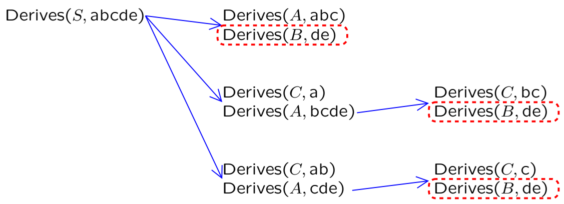
Rekursiivinen Derives on periaatteessa aivan toimiva algoritmi, mutta *erittäin* tehoton. Jos tarkastellaan esim. kieliopinpätkää

$$\begin{array}{lcl} S & \to & AB \mid CA \mid \ldots \\ A & \to & CB \mid \ldots \\ & \ldots & \end{array}$$

ja kutsua Derives(S, abcde), niin kutsu Derives(B, de) tulee toistetuksi ainakin kolme kertaa:



Ongelma muuttuu vielä paljon pahemmaksi pitkillä merkkijonoilla. Algoritmin aikavaativuus on eksponentiaalinen merkkijonon w pituuden suhteen. Sitä voidaan oleellisesti tehostaa käyttämällä *taulukointia* eli *dynaamista ohjelmointia* (dynamic programming).

Turhaa laskentatyötä voidaan vähentää pistämällä kerran lasketut välitulokset taulukkoon. Muodostetaan siis $n \times n$ taulukko $table(1..n, 1..n)$, missä $n = |w|$ on alkuperäisen merkkijonon pituus ja taulukon alkio $table(i, j)$ listaa kaikki ne muuttujat, joista voidaan johtaa merkkijono $w_i \ldots w_j$:

$$table(i, j) = \left\{ A \in V \mid A \overset{*}{\Rightarrow} w_i \ldots w_j \right\}.$$

Edellä käytetty periaate tapauksessa $i < j$ tulee nyt seuraavaan muotoon: jos

- kieliopissa on sääntö $A \to BC$ ja
- tiedetään $B \in table(i, k)$ (eli $B \overset{*}{\Rightarrow} w_i \ldots w_k$) ja
- tiedetään $C \in table(k + 1, j)$ (eli $C \overset{*}{\Rightarrow} w_{k+1} \ldots w_j$)

niin voidaan päätellä

- $A \in table(i, j)$ (eli $A \overset{*}{\Rightarrow} w_i \ldots w_j$).

Tapauksessa $i = j$ pätee $A \in table(i, i)$, jos ja vain jos kieliopissa on sääntö $A \to w_i$.

Edellä esitetyn perusteella arvo $table(i, j)$ osataan laskea, jos tunnetaan $table(p, q)$ indekseille (p, q) joilla $q - p < j - i$. Taulukko siis kannattaa täyttää seuraavasti:

vaiheessa 1 täytetään alkiot $(1, 1), (2, 2), (3, 3), \ldots, (n, n)$.

vaiheessa 2 täytetään alkiot $(1, 2), (2, 3), (3, 4) \ldots, (n - 1, n)$.

vaiheessa 3 täytetään alkiot $(1, 3), (2, 4), (3, 5) \ldots, (n - 2, n)$.

...

vaiheessa ℓ täytetään alkiot $(1, \ell), (2, \ell + 1), (3, \ell + 2) \ldots, (n - \ell + 1, n)$.

...

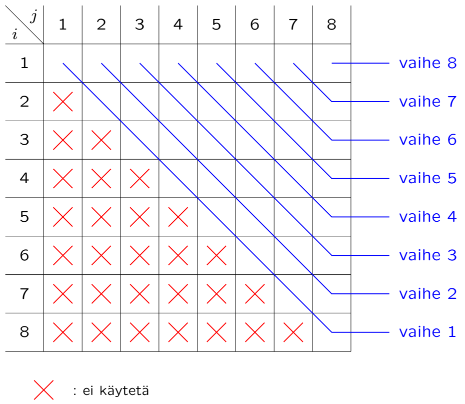
vaiheessa $n - 1$ täytetään alkiot $(1, n - 1)$ ja $(2, n)$

vaiheessa n täytetään alkio $(1, n)$.

Siis vaiheessa ℓ täytetään alkiot $table(i, j)$, missä $j = i + \ell - 1$.

Lopuksi todetaan $w \in L(G)$, jos ja vain jos $S \in table(1, n)$.

Taulukon $table$ täyttämisjärjestys tapauksessa $|w| = 8$. Alkioita $table(i, j)$ missä $j < i$ ei käytetä.

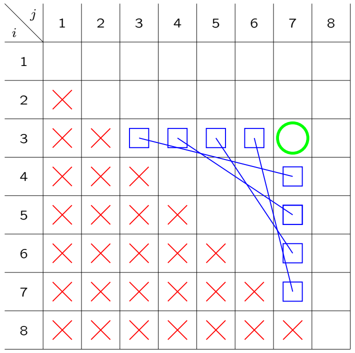


Alkion $table(i,j)$ täyttämiseen tarvitaan

- alkiot $table(i,i), \ldots, table(i,j-1)$ ja
- alkiot $table(i+1,j), \ldots, table(j,j)$.

Säännön $A \rightarrow BC$ perusteella $A \in table(3,7)$, jos jokin seuraavista pätee:

$$
\begin{array}{lll}
B \in table(3,3) & \text{ja} & C \in table(4,7) \\
B \in table(3,4) & \text{ja} & C \in table(5,7) \\
B \in table(3,5) & \text{ja} & C \in table(6,7) \\
B \in table(3,6) & \text{ja} & C \in table(7,7)
\end{array}
$$



Saadaan seuraava algoritmi [Sipser s. 267]:

```
CYK(G, w)
    ▷ Oletus: G = (V, Σ, R, S) on Chomskyn normaalimuodossa
    if w = ε and S → ε on sääntö then return True
    alusta table(i, j) ← ∅ kaikilla i ja j
    for i ← 1 to |w|                ▷ vaihe 1
        do for kaikille muuttujille A ∈ V
            do if A → w_i on sääntö
                then table(i, i) ← table(i, i) ∪ { A }
    for ℓ ← 2 to n                  ▷ vaiheet 2, ..., n
        do for i ← 1 to n − ℓ + 1
            do j ← i + ℓ − 1
                for k ← i to j − 1
                    do for kaikille säännöille A → BC
                        do if B ∈ table(i, k) and C ∈ table(k + 1, j)
                            then table(i, j) ← table(i, j) ∪ { A }
    if S ∈ table(1, n) then return True else return False
```


Esimerkki 2.18: Tarkastellaan merkkijonoa $w =$ abba ja kielioppia

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & AC \mid \mathrm{a} \\
B & \rightarrow & AB \mid \mathrm{b} \\
C & \rightarrow & \mathrm{a} \mid \mathrm{b}
\end{array}
$$

Seuraavan taulukon mukaan $S \in table(1,4)$ eli w kuuluu kieleen. Taulukkoon on lisäksi merkitty, mitkä välivaiheet ovat tuoneet muuttujan S joukkoon $table(1,4)$.

$i \backslash j$	1	2	3	4
1	A C	S A B	S A B	S A
2		B C	S	$\emptyset$
3			B C	S
4				A C

□

Yhteenveto:

- Chomskyn normaalimuoto takaa, että lyhyille merkkijonoille myös johdot ovat lyhyitä (vrt. sivu 151), joten jäsentäminen on ainakin periaatteessa mahdollista syötteen pituuden suhteen polynomisessa ajassa.
- Naiivi rekursiivinen Derives toimii oikein, mutta ei tehokkaasti.
- Varsinainen CYK-algoritmi laskee samat asiat kuin Derives, mutta organisoi laskennan tehokkaammin.

CYK on esimerkki keskeisestä algoritmisuunnittelutekniikasta, jota kutsutaan *taulukoinniksi* eli *dynaamiseksi ohjelmoinniksi*:

- Ongelma jaetaan osaongelmiin.
- Ensin ratkaistaan *kaikki* pienemmät osaongelmat. Niiden ratkaisuista voidaan koota ratkaisut suurempiin osaongelmiin.
- Lopuksi saadaan ratkaisu koko ongelmaan. Turhalta näyttävä pienempien osaongelmien ratkaiseminen vähentää kuitenkin kokonaistyömäärää.

CYK-algoritmissa kolme sisäkkäistä **for**-silmukkaa, joita kutakin iteroidaan $O(n)$ kertaa. Siis algoritmin aikavaativuus on $O(n^3)$, jos kieliopin koko lasketaan vakioksi.

Korollaari 2.19: [Sipser Thm. 7.16] Millä tahansa yhteydettömällä kielellä L kysymys "päteekö $w \in L$?" voidaan ratkaista ajassa $O(|w|^3)$. □

Kuten edellä mainittiin, CYK-algoritmi ei ole realistinen vaihtoehto ohjelmointikielen jäsentämiseen tms. Sovelluksissa käytetään tehokkaampia algoritmeja, jotka kuitenkin toimivat vain sopivasti rajatulle luokalle kielioppeja.

Pinoautomaatit (pushdown automaton, PDA) [Sipser luku 2.2]

Pinoautomaatti on äärellinen automaatti, johon on lisätty rajoittamaton määrä muistia *pinon* (stack) muodossa. Näemme jatkossa, että kielen voi tunnistaa pinoautomaatilla, jos ja vain jos se on yhteydetön.

Esimerkki 2.20: Kielen $\{ 0^n 1^n \mid n \in \mathbb{N} \}$ voi hyväksyä pinon avulla seuraavaan tapaan:

1: **jos** *seuraava merkki* on 0 **niin**
    Push(0)
    mene kohtaan 1
  muuten
    Pop
    mene kohtaan 2
2: **jos** *seuraava merkki* on 1 **niin**
    Pop
    mene kohtaan 2

Merkkijono hyväksytään, jos pino on tyhjä kun merkkijono loppuu. □

Ryhdymme nyt formalisoimaan pinoautomaattia. Tarkastelemme alusta alkaen *epädeterministisiä* pinoautomaatteja. Äärellisistä automaateista poiketen epädeterministinen pinoautomaatti on aidosti determinististä vahvempi laskennan malli (ks. esim. 2.22).

Pinoautomaatti on kuusikko $(Q, \Sigma, \Gamma, \delta, q_0, F)$, missä

1. Q on äärellinen tilajoukko,
2. Σ on äärellinen syöteaakkosto,
3. Γ on äärellinen pinoaakkosto,
4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ on siirtymäfunktio,
5. $q_0 \in Q$ on alkutila ja
6. $F \subseteq Q$ on hyväksyvien tilojen joukko.

Määritelmä muistuttaa epädeterministisen äärellisen automaaton määritelmää; eroina ovat pinoaakkosto ja siirtymäfunktio.

Siirtymäfunktio on siis funktio

$$\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon).$$

Muistetaan, että $\Sigma_\varepsilon = \Sigma \cup \{ \varepsilon \}$ ja $\mathcal{P}$ tarkoittaa potenssijoukkoa.

Siirtymäfunktion tulkinta on seuraava:

> Jos $(q', c) \in \delta(q, a, b)$, niin tilassa q voidaan lukea syötemerkki a, poimia pinon päältä merkki b, siirtyä tilaan q' ja painaa pinoon merkki c.

Tässä a, b tai c voi olla myös ε, jolloin tulkinta on:

- jos $a = \varepsilon$, niin seuraavaa syötemerkkiä ei lueta;
- jos $b = \varepsilon$, niin pinosta ei poimita (eikä lueta) mitään; ja
- jos $c = \varepsilon$, niin pinoon ei paineta mitään.

Muodollisesti pinoautomaatti $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ *hyväksyy* merkkijonon $w \in \Sigma^*$, jos jollain $m \in \mathbb{N}$ on olemassa

- $w_1 \in \Sigma_\varepsilon$, ..., $w_m \in \Sigma_\varepsilon$, joilla $w = w_1 \ldots w_m$;
- tilat $r_0 \in Q$, ..., $r_m \in Q$ ja
- merkkijonot $s_0 \in \Gamma^*$, ..., $s_m \in \Gamma^*$ ("pinon sisällöt"),

missä

- $r_0 = q_0$ ja $s_0 = \varepsilon$ (aluksi pino tyhjä)
- kaikilla $i = 0, \ldots, m-1$ voidaan kirjoittaa $s_i = at$ ja $s_{i+1} = bt$, missä $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, ja
- $r_m \in F$.

Tuttuun tapaan automaatti *hylkää* merkkijonot, joita se ei hyväksy. Automaatin *tunnistama kieli* koostuu sen hyväksymistä merkkijonoista.

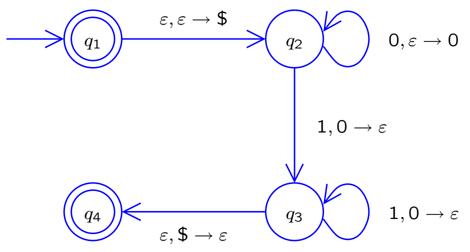
Esimerkki 2.21: [Sipser Ex. 2.14] Kieli $\{ 0^n 1^n \mid n \in \mathbb{N} \}$ voidaan tunnistaa pinoautomaatilla $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$, missä $Q = \{ q_1, q_2, q_3, q_4 \}$, $\Sigma = \{ 0, 1 \}$, $\Gamma = \{ 0, \$ \}$, $F = \{ q_1, q_4 \}$ ja δ on seuraava:

syöte:	0			1			ε		
pino:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{ (q_2, \$) \}$
q_2			$\{ (q_2, 0) \}$	$\{ (q_3, \varepsilon) \}$					
q_3				$\{ (q_3, \varepsilon) \}$				$\{ (q_4, \varepsilon) \}$	
q_4									

Siis esim. $\delta(q_3, \varepsilon, \$) = \{ (q_4, \varepsilon) \}$.

Koska formalismissa ei ole sisäänrakennettua pinon tyhjyystestiä, pinon pohjan merkiksi laitetaan \$. Sama tekniikka toistuu jatkossa.

Esitetään sama havainnollisemmin tilakaaviona. Merkintä $q \xrightarrow{a,b \to c} q'$ tarkoittaa $(q', c) \in \delta(q, a, b)$.

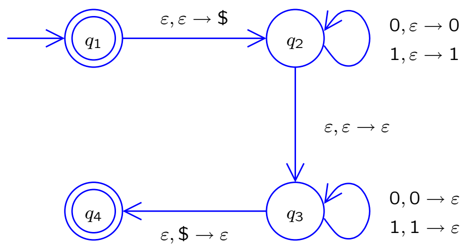


□

Esimerkki 2.22: [Sipser Ex. 2.18] Kieli $\{ ww^{\mathcal{R}} \mid w \in \{ 0, 1 \}^* \}$ voidaan tunnistaa seuraavalla periaatteella:

1. Syötteen ensimmäisen puolikkaan ajan paina merkkejä pinoon.
2. Syötteen toisen puolikkaan ajan poimi merkkejä pinosta ja vertaa juuri luettuun.

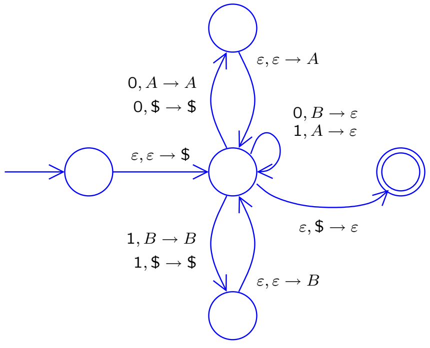
Syötteen keskikohta arvataan epädeterministisesti. Itse asiassa tätä kieltä ei edes ole mahdollista tunnistaa PDA:n deterministisellä variantilla.



□

Esimerkki 2.23: Seuraava PDA hyväksyy ne merkkijonot, joissa on yhtä monta nollaa kuin ykköstä.

Pinoaakkosto on $\Gamma = \{ A, B, \$ \}$. Pinossa pidetään kirjaa tähän mennessä luettujen nollien ja ykkösten määrien erosta. Esim. $AAA\$$ tarkoittaa, että nollia on ollut kolme enemmän, ja $BBBBB\$$, että ykkösiä on ollut viisi enemmän.



$\square$

Yhteydettömien kielioppien ja pinoautomaattien yhteys

[Sipser s. 117–124]

Näemme, että yhteydettömien kielioppien tuottamat kielet ovat tasan samat kuin ne, jotka voidaan tunnistaa pinoautomaatilla. Aloitetaan helpommasta suunnasta.

Lemma 2.24: [Sipser Lemma 2.21] Jos kieli on yhteydetön, se voidaan tunnistaa pinoautomaatilla.

Todistuksen perusidea on laatia annetun kieliopin pohjalta pinoautomaatti, joka toteuttaa seuraavan algoritmin:

Generoi: Tuota epädeterministisesti pinoon merkkijono $w \in \Sigma^*$, jolla $S \overset{*}{\Rightarrow} w$.

Testaa: Vertaa pinon merkkijono syötteeseen merkki kerrallaan. Jos löytyy ero, hylkää. Jos pino tyhjenee samaan aikaan, kun syöte loppuu, niin hyväksy.

Epädeterminismi on oleellista: valitsemalla generoimisvaiheessa sovellettavat säännöt epädeterministisesti varmistetaan, että jokaisella kieleen kuuluvalla merkkijonolla w on mahdollisuus tulla tuotetuksi.

Toteutusta rajoittaa, että automaatin tietorakenne on pino, josta vain huippu on kulloinkin näkyvissä. Siksi generointi- ja testausvaihe pitää lomittaa: aina kun pinon huipulle saadaan päätesymboleita, käydään vertaamassa niitä syötteeseen ennen generoinnin jatkamista.

Saadaan tarkennettu algoritmi:

1. Alusta pinon sisällöksi $S\$$, missä S on lähtösymboli ja \$ merkitsee pinon pohjaa.

2. Toista seuraavaa:
   (a) Jos pinon huipulla on muuttujasymboli A, valitse epädeterministisesti sääntö $A \to w$. Korvaa A merkkijonolla w.
   (b) Jos pinon huipulla on päätesymboli, poista se pinosta ja vertaa seuraavaan syötemerkkiin. Jos ne eroavat, hylkää.
   (c) Jos pinon huipulla on \$, hyväksy jos syöte on loppu; muuten hylkää.

Tämä pitää vielä koodata pinoautomaatiksi.

Esimerkki 2.25: Tarkastellaan kielioppia

$$\begin{array}{lcl} S & \to & S\mathsf{a} \mid T \\ T & \to & \mathsf{b}T\mathsf{c} \mid \varepsilon \end{array}$$

ja merkkijonoa bbcca, jolla on johto

$$S \Rightarrow S\mathsf{a} \Rightarrow T\mathsf{a} \Rightarrow \mathsf{b}T\mathsf{ca} \Rightarrow \mathsf{bb}T\mathsf{cca} \Rightarrow \mathsf{bbcca}.$$

Haluamme muodostaa pinoautomaatin, joka syötteellä bbcca vuorotellen

- soveltaa pinoon ylläolevan johdon sääntöjä ja
- poistaa pinosta syötettä vastaavia päätemerkkejä.

Pinon käyttäytymisen pitäisi siis olla seuraava:

jäljellä oleva syöte	pinon sisältö
bbcca	ε
bbcca	\$
bbcca	S\$
bbcca	Sa\$
bbcca	Ta\$
bbcca	bTca\$
bcca	Tca\$
bcca	bTcca\$
cca	Tcca\$
cca	cca\$
ca	ca\$
a	a\$
ε	\$
ε	ε

$\square$

Jatkossa sallimme merkinnän

$$(r, u) \in \delta(q, a, s)$$

myös, kun $u = u_1 \ldots u_l \in \Gamma^*$ (siis myös kun $l > 1$). Merkintä tarkoittaa

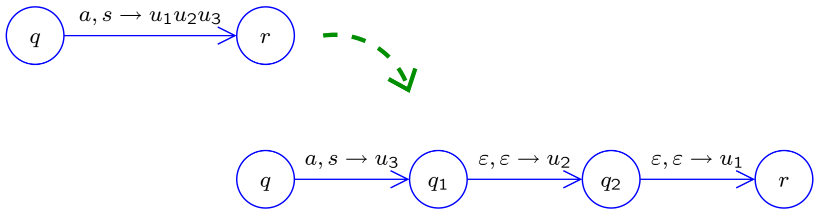
> tilassa $q \in Q$ automaatti saa siirtyä tilaan $r \in Q$ lukemalla syötemerkin $a \in \Sigma$ ja poimimalla pinosta merkin $s \in \Gamma$ ja painamalla pinoon merkit $u_l, \ldots, u_1$.

Myös tapaukset $a = \varepsilon$ ja $s = \varepsilon$ sallitaan kuten ennenkin.

Muodollisesti merkintä tarkoittaa, että automaatissa on tilat $q_1, \ldots, q_{l-1}$, joilla

$$\begin{aligned}
(q_1, u_l) &\in \delta(q, a, s) \\
\delta(q_1, \varepsilon, \varepsilon) &= \{\, (q_2, u_{l-1}) \,\} \\
\delta(q_2, \varepsilon, \varepsilon) &= \{\, (q_3, u_{l-2}) \,\} \\
&\cdots \\
\delta(q_{l-1}, \varepsilon, \varepsilon) &= \{\, (r, u_1) \,\}
\end{aligned}$$

ja joihin ei liity muita siirtymiä. Merkinnän $(r, u_1u_2u_3) \in \delta(q, a, s)$ kaavioesitys:

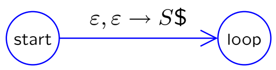


Lemman 2.24 todistus: Olkoon $G = (V, \Sigma, R, S)$ yhteydetön kielioppi. Konstruoimme edellä esitetyn idean mukaisesti pinoautomaatin, joka tunnistaa kielen $L(G)$. Automaatin runkona on kolme tilaa q_{start}, q_{loop} ja q_{accept}. Alkutilana on q_{start} ja ainoa hyväksyvä tila q_{accept}.

Aluksi viedään pinoon loppumerkki \$ ja alkusymboli S siirtymällä

$$\delta(q_{\text{start}}, \varepsilon, \varepsilon) = \{\, (q_{\text{loop}}, S\$) \,\};$$

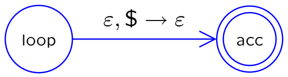
kuvana



Lopuksi pinosta poimitaan loppumerkki \$ siirtymällä

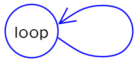
$$\delta(q_{\text{loop}}, \varepsilon, \$) = \{\, (q_{\text{accept}}, \varepsilon) \,\};$$

kuvana



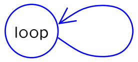
Varsinainen laskenta tapahtuu kahdenlaisilla siirtymillä tilasta q_{loop} itseensä:

- Kaikilla joukon R säännöillä $A \to w$ tulee $(q_{\text{loop}}, w) \in \delta(q_{\text{loop}}, \varepsilon, A)$; kuvana



  Tässä siis pinon huipulla olevasta muuttujasta A tuotetaan pinon huipulle merkkijono w.

- Kaikilla $a \in \Sigma$ tulee $(q_{\text{loop}}, \varepsilon) \in \delta(q_{\text{loop}}, a, a)$; kuvana



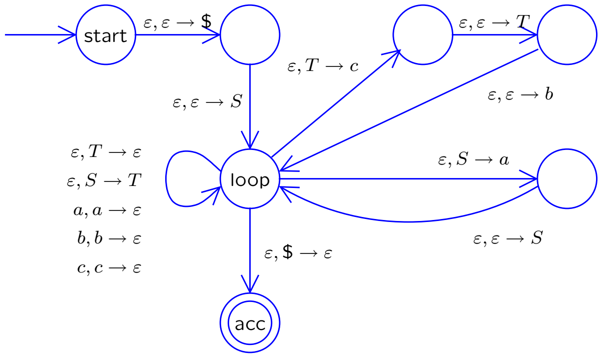
  Tässä siis pinon huipulla oleva päätemerkki a poistetaan ja samalla luetaan vastaava syötemerkki.

Muita siirtymiä ei ole. Selvästi automaatti toteuttaa esitetyn epädeterministisen algoritmin kielen $L(G)$ tunnistamiseksi. □

Esimerkki 2.26: Edellisen esimerkin kielioppista

$$\begin{array}{lcl} S & \to & S\mathsf{a} \mid T \\ T & \to & \mathsf{b}T\mathsf{c} \mid \varepsilon \end{array}$$

saadaan seuraava pinoautomaatti:



□

Haluamme todistaa myös käänteisen suunnan:

Lemma 2.27: [Sipser Lemma 2.27] Pinoautomaatin tunnistama kieli on yhteydetön.

Tämä suunta on hankalampi: meidän pitää osata "purkaa" mikä tahansa pinoautomaatti M ja kuvata sen tunnistama kieli yhteydettömänä kielioppina. Hahmottelemme seuraavassa lemman 2.27 todistuksen esimerkkinä hieman monimutkaisemmasta automaattiteoreettisesta konstruktiosta. Lemman todistus kuitenkaan *ei kuulu koealueeseen*.

Teknisten yksityiskohtien helpottamiseksi todetaan ensin, että automaatista M on helppo muodostaa sellainen saman kielen tunnistava pinoautomaatti M', että

1. automaatissa M' on tasan yksi hyväksyvä tila (merkitään sitä q_{accept})
2. ennen hyväksymistä M' aina tyhjentää pinonsa ja
3. siirtymiä on vain seuraavia lajeja:
   pop-siirtymä: poimii pinosta yhden merkin, ei paina mitään
   $(a, s \to \varepsilon$ missä $a \in \Sigma_\varepsilon$ ja $s \in \Gamma)$ ja
   push-siirtymä: painaa pinoon yhden merkin, ei poimi mitään
   $(a, \varepsilon \to s$ missä $a \in \Sigma_\varepsilon$ ja $s \in \Gamma)$.

Ehdot 1 ja 2 saadaan helposti voimaan lisäämällä pinon loppumerkki ja ylimääräiset pinontyhjennystila ja hyväksyvä tila.

Ehto 3 saadaan voimaan korvaamalla kukin siirtymä $a, s \to s'$ pop-push-yhdistelmällä ja kukin siirtymä $a, \varepsilon \to \varepsilon$ push-pop-yhdistelmällä (yksi uusi tila per korjattava siirtymä).