Fonya ZH recap 2015

szabivános

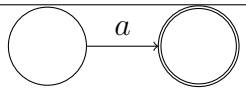
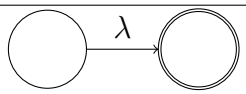
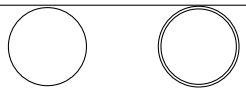
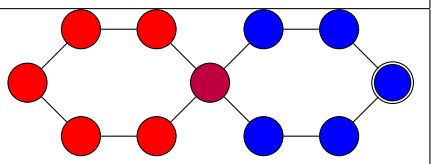
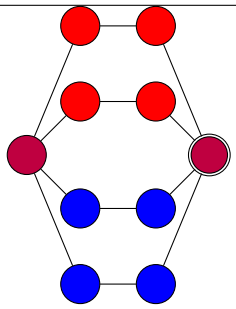
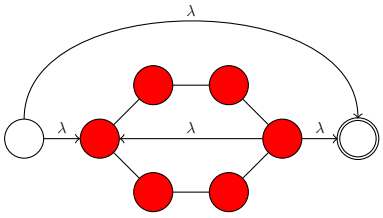
typo lehet, bocs

Regexből DFA-t. Erre direkt algoritmust nem néztünk, olyat tudunk, hogy regexből NFA-t, aztán olyat, hogy NFA-t determinizálni.

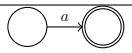
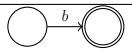
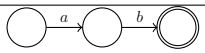
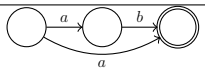
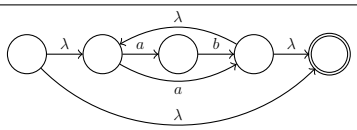
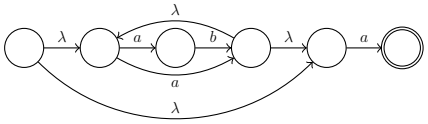
Nézzük ezeket lépésenként.

Thompson algoritmus: input egy regex, output egy λ-átmenetes NFA. A generált automata mindig egy forrás kezdőállapottal és egy nyelő végállapottal fog rendelkezni, ezt a rekurziónál felhasználjuk.

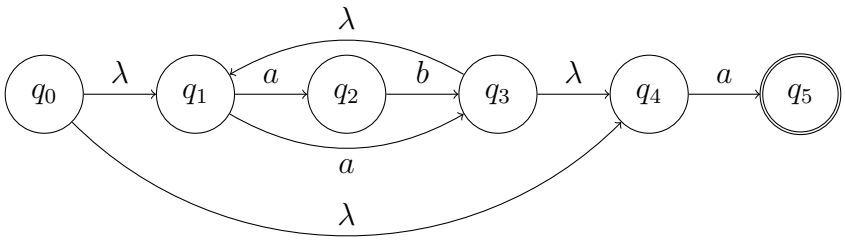
A rekurzió:

Regex	λ-NFA
a	
λ	
$\emptyset$	
$(R_1 R_2)$	
$(R_1 + R_2)$	
$(R)^*$	

Lássuk ezt futni mondjuk az $(a + ab)^* a$ regexen:

Regex	Thompson λ-NFA
a	
b	
ab	
$a + ab$	
$(a + ab)^*$	
$(a + ab)^*a$	

Az egész $(a + ab)^*a$-ra kinagyítom az eredményt és beindexelem az állapotokat, mert ezzel fogunk tovább dolgozni:



Egy X állapot(halmaz) λ-*lezártja* (jelben $\widehat{X}$) az összes olyan állapot halmaza, ahova X-ből nulla, egy vagy több λ-átmenettel el lehet jutni. (Mindig $X \subseteq \widehat{X}$, mert nulla darab λ-lépés megengedett.) Ezt érdemes felírni: $\widehat{q_0} = \{q_0, q_1, q_4\}$, $\widehat{q_1} = \{q_1\}$, $\widehat{q_2} = \{q_2\}$, $\widehat{q_3} = \{q_1, q_3, q_4\}$, $\widehat{q_4} = \{q_4\}$, $\widehat{q_5} = \{q_5\}$.

A Thompson-automatát második lépésben *determinisztikussá* a következőképp tesszük:

- a DFA egy állapota a Thompson-automata állapotainak egy *halmaza* lesz;
- kezdőállapot $\widehat{q_0}$;
- H-ból a hatására $\bigcup_{q \in H} \widehat{q}a$-ba lehet eljutni, azaz vesszük az összes $q \in H$ állapotot, és uniózzuk az összes olyan $\widehat{p}$ halmazt, ahova q-ból a-val p-be lehet jutni;
- végállapot lesz minden olyan halmaz, amiben van eredeti végállapot.

Pl. az első lépésben a $\{q_0, q_1, q_4\}$ kezdőállapotból a hatására el lehet jutni: q_0-ból sehova, q_1-ből q_2-be és q_3-ba, q_4-ből pedig q_5-be, az eredmény tehát a $\widehat{q_2} \cup \widehat{q_3}\widehat{q_5} = \{q_2\} \cup \{q_1, q_3, q_4\} \cup \{q_5\} = \{q_1, q_2, q_3, q_4, q_5\}$ halmaz lesz. Ha viszont ugyanott b-t kapunk, onnan se q_0-ból, se q_1-ből, se q_4-ből nem lehet továbbmenni b-vel, így az $\widehat{\emptyset} = \emptyset$ halmazt kapjuk. (Ami egyébként mindig csapdaállapot.)

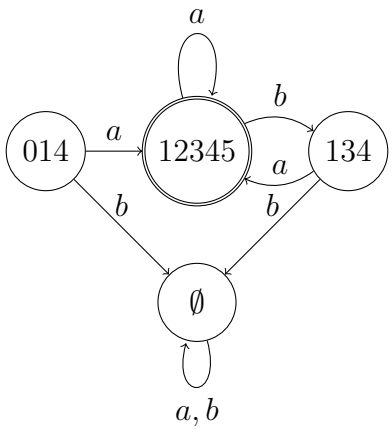
A táblázatot úgy készítjük, hogy először felvesszük sorfejlécnek az új kezdőállapotot, aztán kiszámoljuk a sor celláit, és ahányszor olyan halmazt kapunk, aki nem szerepel még sorfejlécként, felvesszük sorfejlécnek azt is. Előbb-utóbb elfogynak. Célszerű pl. rendezni mindig a halmaz elemeit, hogy az ismétlést könnyebb legyen kiszúrni.

A végeredmény most:

	a	b
$\{q_0, q_1, q_4\}$	$\{q_1, q_2, q_3, q_4, q_5\}$	$\emptyset$
$\{q_1, q_2, q_3, q_4, q_5\}$	$\{q_1, q_2, q_3, q_4, q_5\}$	$\{q_1, q_3, q_4\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3, q_4, q_5\}$	$\emptyset$

és kész is, az egyetlen végállapot a $\{q_1, q_2, q_3, q_4, q_5\}$ (mert az tartalmazza egyedül a q_5-öt).

Aki akarja, le is rajzolhatja ezt az automatát:



De nem muszáj.

Pumpáló lemmás feladatok.

A pumpáló lemma: kapunk egy input L nyelvet és egy *paraméter* K-hoz kell adjunk olyan $w \in L$, $|w| \geq K$ szót, amit bárhogy is vágunk fel $w = w_1w_2w_3$ részre úgy, hogy $|w_1w_2| \leq K$ és $w_2 \neq \lambda$, mindig lesz olyan $i \geq 0$, amire $w_1w_2^iw_3 \notin L$.

Ha csak tehetjük, generáljunk olyan w szót, aminek az első K betűje ugyanaz, mondjuk x, mert akkor csak azt kell megnézni, hogy $w_2 = x^t$, $1 \leq t \leq K$ alakú lehet, akkor $w_1w_2^iw_3$ úgy néz ki, hogy w elé még bejön $(i-1)t$ darab x, és így könnyebb sokszor i-t választani (általában elég az $i = 0$, $i = 2$ valamelyikét megnézni és jó lesz).

Gyakon amit néztünk:

Lássuk be, hogy $L = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$ nem reguláris!

Tegyük fel, hogy L reguláris, legyen K a lemma szerinti konstans.

Legyen $w = a^Kb^K$. Akkor $w \in L$, $|w| = 2K \geq K$.

Ha $w = w_1w_2w_3$ úgy, hogy $|w_1w_2| \leq K$ és $w_2 \neq \lambda$, akkor $w_2 = a^t$ úgy, hogy $1 \leq t \leq K$. Akkor $i = 2$-re $w_1w_2^iw_3 = a^{K+t}b^K \notin L$, ami ellentmondás, tehát L nem reguláris.

Lássuk be, hogy $L = \{a^{n^2} : n \geq 0\}$ nem reguláris!

Tegyük fel, hogy L reguláris, legyen K a lemma szerinti konstans.

Legyen $w = a^{K^2}$. Akkor $w \in L$, $|w| = K^2 \geq K$.

Ha $w = w_1w_2w_3$ úgy, hogy $|w_1w_2| \leq K$ és $w_2 \neq \lambda$, akkor $w_2 = a^t$ úgy, hogy $1 \leq t \leq K$. Akkor $i = 2$-re $w_1w_2^iw_3 = a^{K^2+t}$, ami nem L-beli, mert $K^2 < K^2 + t \leq K^2 + K < K^2 + 2K + 1 = (K+1)^2$, így két szomszédos négyzetszám közé esik. Emiatt L nem reguláris.

Lássuk be, hogy $L = \{w \in \{(,)\}^* : w$ helyes zárójelezés$\}$ nem reguláris!

Tegyük fel, hogy L reguláris, legyen K a lemma szerinti konstans.

Legyen $w = ({}^K)^K$. Akkor $w \in L$, $|w| = 2K \geq K$.

Ha $w = w_1w_2w_3$ úgy, hogy $|w_1w_2| \leq K$ és $w_2 \neq \lambda$, akkor $w_2 = ({}^t$ úgy, hogy $1 \leq t \leq K$. Akkor $i = 2$-re $w_1w_2^iw_3 = ({}^{K+t})^K \notin L$. Emiatt L nem reguláris.

Lássuk be, hogy $L = \{a^p : p$ prímszám$\}$ nem reguláris!

Tegyük fel, hogy L reguláris, legyen K a lemma szerinti konstans.

Legyen $w = a^P$, ahol $P > K$ prímszám (ilyen van, mert végtelen sok prímszám van). Akkor $w \in L$, $|w| = P \geq K$.

Ha $w = w_1w_2w_3$ úgy, hogy $|w_1w_2| \leq K$ és $w_2 \neq \lambda$, akkor $w_2 = a^t$ úgy, hogy $1 \leq t \leq K$. Akkor $i = P + 1$-re $w_1w_2^iw_3 = a^{P+Pt} \notin L$, mert $P + Pt = P(t + 1)$ összetett szám. Emiatt L nem reguláris.

A többi. Ezeket meg lehet így oldani:

- $\{a^nb^{2n} : n \geq 0\}$
- $\{w \in \{a, b\}^* : 2 \cdot |w|_a = |w|_b\}$
- $\{w \in \{(,), [,]\}^* : w$ helyes zárójelezés$\}$
- $\{a^{n^3} : n \geq 0\}$
- $\{a^{2^n} : n \geq 0\}$
- $\{w \in \{a, b\}^* : |w|_a \leq |w|_b\}$
- $\{w \in \{a, b\}^* : |w|_b < |w|_a\}$
- $\{w \in \{a, b\}^* : w$ palindrom$\}$

Nyelvhez CF nyelvtan készítése tippek.

Amit tudunk: $\{a^{t_1n+c_1}b^{t_2n+c_2} : n \geq 0\}$-hoz egy nyelvtan $S \to a^{t_1}Sb^{t_2} \mid a^{c_1}b^{c_2}$. Pl. $\{a^nb^n : n \geq 0\}$-hoz $S \to aSb \mid \lambda$, $\{a^{2n+1}b^n : n \geq 0\}$-hoz $S \to aaSb \mid a$, stb. A másik, amit tudunk, a palindromák: $S \to aSa \mid bSb \mid a \mid b \mid \lambda$.

Ha a nyelvünk szavait két *független* részre tudjuk vágni, akkor tegyük meg, csináljuk meg hozzájuk külön-külön a nyelvtanokat, mondjuk A és B nemterminálisból indítva, aztán vegyünk fel egy $S \to AB$ szabályt. Pl: $\{a^nb^m : n < m\}$-nél ez ugyanaz, mint $\{a^nb^nb^d : n \geq 0, d > 0\}$ (és akkor $n < n + d = m$), itt az a^nb^n és a b^d részek közt semmi kapcsolat nincs, elsőhöz nyelvtan $A \to aAb \mid \lambda$, másodikhoz $B \to bB \mid b$, az egész meg

$$\begin{aligned} S &\to AB \\ A &\to aAb \mid \lambda \\ B &\to bB \mid b. \end{aligned}$$

Egy másik nyelvtan ugyanerre: $S \to aSb \mid Sb \mid b$.

Ha a szavak ,,közepéből" tudunk kivágni független dolgot, akkor azt vezessük le mondjuk A-ból, és az eredeti nyelvtanban a kivágott rész helyére ezt az A-t vezessük le. Pl. $\{a^nb^mc^md^n\}$-nél a

belső $b^m c^m$ rész kiemelhető, $A \to bAc \mid \lambda$, és ekkor az eredeti S-ből $a^n A d^n$ alakú stringeket kell levezessünk, amit tudunk:

$$S \to aSd \mid A$$
$$A \to bAc \mid \lambda.$$

Ha a nyelvünket két nyelv uniójára tudjuk bontani, akkor csináljuk meg hozzájuk külön-külön a nyelvtanokat, mondjuk A és B nemterminálisból indítva, aztán vegyünk fel egy $S \to A \mid B$ szabályt. Pl: $\{a^n b^m : n \neq m\}$-nél az $n \neq m$ az $n < m$ vagy $m < n$ és akkor

$$S \to A \mid B$$
$$A \to aAb \mid Ab \mid b$$
$$B \to aBb \mid aB \mid a.$$

Chomsky-normálformára hozás.

Chomsky alak: minden szabály $A \to BC$ vagy $A \to a$ alakú. Kivéve az $S \to \lambda$ szabály, ez lehet, de ha van, akkor nem lehet S egy szabály jobb oldalán se. Minden CF nyelvtant ilyen alakra lehet hozni.

Algoritmus:

i) *Fake nemterminálisok bevezetése*: minden $a \in \Sigma$-hoz bevezetünk egy új X_a nemterminálist, és egy $X_a \to a$ szabályt; az összes olyan jobb oldalon, ahol a nem egyedül áll, kicseréljük X_a-ra.

ii) *Hosszú jobboldalak tördelése*: a kettőnél hosszabb jobboldalak suffixei helyett új nemterminálisokat vezetünk be, kettő hosszú jobboldalakat létrehozva. Példán keresztül: $A \to X_1X_2X_3X_4$-ből lesz $A \to X_1[X_2X_3X_4]$, $[X_2X_3X_4] \to X_2[X_3X_4]$, $[X_3X_4] \to X_3X_4$. Ha n hosszú a jobboldal: $A \to X_1X_2 \dots X_n$-ből $A \to X_1[X_2 \dots X_n]$, $[X_i \dots X_n] \to X_i[X_{i+1} \dots X_n]$, $[X_{n-1}X_n] \to X_{n-1}X_n$.

iii) *λ-mentesítés*: először meghatározzuk az összes olyan nemterminális $\mathcal{X}$ halmazát, akikből lehet λ-t levezetni. Inicializálás: $\mathcal{X} := \{A : \text{ van } A \to \lambda \text{ szabály}\}$. Iteráció:

$$\mathcal{X} := \mathcal{X} \cup \{A : \text{ van } A \to \alpha \text{ szabály valamilyen } \alpha \in \mathcal{X}^*\text{-ra}\}.$$

(Ha valakinek van csupa nullázható jelből álló jobboldala, akkor őt is bevesszük.) Ha $\mathcal{X}$ már nem változik, akkor kész vagyunk.

Ezek után ,,töröljük az összes $\mathcal{X}$-beli jelet, ahányféleképp csak lehet, de λ jobboldalt nem írunk fel", vagyis:

- az $A \to BC$ szabályok mellé felvesszük $A \to B$-t, ha C nullázható és $A \to C$-t, ha B nullázható;
- az $A \to B$ és $A \to a$ szabályok nem változnak;
- az $A \to \lambda$ szabályokat töröljük.

Ezen kívül, ha S nullázható, akkor felveszünk egy új S' kezdőszimbólumot és $S' \to S|\lambda$ szabályokat.

iv) *Láncszabály-mentesítés*: Először felírunk egy G gráfot. Csúcsai: a nemterminálisok. Élek: $A \to B$ él, ha van $A \to B$ szabály. Ezek után minden A nemterminális új jobb oldala az összes, A-ból G-ben elérhető nemterminális jobb oldala lesz, kivéve a láncszabályokat.

kész, a kapott nyelvtan ekivalens, Chomsky alakú.

Példa. Input:

$$\begin{aligned} S &\to ASAa \mid ACB \mid bb \\ A &\to aB \mid \lambda \mid BSa \\ B &\to AA \mid a \end{aligned}$$

Akkor i) után:

$$\begin{aligned} S &\to ASAX_a \mid ACB \mid X_bX_b \\ A &\to X_aB \mid \lambda \mid BSX_a \\ B &\to AA \mid a \\ X_a &\to a \\ X_b &\to b \end{aligned}$$

Tördelés:

$$\begin{aligned} S &\to A[SAX_a] \mid A[CB] \mid X_bX_b \\ [SAX_a] &\to S[AX_a] \\ [AX_a] &\to AX_a \\ [CB] &\to CB \\ A &\to X_aB \mid \lambda \mid B[SX_a] \\ [SX_a] &\to SX_a \\ B &\to AA \mid a \\ X_a &\to a \\ X_b &\to b \end{aligned}$$

Nullázható: A ($A \to \lambda$ miatt), B ($B \to AA$ miatt) és ha még lenne $\{A, B\}^*$ jobboldalú szimbólum, az is azzá válna, de nincs. Ezeket elhagyjuk:

$$\begin{aligned} S &\to A[SAX_a] \mid [SAX_a] \mid A[CB] \mid [CB] \mid X_bX_b \\ [SAX_a] &\to S[AX_a] \\ [AX_a] &\to AX_a \mid X_a \\ [CB] &\to CB \mid C \\ A &\to X_aB \mid X_a \mid B[SX_a] \mid [SX_a] \\ [SX_a] &\to SX_a \\ B &\to AA \mid A \mid a \\ X_a &\to a \\ X_b &\to b \end{aligned}$$

Láncszabályok: az éleket sorolva $S \to [SAX_a], [CB]$, $[AX_a] \to X_a$, $[CB] \to C$, $A \to X_a, [SX_a]$, $B \to A$. Akkor S-ből elérhető $S, [SAX_a], [CB], C$, $[AX_a]$-ból $[AX_a], X_a$, $[CB]$-ből $[CB], C$, A-ból $A, X_a, [SX_a]$, B-ből pedig $B, A, X_a, [SX_a]$. A többi nemterminálisból csak önmaguk.

Mindenki mellé másoljuk a belőle elérhetőek nem-láncszabály jobboldalait:

$$\begin{aligned}
S &\to A[SAX_a] \mid A[CB] \mid X_bX_b \mid S[AX_a] \mid CB \\
[SAX_a] &\to S[AX_a] \\
[AX_a] &\to AX_a \mid a \\
[CB] &\to CB \\
A &\to X_aB \mid B[SX_a] \mid a \mid SX_a \\
[SX_a] &\to SX_a \\
B &\to AA \mid a \mid X_aB \mid B[SX_a] \mid a \mid SX_a \\
X_a &\to a \\
X_b &\to b
\end{aligned}$$

és kész is. (Aki szemfüles, esetleg észreveheti, hogy C nem terminál, tehát törölhettük volna az elején akár.)

Veremautomata építés

Alapvetően a veremautomata egy *nemdeterminisztikus* számológép, ellátva egy *veremmel*, aminek a szokásos `pop`/`push` műveleteit tudja alkalmazni. A vermet két dologra lehet használni igazán:

- számlálónak (real-life)
- beletenni egy szót és annak a megfordítottját ellenőrizni (somewhat less real-life)

Van ahol a w^{-1} és van, ahol a w^R jelzi az ,,írd le a w szót megfordítva" műveletet, itt most w^R lesz, tehát pl. $bacba^R = abcab$.

Kezdem a számláló managementtel. Ha veremautomatát akarunk tervezni, akkor pl. kezdhetünk egy olyan algoritmussal (nem rögtön veremautomatával), ami

- balról jobbra olvassa a stringet
- egy egészértékű számlálót használ, más változó nincs, a számláló nullára van inicializálva
- a számláló lehet pozitív vagy negatív is
- a számlálót lehet konstanssal növelni, csökkenteni vagy nullázni
- hogy mit csináljon az algoritmus, az függhet az olvasott betűtől és attól, hogy a számláló értéke nulla, pozitív vagy negatív
- meg persze az aktuális programsortól, ahova lehet gotozni is, akár nemdeterminisztikusan ,,egyszer csak" átugorva
- és az algoritmus kimenete persze egy bit lesz, hogy elfogadjuk-e a szót (,,jó alakú-e") vagy sem, elfogadni csak úgy szabad, ha végig is olvastuk

Nézzünk egy példát: hogy az input stringünk a^nb^n alakú-e, arra egy számlálós algoritmus (a számláló `c`):

1. Ha `a`-t olvasunk, `c:=c+1` és goto 1; nemdet átmehetünk 2-re is
2. Ha `b`-t olvasunk, `c:=c-1` és goto 2; nemdet átmehetünk 3-ra is

3. Ha elfogyott az input és `c==0`, elfogadjuk a stringet.

Pl. ez az algoritmus az *aabb* inputra először növeli a számlálót eggyel, majd még eggyel, ahogy olvassa be az `a`-kat, aztán az utolsó `a` után egyszer csak „tudatállapotot vált" (átmegy a 2. pontra) és onnan kezdve ahogy jönnek a `b`-k, csökkenti a számlálót, és mikor elfogynak a `b`-k is, átugrik a 3. pontra, ott ellenőrzi, 0-e a számláló és mivel annyi, elfogadja a stringet. *aaabb*-re pl. 1 lenne a számláló a végén, azt nem fogadná el, az *aba*-t meg végig se tudja olvasni (mert a 2. pontban tud csak `b`-t olvasni, ahonnan nem tud visszajutni az 1-be, ahol tudná az azt követő `a`-t).

Ha a sorrend nem számít és az $\{w \in \{a,b\}^* : |w|_a = |w|_b\}$ nyelvre írunk algoritmust:

1. Ha `a`-t olvasunk, növeljük `c`-t és goto 1; ha `b`-t, csökkentsük `c`-t és goto 1; nemdet mehetünk 2-re is
2. Ha elfogyott az input és `c==0`, elfogadjuk a stringet.

Ha nem egyenlőséget tesztelünk és az $\{w \in \{a,b\}^* : |w|_a < |w|_b\}$ nyelvre írunk algoritmust, akkor az utolsó lépésben `c<0`-ra kell teszteljünk.

Ha pl. helyes zárójelezéseket akarunk felismerni, pl. (()), ()(), (())() jó,)(nem jó:

1. Ha `(`-t olvasunk, növeljük `c`-t és goto 1; ha `)`-t és `c>0`, akkor csökkentsük `c`-t és goto 1; nemdet mehetünk 2-re is
2. Ha elfogyott az input és `c==0`, elfogadjuk a stringet.

(Ez mondjuk az üres stringet is elfogadja; ha ezt nem akarjuk, akkor pl. tehetjük a goto 2 részt `)` kezelés utánra is, a helyes zárójelezések mindenképp csukójelre végződnek.)

Ha pl. lineáris összehasonlítást akarunk csinálni, pl. $a^n b^{2n+1}$, akkor nem eggyel változtatgatjuk a számlálónkat:

1. Ha `a`-t olvasunk, akkor `c:=c+2` és goto 1; nemdet mehetünk 2-re is és akkor `c:=c+1`
2. Ha `b`-t olvasunk, akkor `c:=c-1` és goto 2; nemdet mehetünk 3-ra is
3. Ha elfogyott az input és `c==0`, elfogadjuk a stringet.

Pl. az *aabbbbb* olvasásakor az első *a* után 2, a második után 4 lesz a számláló értéke, aztán gotozunk 2-re, ekkor 5, és utána minden egyes *b*-re csökken a számláló, a végén épp 0 lesz és elfogadjuk.

Persze lehet ezt még cizellálni, pl. $\{a^i b^j c^k : i + k = j\}$-nél az *a*-k és a *c*-k növelik eggyel a számlálót, a *b*-k meg csökkentik és a végén 0-ra tesztelünk, stb.

Hogy lesz ebből veremautomata? A veremben tartjuk a számlálót, háromféle jelet használunk:

- a kezdetben is benne levő Z-t;
- a pozitív értéket jelző $+$-t;
- a negatív értéket jelző $-$-t

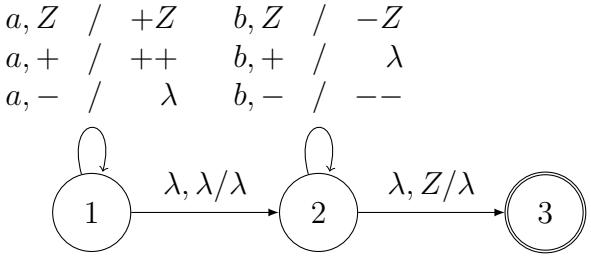
mégpedig úgy, hogy a verem tartalma Z, ha a számláló értéke 0, $+Z$, ha 1, $++Z$, ha 2 stb, általában $+^nZ$ jelzi az $n > 0$ értéket és $-^nZ$ a $-n$ értéket. Nem keverjük a $+/-$ jeleket, mindig Z van a verem alján.

Ekkor:

- Egy programsorból lesz (kb) egy állapot;
- A számláló növelése eggyel:
  - ha a verem tetején 0 vagy + van, akkor tegyünk a helyére +0-t vagy ++-t
  - ha pedig −, akkor azt csak vegyük le.
- A számláló csökkentése eggyel:
  - ha a verem tetején 0 vagy − van, akkor tegyünk a helyére −0-t vagy −−-t
  - ha pedig +, akkor azt csak vegyük le.
- A számláló csökkentése konstanssal: tegyünk annyi eggyel csökkentő állapotot be, amennyivel csökkenteni akarunk. Növelésre ugyanez.
- A számláló tesztelése: ha Z van a verem tetején, akkor 0, ha +, pozitív, ha −, negatív.
- Az ,,elfogadjuk a stringet" rész egy végállapotba ugrás lesz, a nemdet gotozás pedig nem olvas be jelet az inputról.

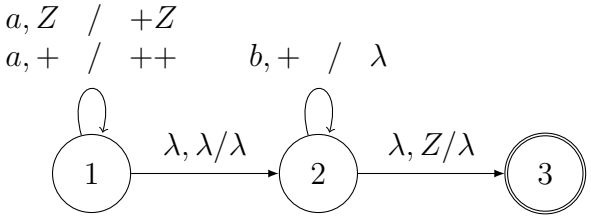
Nézzük meg az előző algoritmusokat veremautomatára ,,implementálva":

- a^nb^n:



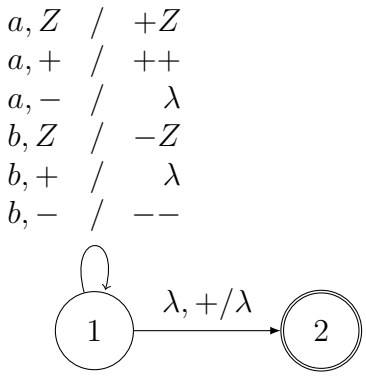
  Tehát: az első állapot amíg a-t olvasunk, növeli a számlálót, majd ugrik a másodikba, ami amíg b-t olvasunk csökkenti a számlálót, aztán ugrunk a 3-ra és elfogadunk, feltéve, hogy a számláló értéke épp 0.

  Ennél a feladatnál persze észre lehet venni, hogy a számláló értéke ha mínuszba csúszik, akkor úgyse fogjuk elfogadni az inputot, és ekkor a negatív számokat kezelnünk se kell, ha elpusztul menet közben az algoritmus nem kezelt if miatt, akkor is elutasítja az inputot, olyankor nem baj, ha nem olvassuk végig:



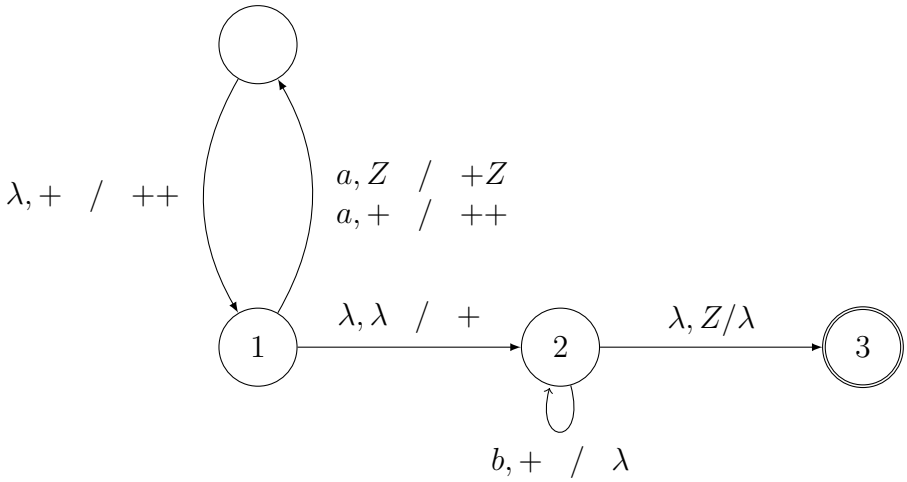
és ez is ugyanazt a nyelvet ismeri fel.

- $\{w \in \{a,b\}^* : |w|_a > |w|_b\}$:



Itt ugye az első sor manageli a számlálót, a-ra növel, b-re csökkent, kell hogy legyen negatív is managelve, és a végén ha pozitív lett a számláló, akkor fogad el.

- $\{a^n b^{2n+1} : n \geq 0\}$:



Itt pedig a bal oldali két állapot együttesen manageli a kettővel növelést (btw mivel itt a számláló csak pozitív lehet, egy + helyett + + + és egy Z helyett + + Z betétele is jó lenne, de általánosan, ha nem tudod, hogy a számlálód előjele mi lehet egy adott ponton, ezt külön kell managelni). Arra kell odafigyelni, hogy ilyenkor csak az első nyíl (ami kifele megy az 1-ből) olvasson be betűt az inputról, a többieknek az inputból λ-t olvasunk, így egy a-ra több inc fut le majd. Eztán a 2-be ugráskor még teszünk egy +-t a verembe, utána minden b-re csökkentünk és ha a végén 0 lesz, akkor jó.

Annyit tennék még hozzá, hogy számtudból ez OK, fonyából az $a, \lambda/X$ alakú átmenetek helyett $a, A/XA$, $a, B/XB$, ... kell megadnunk, mert számtudból megengedett az, hogy a verem tetejét meg se nézzük (ezt jelzi ugye a vessző utáni λ), fonyából meg mindenképp popnunk kell egy jelet. Tehát amit a JFLAP bevesz (λ a verem tetején), azt fonyából hogy valid legyen, végig kell iterálnunk az összes lehetséges veremjelen, és mindre ugyanúgy switchelni.

Hát kb így. lehet még egymásba ágyazni számlálókat (pl. az $a^n b^m c^m d^n$ nyelvnél), de ott már azt csinálnám, hogy CF nyelvtant gyártok rá és azt konvertálom automatára, ahogy a stringbabráló algoritmusokból is (ww^R meg ezek).