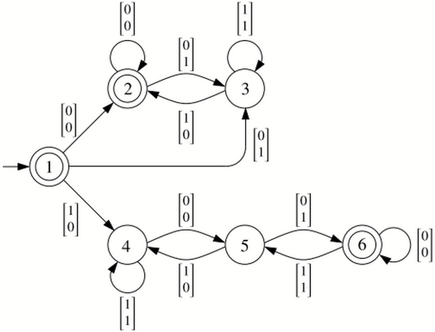
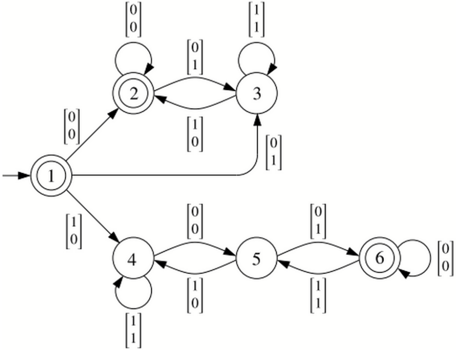
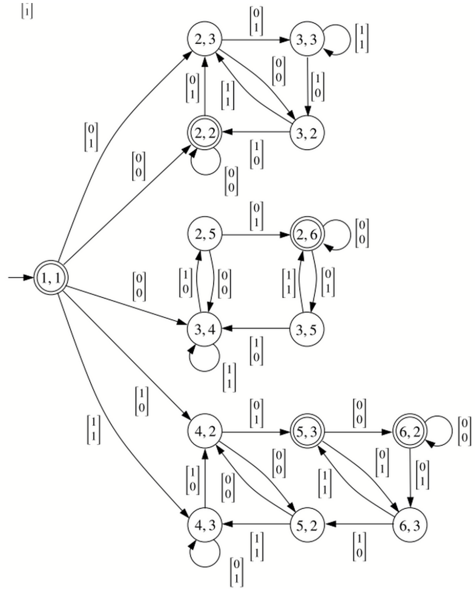

1
2
3
4
5
6


1
2
3
4
5
6


1, 1
2, 3
3, 3
2, 2
3, 2
2, 5
2, 6
3, 4
3, 5
4, 2
5, 3
6, 2
4, 3
5, 2
6, 3

Pre and Post

- Goal (for post):

  given

  – an automaton A recognizing a set X, and

  – a transducer T recognizing a relation R

  construct an automaton B recognizing the set

$$\{ y \mid \exists x \in X : (x, y) \in R \}$$

We slightly modify the construction for join.

Instead of:

$$\begin{bmatrix} q_{01} \\ q_{02} \end{bmatrix} \xrightarrow{\begin{bmatrix} a_1 \\ b_1 \end{bmatrix}} \begin{bmatrix} q_{11} \\ q_{12} \end{bmatrix} \quad \text{iff} \quad \begin{array}{l} q_{01} \xrightarrow{\begin{bmatrix} a_1 \\ c_1 \end{bmatrix}} q_{11} \\ q_{02} \xrightarrow{\begin{bmatrix} c_1 \\ b_1 \end{bmatrix}} q_{12} \end{array}$$

for some letter c1

we now use

$$\begin{bmatrix} q_{01} \\ q_{02} \end{bmatrix} \xrightarrow{b_1} \begin{bmatrix} q_{11} \\ q_{12} \end{bmatrix} \quad \text{iff} \quad \begin{array}{l} q_{01} \xrightarrow{a_1} q_{11} \\ q_{02} \xrightarrow{\begin{bmatrix} a_1 \\ b_1 \end{bmatrix}} q_{12} \end{array}$$

for some letter a_1

From Join to Post

$Join(T_1, T_2)$

Input: transducers $T_1 = (Q_1, \Sigma \times \Sigma, \delta_1, q_{01}, F_1)$, $T_2 = (Q_2, \Sigma \times \Sigma, \delta_2, q_{02}, F_2)$

Output: transducer $T_1 \circ T_2 = (Q, \Sigma \times \Sigma, \delta, q_0, F)$

$Q, \delta, F' \leftarrow \emptyset;\ q_0 \leftarrow [q_{01}, q_{02}]$

$W \leftarrow \{[q_{01}, q_{02}]\}$

while $W \neq \emptyset$ **do**

  pick $[q_1, q_2]$ **from** W

  add $[q_1, q_2]$ **to** Q

  if $q_1 \in F_1$ and $q_2 \in F_2$ **then add** $[q_1, q_2]$ **to** F'

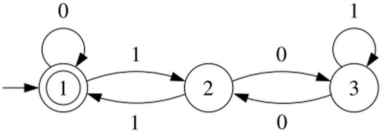
  for all $(q_1, (a, c), q_1') \in \delta_1, (q_2, (c, b), q_2') \in \delta_2$ **do**

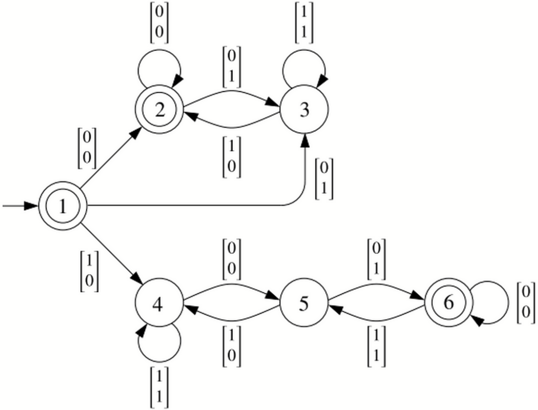
    add $([q_1, q_2], (a, b), [q_1', q_2'])$ **to** δ

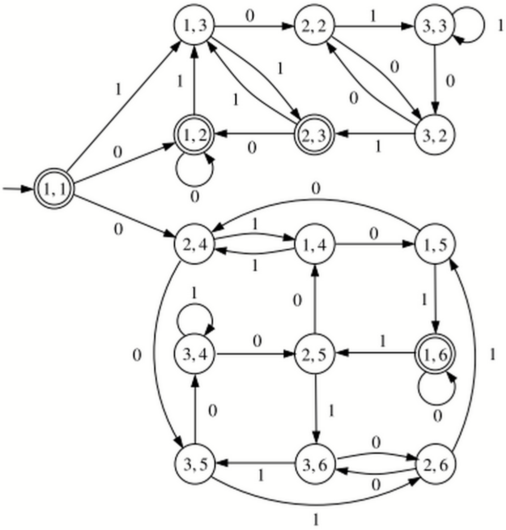
    if $[q_1', q_2'] \notin Q$ **then add** $[q_1', q_2']$ **to** W

$F \leftarrow$ **PadClosure**$((Q, \Sigma \times \Sigma\delta, q_0, F'), (\#, \#))$

Example: compute the set { f(n) | n multiple of 3 }






1, 1
1, 2
1, 3
2, 2
3, 3
2, 3
3, 2
2, 4
1, 4
1, 5
3, 4
2, 5
1, 6
3, 5
3, 6
2, 6
0
1

6. Some pattern matching

Given

- a word w (the text) of length n, and
- a regular expression p (the pattern) of length m,

determine the smallest number k' such that there is a subword $w_{k,k'}$ of w with

$$w_{k,k'} \in L(p) \ .$$

Remark: We here minimize the right end of the matching subword. To make a match unique, one could require *e.g.*, that its length is minimal (or maximal).

NFA-based solution

PatternMatchingNFA(t, p)
Input: text $t = a_1 \dots a_n \in \Sigma^+$, pattern $p \in \Sigma^*$
Output: the first occurrence of p in t, or $\perp$ if no such occurrence exists.

```
1  A ← RegtoNFA(Σ* p)
2  S ← {q_0}
3  for all k = 0 to n − 1 do
4      if S ∩ F ≠ ∅ then return k
5      S ← δ(S, a_{k+1})
6  return ⊥
```

- Line 1 takes $O(m^3)$ time, output has $O(m)$ states
- Loop is executed at most n times
- One iteration takes $O(s^2)$ time , where s is the number of states of A
- Since $s = O(m)$, the total runtime is $O(m^3 + nm^2)$, and $O(nm^2)$ for $m \leq n$.

DFA-based solution

PatternMatchingDFA(t, p)
Input: text $t = a_1 \dots a_n \in \Sigma^+$, pattern p
Output: the first occurrence of p in t, or $\perp$ if no such occurrence exists.

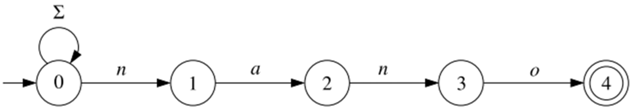
```
1  A ← NFAtoDFA(RegtoNFA(Σ* p))
2  q ← q_0
3  for all k = 0 to n − 1 do
4      if q ∈ F then return k
5      q ← δ(q, a_{k+1})
6  return ⊥
```

- Line 1 takes $2^{O(m)}$ time
- Loop is executed at most n times
- One iteration takes constant time
- Total runtime is $O(n) + 2^{O(m)}$

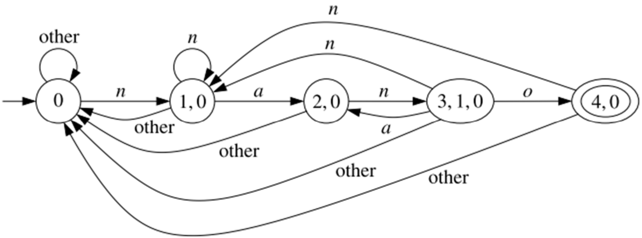
The word case

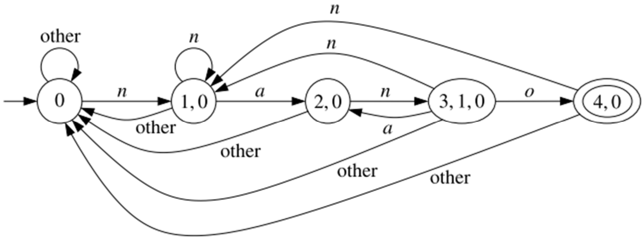
- The pattern p is a word of length m
- Naive algorithm: move a window of size m along the word one letter at a time, and compare with p after each step. Runtime: $O(nm)$
- We give an algorithm with $O(n+m)$ runtime for any alphabet of size $0 \leq |\Sigma| \leq n$.
- First we explore in detail the shape of the DFA for $\Sigma^* p$.

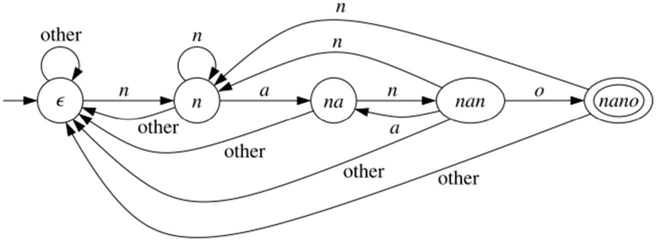
Obvious NFA for $\Sigma^* p$ and $p = nano$



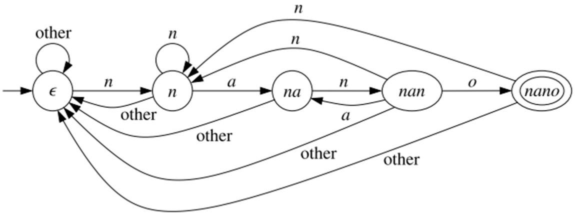
Result of applying NFAtoDFA




other
n
n
n
0
n
1, 0
a
2, 0
n
3, 1, 0
o
4, 0
other
a
other
other
other

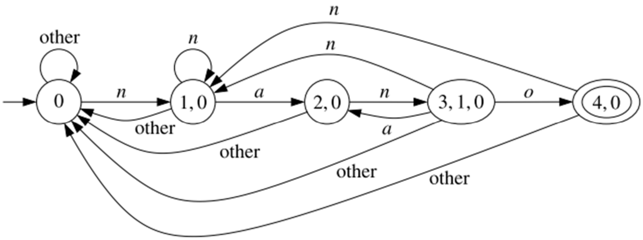

other
n
n
n
ϵ
n
n
a
na
n
nan
o
nano
other
a
other
other
other

Intuition



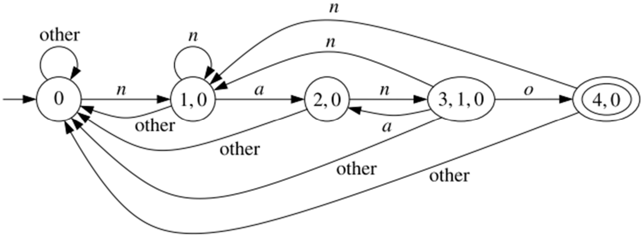
- Transitions of the „spine“ correspond to hits: the next letter is the one that „makes progress“ towards nano
- Other transitions correspond to misses, i.e., „wrong letters“ and „throw the automaton back“

Observations



- For every state $i = 0, 1, \ldots, 4$ of the NFA there is exactly one state S of the DFA such that i is the largest state of S.
- For every state S of the DFA, with the exception of $S = \{0\}$, the result of removing the largest state is again a state of the DFA.

Observations



- For every state $i = 0, 1, \ldots, 4$ of the NFA there is exactly one state S of the DFA such that i is the largest state of S.
- For every state S of the DFA, with the exception of $S = \{0\}$, the result of removing the largest state is again a state of the DFA.
- Do these properties hold for every pattern p?

Heads and tails, hits and misses

- Head of S, denoted $h(S)$: largest state of S
- Tail of S, denoted $t(S)$: rest of the state
- Example: $h(\{3,1,0\}) = 3$, $t(\{3,1,0\}) = \{1,0\}$

- Given a state S, the letter leading to the next state in the „spine“ is the (unique) hit letter for S
- All other letters are miss letters for S
- Example: hit for {3,1,0} is *o*, whereas *n* or *a* are misses

- Fund. Prop: Let S_k be the k-th state picked from the worklist during the execution of *NFAtoDFA*(A_p).

  (1) $h(S_k) = k$,

  (2) If $k > 0$, then $t(S_k) = S_l$ for some $l < k$

Proof Idea:

- (1) and (2) hold for $S_0 = \{0\}$.
- For S_k we look at $\delta(S_k, a)$ for each a, where δ transition relation of A_p .
- By i.h. we have $S_k = \{k\} \cup S_l$ for some $l < k$
- We distinguish two cases: *a* is a hit for S_k, and *a* is a miss for S_k .

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

Hit:

$$\begin{array}{ccc} \{k\} & \cup & S_l \\ \downarrow a & & \downarrow a \\ \{k+1\} & \cup & \delta(S_l, a) \end{array}$$

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

$$\{k\} \quad \cup \quad S_l$$

Hit: $\quad a \downarrow \qquad\qquad a \downarrow$

$$\{k+1\} \cup \delta(S_l, a)$$

Added to the worklist earlier, and so some $S_{l'}$

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

Hit:

$$\begin{array}{ccc} \{k\} & \cup & S_l \\ \downarrow a & & \downarrow a \\ \{k+1\} & \cup & \delta(S_l, a) \\ = & & = \\ \{k+1\} & \cup & S_{l'} \end{array}$$

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

$$\{k\} \quad \cup \quad S_l$$

Miss: $\quad a \downarrow \qquad\qquad a \downarrow$

$$\emptyset \quad \cup \quad \delta(S_l, a)$$

- $S_k = \{k\} \cup S_l$ for some $l < k$
- $\delta(S_k, a) = \delta(k, a) \cup \delta(S_l, a)$

Miss:

$$\begin{array}{ccc} \{k\} & \cup & S_l \\ \text{a}\downarrow & & \text{a}\downarrow \\ \emptyset & \cup & \delta(S_l, a) \\ & & = \\ & & S_{l'} \end{array}$$

Consequences

Prop: The result of applying *NFAtoDFA*(A_p), where A_p is the obvious NFA for $\Sigma^* p$, yields a minimal DFA with m states and $|\Sigma|m$ transitions.

Proof: All states of the DFA accept different languages.

So: concatenating *NFAtoDFA and PatternMatchingDFA* yields a $O(n + |\Sigma|m)$ algorithm.

- Good enough for constant alphabet
- Not good enough for $|\Sigma| = O(n)$