- Vertailuissa $P = S^0$ ja $P = S^j$ riveillä 8 ja 10 vertaillaan m-merkkisiä jonoja. Ne toteutetaan esim. triviaalialgoritmilla.

Sanomme

- Väärä täsmäys (false match): $H(P) = H(S^j)$, mutta $P \neq S^j$.
- Oikea täsmäys (proper match): $H(P) = H(S^j)$ ja $P = S^j$.

Algoritmin 1.3.1 aikavaatimus on selvästi muotoa $O(m + n + fm + hm)$, missä f on väärien ja h oikeiden täsmäysten lukumäärä. Pahimmassa tapauksessa tämä on $O(mn)$. Keskimääräisessä tapauksessa (tietyin oletuksin) f ja h ovat pieniä, joten keskimääräisessä tapauksessa aikavaatimus on $O(m+n)$.

Karp ja Rabin tarkastelevat myös universaaliin hajautukseen perustuvia versioita algoritmista:

- Valitaan q *satunnaisesti* lukujoukosta Q.
- Q:n luvut voivat olla pieniäkin, eikä niiden tarvitse olla kokonaislukuja.
- Väärän täsmäyksen todennäköisyys on erittäin pieni syötteestä riippumatta (universaalius).
- q voidaan myös vaihtaa aina väärän täsmäyksen sattuessa.

1.4 Eliminointimenetelmä

U. Vishkin: Deterministic sampling — a new technique for fast pattern matching. *SIAM Journal of Computing* 20(1):22–40, 1991.

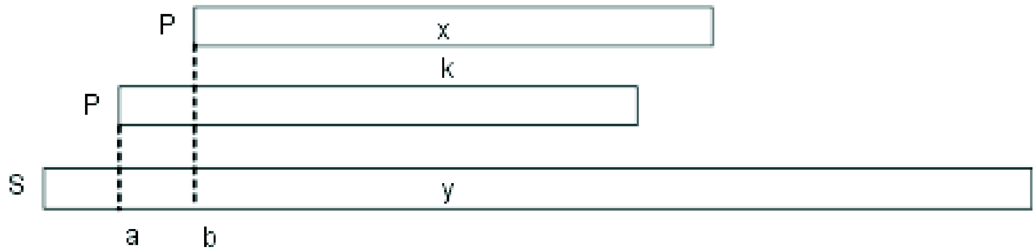
- *Kandidaatti*: tekstin positio, josta hahmon esiintymä voi alkaa.
- Idea:
  - Asetetaan kaikki tekstin positiot kandidaateiksi.
  - Karsitaan kandidaatteja tutkimalla tekstiä.
  - Operaatio: *parivertailu* (duel, kaksintaistelu).
  - Jäljelle jääneitä kandidaatteja vastaavat esiintymät tutkitaan tarkasti.
- Käyttöalueet: rinnakkaisalgoritmit ja kaksiulotteinen merkkijononsovitus.

Konsistenssi

- P:stä muodostetaan taulukko W (witness).
- Asetetaan $W[d] = i$, jolle pätee $p[i] \neq p[i+d]$, $0 \leq i < m - d$.
  - Mikäli tällaista i ei ole, asetetaan $W[d] = -1$.
  - Mikäli ehdon täyttäviä i:n arvoja on useita, valitaan niistä esim. pienin.
- Esim. tekstille `xyzxyzwxyz` saadaan taulukko

```
      x y z x y z w x y z
  i     1 2 3 4 5 6 7 8 9
W[i]    0 0 3 0 0 0-1 0 0
```

- Kandidaatit a ja b, $a \leq b$, ovat *konsistentteja*, jos $|b - a| \geq m$ tai $W[b - a] = -1$.
  - Jos kandidaatit eivät ole konsistentteja, toinen niistä voidaan eliminoida.
- Konsistenssi on *transitiivinen*: jos a ja b ovat konsistentteja, b ja c ovat konsistentteja sekä $a < b < c$, niin a ja c ovat konsistentteja.
- Merkitään
  - $incons(Z)$: pinon Z kaksi päällimmäistä alkiota eivät ole konsistentteja.
  - $duel(a, b) = (\textbf{if } p[W[b-a]] = s[b + W[b-a]] \textbf{ then } b \textbf{ else } a)$



Kuva 1.1: Merkkejä x ja y vertaillaan, $k = W[b - a]$. Jos $x = y$, valitaan b, muuten a.

Algoritmi 1.4.1 *Eliminointimenetelmä*

Syöte: teksti S ja hahmo P
Tuloste: P:n ensimmäinen esiintymä S:ssä
(1) muodosta taulukko W;
(2) $Z := \emptyset$;
(3) **for** $j := 0$ **to** $n-1$ **do**
(4) $\quad push(Z, j)$;
(5) $\quad$ **while** $|Z| \geq 2$ **and** $incons(Z)$ **do**
(6) $\qquad b := Pop(Z)$;
(7) $\qquad a := Pop(Z)$;
(8) $\qquad Push(Z, duel(a, b))$;
(9) **for** $j := 0$ **to** $n-1$ **do** $cand[i] := 0$;
(10) **while not** $empty(Z)$ **do** $cand[Pop(Z)] := 1$;
(11) $i := -1$;
(12) **for** $j := 0$ **to** $n-1$ **do**
(13) $\quad$ **if** $i = -1$ **and** $cand[j] = 1$ **then** $i := 0$;
(14) $\quad$ **if** $i \geq 0$ **and** $P[i] = S[j]$ **then** $i := i+1$; **else** $i := -1$;
(16) $\quad$ **if** $i = m$ **then return** $j - m + 1$;

Algoritmin 1.4.1 alkuosa (rivit 1–10) löytää hahmolle P kandidaattijoukon $\{i \mid cand[i] = 1\}$, joka on *täydellinen* (eli sisältää jokaisen P:n esiintymän alkuposition S:ssä) ja *konsistentti* (siihen kuuluvat kandidaatit ovat pareittain konsistentteja).

Algoritmin loppuosassa etsitään täsmäävää kandidaattia. Jokainen epätäsmäys eliminoi kaikki sen kohdalla olevat kandidaatit (konsistenssi).

Algoritmi 1.4.1 toimii selvästi ajassa $O(n)$ ja tilassa $O(n+m)$. Taulukko W voidaan muodostaa ajassa $O(m)$.

1.5 Shift-Or-algoritmi

R. Baeza-Yates & G. Gonnet: A new approach to text searching. *Communications of ACM* 35(10):74–82, 1992.

Olkoon w tietokoneen sananpituus. Oletetaan aluksi, että $|P| = m \leq w$.
Idea:

- Ylläpidetään m-bittistä bittivektoria E, joka esittää haun tilanteen kullakin hetkellä. Bitti $E.i = 0$, jos $p[0 \ldots i]$ täsmää tähän asti luetun tekstin loppuun. Eli, kun tekstistä on luettu $S[0 \ldots j]$, $E.i = 0$, jos $P[0 \ldots i] = S[j-i \ldots j]$, muuten $E.i = 1$.

- Esiintymä on löytynyt, kun bitti $E.(m-1) = 0$.
- Tilannetta päivitetään kunkin tekstimerkin kohdalla käyttämällä kullekin merkille $a \in \Sigma$ laskettuja bittivektoreita $T[a]$, joille pätee bitti $T[a].i = 0$, jos $p[i] = a$, muuten $T[a].i = 1$.
- Päivitys toteutetaan kahdessa vaiheessa:
  (1) E:n bittejä siirretään askel vasemmalle ($E' := E << 1$)
  (2) Suoritetaan bittivektorien E' ja $T[s[i]]$ or-operaatio ($E' \mid\mid T[s[i]]$).

Algoritmi 1.5.1 *Shift-Or*

Syöte: $S, P, m \leq w$
Tuloste: P:n esiintymien loppukohdat
(1) **for** $a \in \Sigma$ **do** $T[a] := 2^m - 1$;
(2) **for** $i := 0$ **to** $m-1$ **do** $T[p[i]] := T[p[i]] - 2^i$;
(3) $E := 2^m - 1$;
(4) **for** $j := 0$ **to** $n-1$ **do**
(5) $\quad$ $E := (E << 1) \mid\mid T[s[j]]$;
(6) $\quad$ **if** $E.(m-1) = 0$ **then** write j

Esimerkki 1.5.2 $P =$ `assi`, $S =$ `apassit`, *bittivektorit pystysuorassa*

`apuvektorit`

	a	*i*	*p*	*s*	*t*
a	0	1	1	1	1
s	1	1	1	0	1
s	1	1	1	0	1
i	1	0	1	1	1

`sovituksen vaiheet`

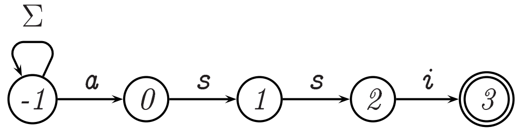
	a	*p*	*a*	*s*	*s*	*i*	*t*
1	0	1	0	1	1	1	1
1	1	1	1	0	1	1	1
1	1	1	1	1	0	1	1
1	1	1	1	1	1	0	1

Varaamalla E:lle useita sanoja, voidaan käsitellä pitempiä hahmoja. Algoritmin ominaisuuksia:

- Selauksen (rivit 4-6) aikavaatimus: $O(\lceil m/w \rceil n)$.
- Alustuksen (rivit 1-2) aikavaatimus: $O(\lceil m/w \rceil \sigma + m)$.
- Tilavaatimus: $O(\lceil m/w \rceil \sigma)$.

Shift-or-algoritmin voidaan myös ajatella simuloivan epädeterminististä hahmonsovitusautomaattia (ks. luku 1.7). Bittivektori E on automaatin tilavektori:
Kun $S[0 \dots j]$ on selattu, $E.i = 0$ joss automaatissa on polku, joka johtaa tilaan i syötteellä $S[0 \dots j]$.

Esimerkki 1.5.3 *Hahmon assi epädeterministinen hahmonsovitusautomaatti.*



1.6 Aho-Corasick-algoritmi

A. Aho & M. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM* 18(6): 333–340, 1975.

- KMP-algoritmin yleistys usealle hahmolle.
- On annettu teksti S, $|S| = n$, ja joukko hahmoja $\mathbb{P} = \{P_1, \dots, P_k\}$. Merkitään $|\mathbb{P}| = m = \sum_{i=1}^{k} |P_i|$.
- On etsittävä S:stä kaikkien $P_i \in \mathbb{P}$ kaikki esiintymät.

Ratkaisuja

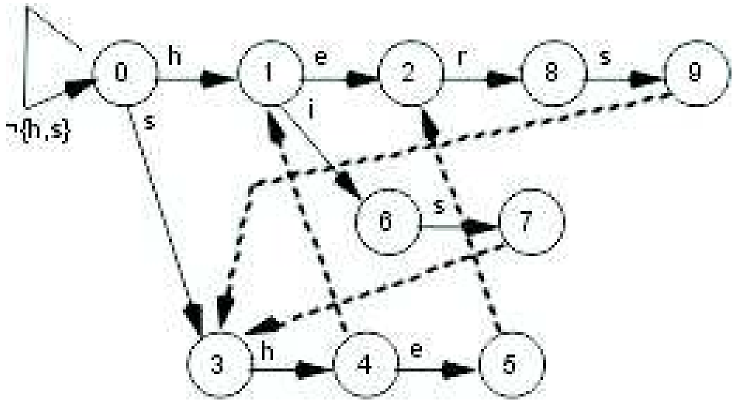
1. Sovelletaan esim. KMP-algoritmia vuorotellen kullakin hahmolla P_i. Aikavaatimus pahimmassa tapauksessa $O(m + kn)$.
2. Aho-Corasick-algoritmi. Teksti S selataan vain kerran. Aikavaatimus on pahimmassa tapauksessa $O(m + n)$.
3. Myös BM-algoritmille ja sen variaatioille on vastaavia yleistyksiä.

Aho-Corasick

Muodostetaan $\mathbb{P}$:lle jononsovitus-automaatti $M_\mathbb{P}$. $M_\mathbb{P}$:n osat:

- Varsinaiset siirtymät: $goto[j, a]$ — tilasta j siirrytään merkillä a tilaan $goto[j, a]$. Jos tilasta j ei ole siirtymää a:lla, $goto[j, a] = \perp$.
- Korjaussiirtymät: $fail[j]$ — jos $goto[j, a] = \perp$ vuorossa olevalla tekstimerkillä a (ts. mikään varsinainen siirtymä ei sovi), siirrytään tilaan $fail[j]$.
- Tulostusfunktio: $output[j]$ — sisältää hahmot, jotka tulostetaan, kun tullaan tilaan j (ts. näiden hahmojen esiintymät on nyt löydetty).

Esimerkki 1.6.1 *Aho-Corasick jononsovitus-automaatti $M_\mathbb{P}$ hahmojoukolle $\mathbb{P} = \{$`he, she, his, hers`$\}$.*



Varsinaiset siirtymät muodostavat $\mathbb{P}$:stä ns. trie-rakenteen. Varsinainen siirtymä (nuoli yhtenäisellä viivalla) lukee tekstistä yhden merkin, esim. $goto[1, i] = 6$. Korjaussiirtymää (nuoli katkoviivalla) seurataan, jos varsinainen ei sovi, esim. $fail[9] = 3$; jokaisesta tilasta ($\neq 0$) lähtee korjaussiirtymä (alkutilaan 0 johtavat korjaussiirtymät on jätetty merkitsemättä).

Tulostusfunktio:

j	$output[j]$
2	{`he`}
5	{`she, he`}
7	{`his`}
9	{`hers`}

Seuraava algoritmi esittää $M_\mathbb{P}$:n toiminnan, kun se selaa S:n:

Algoritmi 1.6.2 *Aho-Corasick-selaus*

Syöte: S ja $M_{\mathbb{P}}$ (=funktiot *goto*, *fail* ja *output* hahmojoukolle $\mathbb{P}$)
Tuloste: Kaikki $\mathbb{P}$:n hahmojen esiintymät (= esiintymien loppukohdat) S:ssä

$state := 0;$
for $j := 0$ **to** $n-1$ **do**
  while $goto[state, S[j]] = \perp$ **do** $state := fail[state];$
  $state := goto[state, S[j]];$
  if $output[state] \neq \emptyset$ **then** $print(j, output[state]);$

Goto-funktion konstruktio

Lisätään hahmot P_i yksitellen trie-rakenteeseen, joka esittää *goto*-funktiota. Kun P_i on lisätty, päivitetään samalla *output*-funktiota. (*fail*-funktion konstruktio täydentää *output*-funktion).

Algoritmi 1.6.3 *Aho-Corasick-esiprosessointi I*
Syöte: $\mathbb{P} = \{P_1, \ldots, P_k\}$
Tuloste: *goto*-funktio ja osittain laskettu *output*-funktio
Oletukset: $output(s)$ on aluksi tyhjä, kun tila s syntyy;
  $goto[s, a] = \perp$, jos $goto[s, a]$ on määrittelemättä
  tai ko. siirtymä ei lainkaan kuulu $M_{\mathbb{P}}$:hen.

procedure $enter(P_i);$
begin
$state := 0; j := 0;$
while $goto[state, P_i[j]] \neq \perp$ **do**
  $state := goto[state, P_i[j]];$
  $j := j + 1;$
for $h := j$ **to** $|P_i| - 1$ **do**
  $newstate := newstate + 1;$
  $goto[state, P_i[h]] := newstate;$
  $state := newstate;$
$output[state] := \{P_i\}$
end
begin
$newstate := 0;$
for $i := 1$ **to** k **do** $enter(P_i);$
for $a \in \Sigma$ **do if** $goto[0, a] = \perp$ **then** $goto[0, a] := 0;$
end

Fail-funktion konstruktio

Laskenta etenee tasoittain alkaen *goto*-trien juuresta. Tasolla d tilojen $fail$-funktion laskemiseksi käydään tason $d-1$ tilat r läpi seuraavasti:

1. Muuten, jokaisella a siten, että $goto[r, a] = s$, $s \neq \perp$:

   a) $state := fail[r]$

   b) Toista $state := fail[state]$ kunnes saavutetaan $state$, jolla $goto[state, a] \neq \perp$. (Sellainen löytyy aina, koska $goto[0, a] \neq \perp$.)

   c) Aseta $fail[s] := goto[state, a]$

Algoritmi 1.6.4 *Aho-Corasick-esiprosessointi II*

Syöte: *goto*-funktio ja alustava *output*-funktio Algoritmista 1.6.3
Tuloste: $fail$-funktio ja lopullinen *output*-funktio

$queue := \emptyset$;
for $a \in \Sigma$ **do if** $goto[0, a] \neq 0$ **then**
    $s := goto[0, a]$;
    $push\_back(queue, s)$;
    $fail[s] := 0$;
while $queue \neq \emptyset$ **do**
    $r := pop\_front(queue)$;
    for $a \in \Sigma$ **do if** $goto[r, a] \neq \perp$ **then**
        $s := goto[r, a]$;
        $push\_back(queue, s)$;
        $state := fail[r]$;
        while $goto[state, a] = \perp$ **do** $state := fail[state]$
        $fail[s] := goto[state, a]$
        $output[s] := output[s] \cup output[fail[s]]$

AC-algoritmin oikeellisuus

Jos tilasta 0 tilaan s johtava *goto*-reitti selaa jonon u, kutsutaan u:ta s-jonoksi. Kaikkien s-jonojen, $s \in M_{\mathbb{P}}$, joukkoa kutsutaan $M_{\mathbb{P}}$-jonoiksi. Jono u on $M_{\mathbb{P}}$-jono, joss se on jonkin hahmon $P_i \in \mathbb{P}$ alkuosa.

Lemma 1.6.5 *Oletetaan, että jono u on s-jono ja v on t-jono automaatissa $M_{\mathbb{P}}$. Silloin $fail[s] = t$, jos ja vain jos v on u:n pisin aito loppuosa, joka on $M_{\mathbb{P}}$-jono.*

Todistus. Käytetään induktiota $|u|$:n (s:n syvyyden) suhteen.

- Kun $|u| = 1$, Alg. 1.6.4 asettaa $fail[s] := 0$. Koska 0-jono on tyhjä jono, joka on u:n ainoa aito loppuosa, väite pätee.
- Oletetaan, että $|u| = j + 1$, $j > 0$, ja että väite on tosi kaikille tiloille, joiden syvyys on enintään j. Olkoon $u[0 \dots j)$ r_0-jono, eli $goto[r_0, u[j]] = s$. Olkoot $r_1, \dots, r_p$ tiloja siten, että
  1. $r_{i+1} = fail[r_i]$, $0 \le i < p$,
  2. $goto[r_i, u[j]] = \perp$, $1 \le i < p$, ja
  3. $goto[r_p, u[j]] = t \ne \perp$.
- Siis $r_1, \dots, r_p$ ovat ne tilat "state", joiden kautta Alg. 1.6.4:n sisin silmukka kulkee. Sen jälkeen algoritmi asettaa: $fail[s] := t$.
- Olkoon v_i r_i-jono, $0 \le i \le p$. Induktio-oletuksesta seuraa, että v_1 on $v_0 = u[0 \dots j)$:n pisin aito loppuosa, joka on $M_{\mathbb{P}}$-jono. Vastaavasti v_2 on v_1:n pisin aito loppuosa, joka on $M_{\mathbb{P}}$-jono, jne.
- v_0:n jokainen aito loppuosa on myös v_1:n loppuosa. Siten v_2:n täytyy olla v_0:n toiseksi pisin aito loppuosa, joka on $M_{\mathbb{P}}$-jono. Vastaavasti v_3 on v_0:n kolmanneksi pisin aito loppuosa, joka on $M_{\mathbb{P}}$-jono, jne.
- Siten v_p on v_0:n pisin aito loppuosa, jolla $v_p u[j] = v$ on $M_{\mathbb{P}}$-jono.
- Siis v on u:n pisin aito loppuosa, joka on $M_{\mathbb{P}}$-jono.

$\square$

Lemma 1.6.6 *Tulosjoukko output$[s]$ sisältää jonon y, jos ja vain jos y on jokin $\mathbb{P}$:n hahmo, joka on s-jonon loppuosa.*

Todistus. Kuten Lemma 1.6.5. $\square$

Lemma 1.6.7 *Kun $M_{\mathbb{P}}$ on selannut (Alg. 1.6.2) $j + 1$ merkkiä S:n alusta, se on tilassa s, jos ja vain jos s-jono on $S[0 \dots j]$:n pisin loppuosa joka on $M_{\mathbb{P}}$-jono.*

Todistus.. Kuten Lemma 1.6.5. $\square$

Lause 1.6.8 *Algoritmi 1.6.2 tulostaa täsmälleen kaikki $\mathbb{P}$:n esiintymät S:stä.*

Todistus. Lemman 1.6.6 perusteella algoritmi ei koskaan tulosta väärää esiintymää. On vielä todistettava, että se tulostaa kaikki esiintymät.

- Olkoon j jonkin P_i:n esiintymän päättymiskohta S:ssä.
- Koska P_i on $S[0 \ldots j]$:n loppuosa ja $M_{\mathbb{P}}$-jono, on algoritmi $S[0 \ldots j]$:n selauksen jälkeen tilassa s, jonka syvyys on vähintään $|P_i|$ (Lemma 1.6.7).
- Siten P_i on s-jonon loppuosa ja algoritmi tulostaa esiintymän (Lemma 1.6.6).

□

AC-algoritmin aikavaatimus

Lause 1.6.9 *Automaatti $M_{\mathbb{P}}$ (Alg. 1.6.2) tekee vähemmän kuin $2n$ tilasiirtymää selatessaan tekstin S, $|S| = n$.*

Todistus. $Fail$-siirtymiä voi olla korkeintaan $goto$-siirtymien määrä -1, koska jokainen $fail$-siirtymä johtaa aidosti lähemmäksi tilaa 0 (eli syvyys vähenee). Koska $goto$-siirtymiä on n kpl, $fail$-siirtymiä voi siis olla korkeintaan $n - 1$ kpl, joten siirtymiä on yhteensä vähemmän kuin $2n$ kpl. □

Lauseessa 1.6.9 Algoritmille 1.6.2 saatu vertailujen määrän $\mathcal{O}(n)$ raja ei välttämättä todista, että todellinen aikavaatimus olisi myös $\mathcal{O}(n)$. Aikavaatimukseen vaikuttavat myös seuraavat operaatiot:

1. $goto[q, a]$:n evaluointi	$\mathcal{O}(1)$ tai $\mathcal{O}(\log \min(k, \sigma))$
2. $fail[q]$:n evaluointi	$\mathcal{O}(1)$
3. testaus: $output[q] = \emptyset$	$\mathcal{O}(1)$
4. $output[q]$:n tulostaminen	?

Jos kaikki funktiot esitetään taulukkoina, Alg. 1.6.2:n aikavaatimus on $\mathcal{O}(n + r)$, missä r on esiintymien kokonaismäärä. Jos tila ei salli taulukoita (jos k ja σ suuria), voidaan käyttää listoja tai hakupuita, jotka antavat aikavaatimuksen $\mathcal{O}(n \log(\min(k, \sigma)) + r)$.

Keskitytään jatkossa taulukkototeutukseen.

Lause 1.6.10 *Lause Alg. 1.6.3 laskee goto-funktion ja alustaa output-funktion ajassa $\mathcal{O}(\sigma m)$.*

Todistus. Helppo. □

Lause 1.6.11 *Lause Alg. 1.6.4 laskee $fail$-funktion ja viimeistelee output-funktion ajassa $\mathcal{O}(\sigma m)$.*

Todistus. Väite seuraa helposti, kun huomioidaan:

- Rivin (12) silmukka suoritetaan alle m kertaa, koska $fail$-siirtymien määrä voidaan rajata alle m:n kuten lauseen 1.6.9 todistuksessa.
- $output$-funktioiden yhdiste voidaan toteuttaa listojen katenaationa ajassa $\mathcal{O}(1)$.

□

Seuraus 1.6.12 *AC-algoritmin kokonaisaikavaatimus on $\mathcal{O}(n + \sigma m + r)$.*

Jos aakkoston koko on vakio ja tulosteen koko oletetaan pieneksi, aikavaatimus on $\mathcal{O}(n + m)$.

Koetulos: $n = 10^7$, etsintäaika tunteina

	$k = 15$	$k = 24$
triv.ratk.	0.79	1.27
AC	0.18	0.21

1.7 Hahmonsovitus äärellisen automaatin avulla

Edellä on tarkasteltu tapauksia: yksi hahmo ja joukko hahmoja. Seuraavassa tarkastellaan yleisempää tapausta, jossa etsittävien hahmojen joukko on ilmaistu säännöllisenä lausekkeena. Aluksi hieman kertausta:

Määritelmä 1.7.1 *Säännölliset lausekkeet:*

1. *$\emptyset$ on säännöllinen lauseke; $\emptyset$ esittää tyhjää merkkijonojoukkoa.*
2. *ϵ on säännöllinen lauseke; ϵ esittää joukkoa $\{\epsilon\}$, ϵ = tyhjä jono.*
3. *Kaikilla $a \in \Sigma$, a on säännöllinen lauseke; a esittää joukkoa $\{a\}$.*
4. *Jos A ja B ovat säännöllisiä lausekkeita ja $L(A)$ ja $L(B)$ ovat näiden esittämät joukot, seuraavat lausekkeet ovat säännöllisiä lausekkeita:*

$$\begin{aligned} (A \cup B) \text{ yhdiste} \quad &; \quad L(A \cup B) = L(A) \cup L(B) \\ (AB) \text{ katenaatio} \quad &; \quad L(AB) = L(A)L(B) \\ (A^*) \text{ sulkeuma} \quad &; \quad L(A^*) = L(A)^* \end{aligned}$$

Ongelma 1.7.2 *On annettu säännöllinen lauseke A (= hahmojoukon esitys) ja teksti S.*

O1. Onko $S \in L(A)$?

O2. Anna kaikki S:n alkuosat, jotka kuuluvat joukkoon $L(A)$.

O3. Etsi kaikki $L(A)$:n jonojen esiintymät S:stä.

- O1:n ratkaisu: Muodostetaan M_A, $L(M_A) = L(A)$. Selataan S M_A:lla. Tutkitaan lopussa, onko M_A lopputilassa.
- O2:n ratkaisu: Kuten O1, mutta tutkitaan jokaisen merkin jälkeen, onko M_A lopputilassa.
- O3:n ratkaisu: Ratkaistaan O2 lausekkeella $\Sigma^* A$ (missä Σ on kaikkien aakkoston merkkien yhdiste $a \cup b \cup \ldots$).

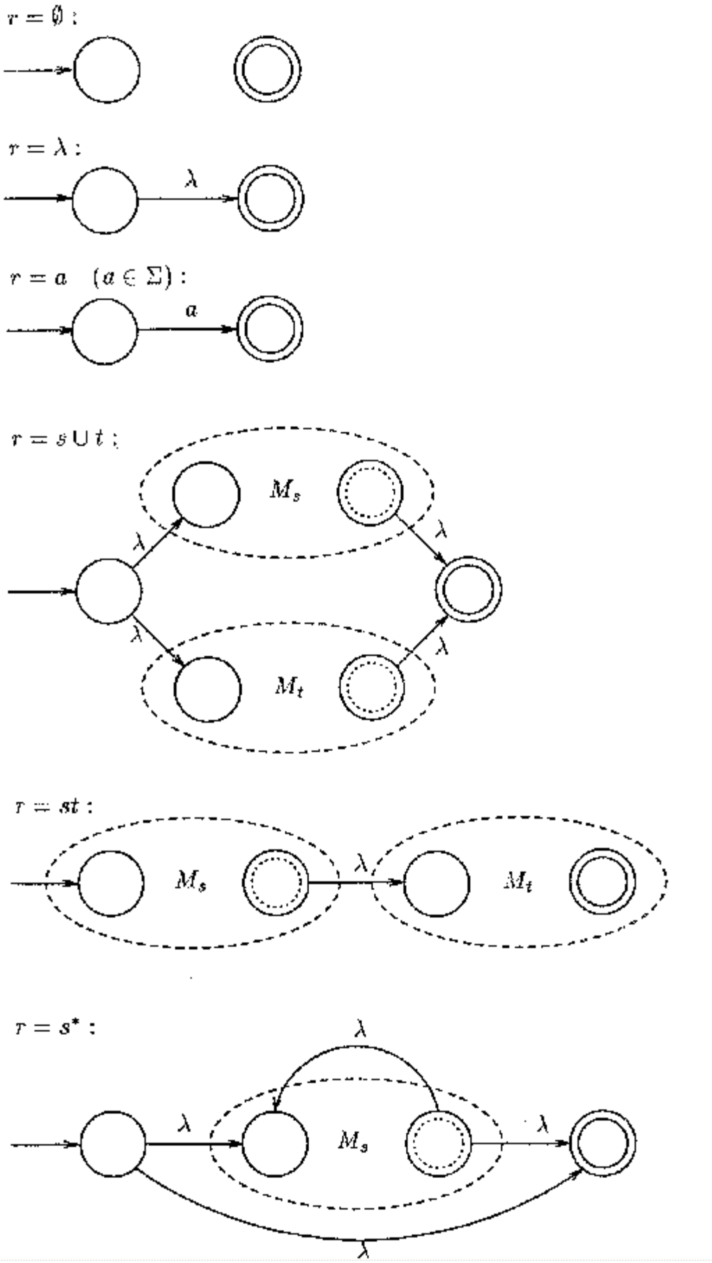
Tarkastellaan jatkossa lähinnä ongelmaa O1.

Määritelmä 1.7.3 *Epädeterministinen äärellinen automaatti (NFA) $M = (Q, \Sigma, \delta, q_0, F)$ muodostuu osista:*

1. *Q on äärellinen tilojen joukko.*
2. *Σ on syöttöaakkosto.*
3. *δ on tilasiirtymäfunktio, $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to Q$:n osajoukot ($\delta$ siis sallii useita siirtymiä samalla aakkosella).*
4. *$q_0 \in Q$ on alkutila.*
5. *$F \subseteq Q$, lopputilojen joukko.*

Nyt voidaan myös määritellä:

- *Siirto*: $(q, aw) \vdash (q', w)$ jos $q' \in \delta(q, a)$, $a \in \Sigma \cup \{\epsilon\}$, $w \in \Sigma^*$.
- *Hyväksytty kieli*: $L(M) = \{w \mid (q_0, w) \vdash^* (q, \epsilon), q \in F\}$, missä "$\vdash^*$" tarkoittaa "0:lla tai useammalla siirrolla".
- Konstruktio: ks. esim. Ohjelmoinnin ja laskennan perusmallit -kurssi tai Kuva 1.2.



Kuva 1.2: NFA:n konstruktio säännöllisestä lausekkeesta. Kuvassa λ tarkoittaa ϵ:ia.

Lause 1.7.4 *Olkoon NFA $M_A = (Q, \Sigma, \delta, q_0, \{q_f\})$ konstruoitu A:sta em. tekniikalla. Silloin $L(M_A) = L(A)$ ja*

(1) $|Q| = 2|A|$ ($|A|$ on A:n pituus merkkeinä).

(2) $\delta(q_f, a) = \emptyset$ kaikilla $a \in \Sigma \cup \{\epsilon\}$.

(3) Kustakin tilasta $q \in Q$ lähtee korkeintaan 2 siirtymää.

Todistus. $L(M_A) = L(A)$: ks. esim. Ohjelmoinnin ja laskennan perusmallit -kurssi. Väite (2) pätee selvästi konstruktion nojalla. Väitteet (1) ja (3) osoitetaan induktiolla $|A|$:n suhteen. □

Tekstin S selaaminen automaatilla M_A

Tarkastellaan, miten NFA M_A:n vaihtoehtoisia polkuja voidaan seurata samanaikaisesti. Vaihtoehtoinen ratkaisutapa olisi muodostaa ensin M_A:sta vastaava deterministinen äärellinen automaatti M'_A. M'_A voi olla hyvin suuri: $|M'_A| = O(2^t)$, missä t on M_A:n tilojen lukumäärä, joten M'_A :n muodostaminen ei aina kannata, varsinkaan jos S on lyhyt. M'_A:n etuna on, että S voidaan selata ajassa $O(|S|)$.

Merkitään Q_j:llä niiden tilojen joukkoa, joissa M_A voi olla luettuaan jonon $S[0 \ldots j) : Q_j = \{q \mid (q_0, S[0 \ldots j)) \vdash^* (q, \epsilon)\}$. Silloin: $q \in Q_j$, jos ja vain jos on olemassa $q' \in Q_{j-1}$ siten, että tilasta q' siirrytään tilaan q_1 merkillä $S[j-1]$ ja tilasta q_1 siirrytään 0:lla tai useammalla ϵ-siirrolla tilaan q.

Algoritmi 1.7.5 *NFA:n simulointi*
Syöte: NFA $M_A = (Q, \Sigma, \delta, q_0, \{q_f\})$, Teksti $S = S[0 \ldots n)$.
Tuloste: Tilajoukkojono $Q_0, Q_1, \ldots, Q_n$.
Huom. $S \in L(M_A)$, jos ja vain jos $q_f \in Q_n$.
Jono $queue$ ja joukot $Q_0, Q_1, \ldots, Q_n$ ovat aluksi tyhjiä.

for $j := 0$ **to** n **do**
    merkitse kukin $q \in Q$ saavuttamattomaksi;
    if $j = 0$ **then** $Q_0 := \{q_0\}$; $queue := \{q_0\}$; merkitse q_0 saavutetuksi
    else
        for $q \in Q_{j-1}$ **do for** $p \in \delta(q, S[j-1])$ **do**
            if p on saavuttamaton **then**
                $Q_j := Q_j \cup \{p\}$;
                $push\_back(queue, p)$;
                merkitse p saavutetuksi;
    while $queue \neq \emptyset$ **do**

(11) $\qquad q := pop\_front(queue)$;
(12) $\qquad$ **for** $p \in \delta(q, \epsilon)$ **do**
(13) $\qquad\qquad$ **if** p on saavuttamaton **then**
(14) $\qquad\qquad\qquad Q_j := Q_j \cup \{p\}$;
(15) $\qquad\qquad\qquad push\_back(queue, p)$;
(16) $\qquad\qquad\qquad$ merkitse p saavutetuksi;

Riveillä 10-16 seurataan ϵ-siirtymiä ja se osa voitaisiin jättää pois, jos automaatissa ei olisi ϵ-siirtymiä. Niiden poisto (ks. esim. Ohjelmoinnin ja laskennan perusmallit) saattaisi kuitenkin kasvattaa siirtymien määräksi $\mathcal{O}(|Q|^2)$.
Merkitään: $||M_A||$ = tilojen lkm + siirtymien lkm.

Lause 1.7.6 *NFA M_A:n simulointi Algoritmilla 1.7.5, kun syöte on $S = s[0 \ldots n)$, vie ajan $O(||M_A|| \cdot n)$.*

Todistus.

- Rivi 2 vie ajan $O(|Q|)$.
- Siirtymäfunktion δ evaluointi voidaan toteuttaa ajassa $\mathcal{O}(1)$, koska kustakin tilasta on enintään kaksi siirtymää. Kaikki muutkin operaatiot toimivat vakioajassa.
- Riveillä 3–16 viedään vain saavuttamattomia tiloja jonoon ja merkitään ne samalla saavutetuiksi. Sitten seurataan saavutetuista tiloista alkavia siirtymiä, joten kutakin siirtymää seurataan korkeintaan kerran. Yhden siirtymän käsittely vie $O(1)$ ajan, joten aikaa kuluu O(siirtymien lkm).
- Rivit 2–16 toistetaan n kertaa. Siis kokonaisaika on $O(||M_A|| \cdot n)$.

$\square$

Seuraus 1.7.7 *Koska $||M_A|| \leq 6 \cdot |A|$ (Lause 1.7.4), voidaan*

- *testata onko $S \in L(A)$ ajassa $\mathcal{O}(|A| \cdot |S|)$ (ongelma O1),*
- *laskea S:n alkuosat, jotka ovat $L(A)$:ssa, ajassa $\mathcal{O}(|A| \cdot |S|)$ (ongelma O2), ja*
- *löytää S:stä $L(A)$:n jonojen esiintymät ajassa $O((|\Sigma| + |A|) \cdot |S|)$ (ongelma O3).*

Todistus. O2: Ajassa $\mathcal{O}(|A|)$ voidaan tutkia, sisältääkö Q_j lopputilan.
O3: $|\Sigma^* A| = O(|\Sigma| + |A|)$. $\square$