emptyset$
E	$\emptyset$	$\{D\}$	$\emptyset$	$\emptyset$

ϵ-Closures

If $X \subseteq Q$ we define the ϵ-closure ECLOSE(X) inductively

BASIS: If $q \in X$ then q is in ECLOSE(X)

INDUCTION: If p is in ECLOSE(X) and $r \in \delta(p, \epsilon)$ then r is in ECLOSE(X)

Note that ECLOSE($\emptyset$) $= \emptyset$

Informally, we follow all transitions out of X that are labeled ϵ. We say that X is ϵ-*closed* iff $X =$ ECLOSE(X).

Remark: X is ϵ-closed iff $q \in X$ and $q \xrightarrow{\epsilon} q'$ implies $q' \in X$

ϵ-Closures

Yet another way to present ECLOSE(X) is with the two rules

$$\frac{q \in X}{q \in \mathsf{ECLOSE}(X)}$$

$$\frac{q \in \mathsf{ECLOSE}(X) \qquad q' \in \delta(q, \epsilon)}{q' \in \mathsf{ECLOSE}(X)}$$

Intuitively $q' \in \mathsf{ECLOSE}(X)$ iff there exists $q_0 \in X$ and a sequence of ϵ-transitions

$q_0 \stackrel{\epsilon}{\to} q_1 \stackrel{\epsilon}{\to} \ldots \stackrel{\epsilon}{\to} q_n = q'$

ϵ-Closures

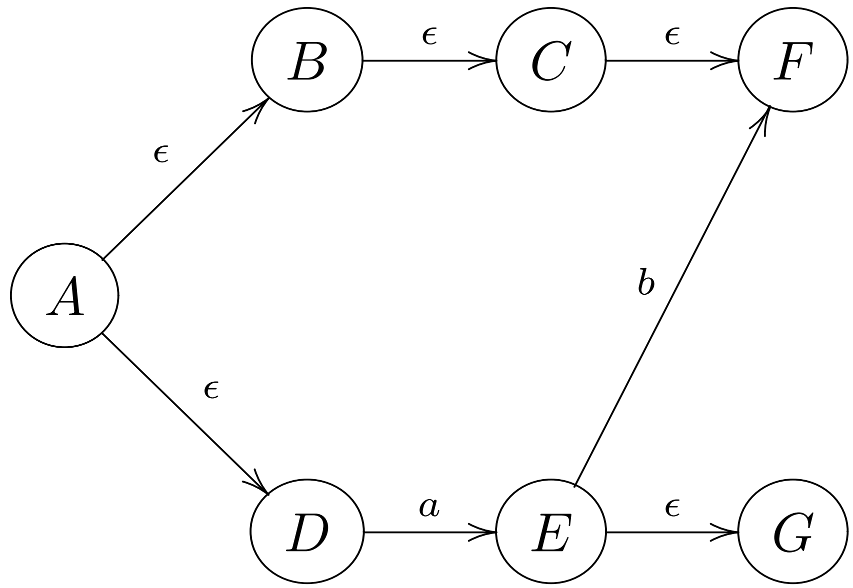
We say that $Y \subseteq Q$ is ϵ-closed iff

If q in Y and q' in $\delta(q, \epsilon)$ then q' in Y

We have that ECLOSE(X) is the *smallest* subset of Q containing X which is ϵ-closed

ϵ-Closures

For the automaton



we have

ECLOSE($\{A\}$) = {A,B,C,D,F}

Functional representation

```
import List(union)

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

jump :: Q -> [Q]

jump A = [B,D]
jump B = [C]
jump C = [F]
jump F = []
jump D = []
jump E = [G]
```

Functional representation

```
isSub as bs = and (map (\x -> elem x bs) as)

isClos as = isSub (as >>= jump) as

closure qs =
 let qs’ = qs >>= jump
 in if isSub qs’ qs then qs
      else closure (union qs qs’)
```

How to run an ϵ-NFA

Given any ϵ-NFA $E = (Q, \Sigma, \delta, q_0, F)$ we define

$\hat{\delta}(q, \epsilon) = \text{ECLOSE}(\{q\})$

$\hat{\delta}(q, ay) = \bigcup_{p \in \Delta(\text{ECLOSE}(q), a)} \hat{\delta}(p, y)$

where $\Delta(X, a) = \cup_{q \in X} \delta(q, a)$

Definition: $L(E) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$

Remark: All sets $q.x = \hat{\delta}(q, x)$ are ϵ-closed

Remark: $q.a$ is ECLOSE(Δ(ECLOSE(q),a))

Representation in functional programming

```
import List(union)

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

jump :: Q -> [Q]
jump A = [B]
jump B = []
jump C = []
jump D = []
jump E = []
```

Representation in functional programming

```
isSub as bs = and (map (\ x -> elem x bs) as)

isClos as = isSub (as >>= jump) as

closure qs =
 let qs' = qs >>= jump
 in if isSub qs' qs then qs
      else closure (union qs qs')
```

Representation in functional programming

```
next a A | elem a "+-" = [B]
next a B | elem a "0123456789" = [B,E]
next a C | elem a "0123456789" = [D]
next a D | elem a "0123456789" = [D]
next '.' B = [C]
next '.' E = [D]
next _ _ = []

run [] q = closure [q]
run (a:x) q = closure [q] >>= next a >>= run x
```

Representation in functional programming

We can prove by induction on x that run x q is always ϵ-closed

The main Lemma is that any union of ϵ-closed sets is a set which is ϵ-closed

Eliminating ϵ-Transitions

We define then the DFA $D = (Q_D, \Sigma_D, \delta_D, q_D, F_D)$ where

Q_D is the set of ϵ-closed subsets of Q

$\Sigma_D = \Sigma$

$\delta_D(X, a) = \text{ECLOSE}(\Delta(X, a))$

$q_D = \text{ECLOSE}(\{q_0\})$

$F_D = \{X \in Q_D \mid X \cap F \neq \emptyset\}$

Lemma: For any $x \in \Sigma^*$ we have $\hat{\delta}(q_0, x) = \hat{\delta_D}(q_D, x)$

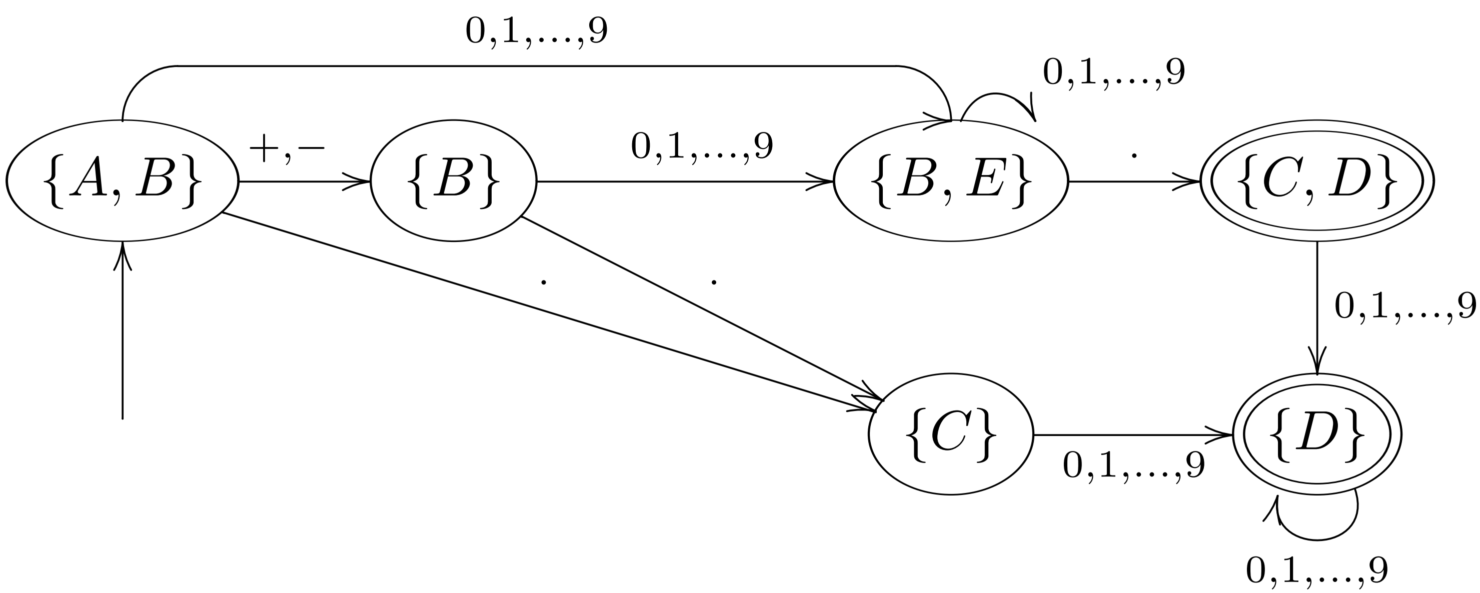
Theorem: $L(E) = L(D)$

Proof: We have $x \in L(E)$ iff $\hat{\delta}(q_0, x) \cap F \neq \emptyset$ iff $\hat{\delta}(q_0, x) \in F_D$ iff $\hat{\delta_D}(q_D, x) \in F_D$ iff $x \in L(D)$. We use the Lemma to replace $\hat{\delta}(q_0, x)$ by $\hat{\delta_D}(q_D, x)$

Eliminating ϵ-Transitions

Similar construction as for building a DFA from a NFA but now we close at each steps

For the example of decimal numbers we get the following automaton



where the state $\emptyset$ is not represented

Once again, we get this program mechanically!

Representation in functional programming

```
pNext a qs = closure (qs >>= next a)

pRun [] qs = qs
pRun (a:x) qs = pRun x (pNext a qs)

run x q = pRun x (closure [q])
```

NFA as labelled graphs

A NFA $A = (Q, \Sigma, \delta, q_0, F)$ can be seen as a labelled graph

$q_1 \xrightarrow{a} q_2$ iff $q_2 \in \delta(q_1)$

We define also, for $x \in \Sigma^*$

$q_1 \xrightarrow{x} q_2$

by induction on x

If $x = \epsilon$ this means $q_1 = q_2$

If $x = ay$ this means that there exists $q \in Q$ such that $q_1 \xrightarrow{a} q$ and $q \xrightarrow{y} q_2$

We have $q_1 \xrightarrow{x} q_2$ iff $q_2 \in \hat{\delta}(q_1, x)$

$L(A) = \{x \in \Sigma^* \mid (\exists q \in F)\ q_0 \xrightarrow{x} q\}$

The Product Construction on NFA

Given $A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ two NFAs with the same alphabet Σ we define the product $A = A_1 \times A_2$ as

- the set of state is $Q_1 \times Q_2$
- $\delta((r_1, r_2), a) = \delta_1(r_1, a) \times \delta_2(r_2, a)$. In this way $(r_1, r_2) \xrightarrow{a} (s_1, s_2)$ iff *both* $r_1 \xrightarrow{a} s_1$ and $r_2 \xrightarrow{a} s_2$.
- (r_1, r_2) is accepting iff $r_1 \in F_1$ and $r_2 \in F_2$
- the initial state is (q_1, q_2)

Lemma: $(r_1, r_2) \xrightarrow{x} (s_1, s_2)$ *iff* $r_1 \xrightarrow{x} s_1$ *and* $r_2 \xrightarrow{x} s_2$

Proposition: $L(A_1 \times A_2) = L(A_1) \cap L(A_2)$

Complement of a NFA

Be careful!

In general we don't have $L(A') = \Sigma^* - L(A)$ if

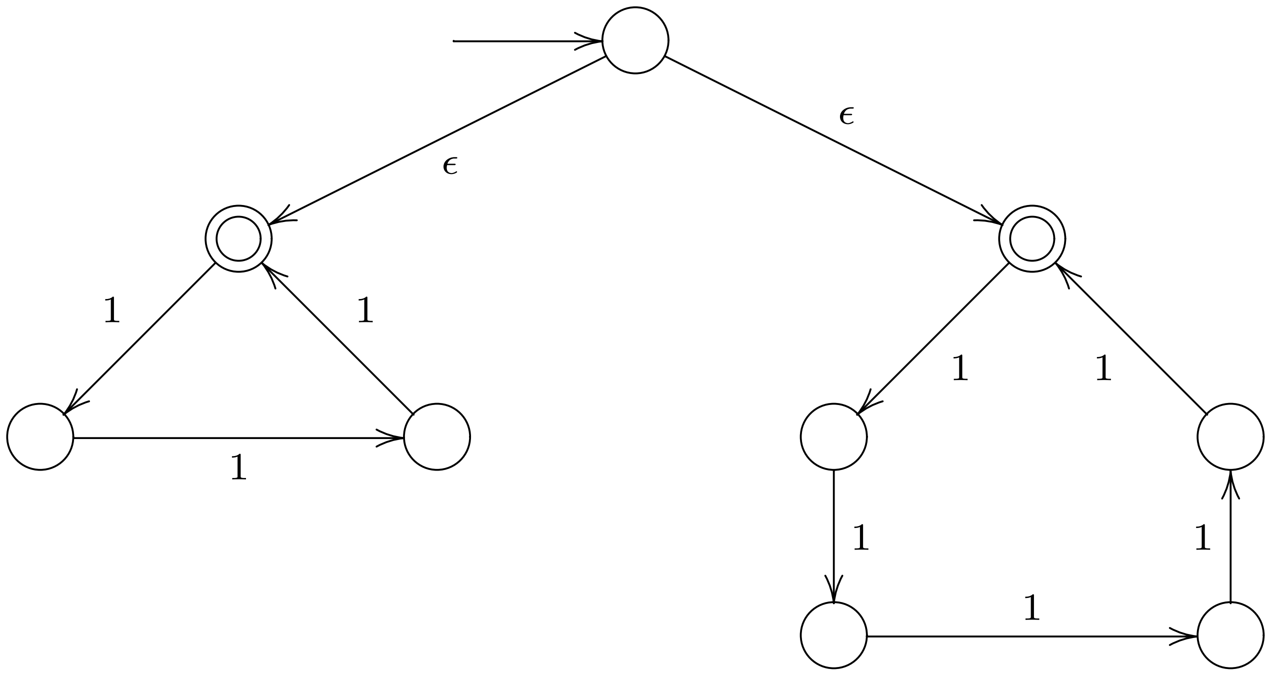
$A' = (Q, \Sigma, \delta, q_0, Q - F)$

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

and A is a NFA

Automata with ϵ-Transitions

$\Sigma = \{1\}$



ϵ-NFA accepting all words of length multiple of 3 *or* 5

The automaton *guesses* the right direction, and then *verifies* that $|w|$ is correct!

Eliminating ϵ-Transitions

This corresponds to the NFA