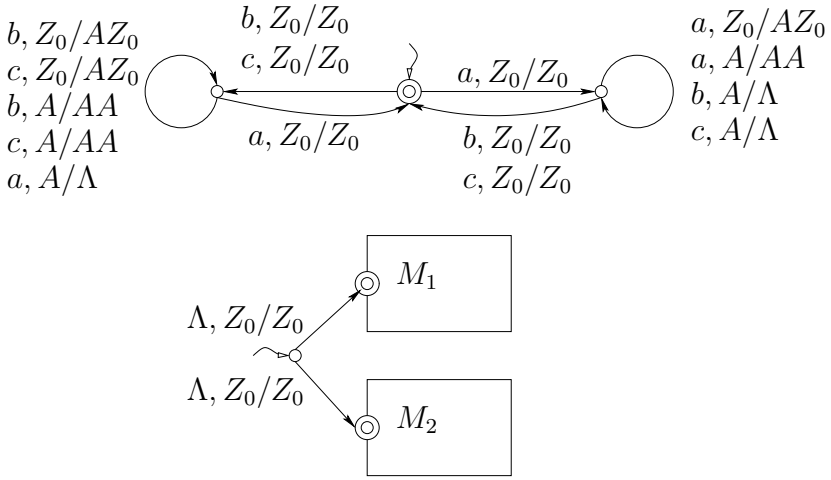
ANTWOORDEN tentamen FUNDAMENTELE INFORMATICA 3
vrijdag 9 januari 2009, 10.00 - 13.00 uur

Opgave 1 [25 pt]
$L_1 = \{w \in \{a,b,c\}^* \mid n_a(w) = n_b(w) + n_c(w)\}$,
$L_2 = \{w \in \{a,b,c\}^* \mid n_a(w) + n_b(w) = n_c(w)\}$ en
$L_3 = L_1 \cup L_2$.



Een DPDA voor M_1 en (schematisch) een PDA voor M_3.

a. De bovenste automaat in de figuur is een DPDA voor L_1. De begintoestand (tevens eindtoestand) geeft aan dat het aantal gelezen a's even groot is als de aantallen gelezen b's en c's samen. In de linkertoestand liggen de b's en c's voor en in de rechter juist de a's.
Voor M_2 kan je een DPDA geven die er net zo uit ziet alleen zijn de a's en de c's verwisseld (merk op dat we maar één extra stapelsymbool naast Z_0 gebruiken!)

b. L_1 noch L_2 kunnen door een DPDA worden geaccepteerd met lege stapel. Dit is een gevolg van het feit dat geen van beide talen prefix-vrij is (er zijn x en $y \neq \Lambda$ zodat zowel x als xy woorden uit de betreffende taal zijn, bijvoorbeeld ac en $acac$). Een deterministische automaat heeft per woord hooguit één berekening. Als na uitlezen daarvan de stapel leeg is kan er dus geen verlenging van dat woord meer worden geaccepteerd. (Zie opgave 7.15.)

N.B.1: een stapel met alleen nog een bodemsymbool als Z_0 is niet leeg!
N.B.2: de constructie in het bewijs van Stelling 7.3 (die van een gewone PDA een maakt die dezelfde taal met lege stapel accepteert) vernietigt determinisme!

c. De onderste automaat in de figuur is een PDA die M_3 accepteert (door niet deterministisch te proberen (een gok) of het woord door M_1 dan wel M_2 geaccepteerd kan worden (de twee Λ-transities vanuit de nieuwe begintoestand gaan naar de oorspronkelijke begintoestanden van M_1 en M_2). Aangezien $\Lambda \in L_3$ zou de nieuwe begintoestand ook nog toegevoegd kunnen worden aan de verzameling eindtoestanden.

d. De toestanden van M_3 uit het vorige onderdeel krijgen elk een naam: q_0 voor de begintoestand; in M_1 van links naar rechts q_{L1}, q_{01} en q_{R1} en in M_2

van links naar rechts q_{L2}, q_{02} en q_{R2}.
Accepterende berekeningen van M_3 (één accepterende berekening per woord geven is genoeg)
voor Λ: $(q_0, \Lambda, Z_0) \vdash (q_{01}, \Lambda, Z_0)$ en $(q_0, \Lambda, Z_0) \vdash (q_{02}, \Lambda, Z_0)$;
voor $aacc$: $(q_0, aacc, Z_0) \vdash (q_{01}, aacc, Z_0) \vdash (q_{R1}, acc, Z_0) \vdash (q_{R1}, cc, AZ_0) \vdash (q_{R1}, c, Z_0) \vdash (q_{01}, \Lambda, Z_0)$
en $(q_0, aacc, Z_0) \vdash (q_{02}, aacc, Z_0) \vdash (q_{R2}, acc, Z_0) \vdash (q_{R2}, cc, AZ_0) \vdash (q_{R2}, c, Z_0) \vdash (q_{02}, \Lambda, Z_0)$;
voor $aabc$: $(q_0, aabc, Z_0) \vdash (q_{01}, aabc, Z_0) \vdash (q_{R1}, abc, Z_0) \vdash (q_{R1}, bc, AZ_0) \vdash (q_{R1}, c, Z_0) \vdash (q_{01}, \Lambda, Z_0)$ dit is de enige accepterende berekening van M_3 voor $aabc$;
voor $aabccc$: $(q_0, aabccc, Z_0) \vdash (q_{02}, aabccc, Z_0) \vdash (q_{L2}, abccc, Z_0) \vdash (q_{L2}, bccc, AZ_0) \vdash (q_{L2}, ccc, AAZ_0) \vdash (q_{L2}, cc, AZ_0) \vdash (q_{L2}, c, Z_0) \vdash (q_{02}, \Lambda, Z_0)$ dit is de enige accepterende berekening van M_3 voor $aabccc$.

e. L_3 is geen deterministische context-vrije taal.
We bewijzen deze bewering d.m.v. een tegenspraak.
Neem dus aan dat L_3 wel een DCFL is.
Dat betekent dat er een DPDA is die L_3 accepteert. Volgens I is $L_3^{\#}$ dan ook een DCFL. Merk op dat $\{a\}^*\{b\}^*\{\#\}\{c\}^*$ een reguliere taal is (rechttoe rechtaan FA) en dan volgt met II dat $L_3^{\#} \cap \{a\}^*\{b\}^*\{\#\}\{c\}^*$ een DCFL is, wat in tegenspraak is met III, die beweert dat deze taal niet eens context-vrij is.
Onze aanname dat L_3 een DCFL is, blijkt niet juist.

f. $L_4 = L_1 \cap L_2$ een DCFL, want $L_1 \cap L_2 = \{w \in \{a,b,c\}^* \mid n_a(w) = n_c(w) \text{ en } n_b(w) = 0\}$ en deze taal wordt geaccepteerd door een DPDA zoals bijvoorbeeld M_1 waarin alle b-transities zijn weggelaten.

Opgave 2 [18 pt]
CFG G heeft startsymbool S, terminaal alfabet $\{d, e, f\}$ en producties
$S \to Tf \quad T \to TXd \mid TYe \mid \Lambda \quad X \to eX \mid eY \quad Y \to Yd \mid \Lambda.$

a. Eliminatie van de links-recursie in G (met nieuwe niet-terminalen U en V):
$T \to TXd \mid TYe \mid \Lambda$ wordt vervangen door $T \to U$ en $U \to XdU \mid YeU \mid \Lambda$.
$Y \to Yd \mid \Lambda$ wordt vervangen door $Y \to V$ en $V \to dV \mid \Lambda$.
(De ketenproducties $T \to U$ en $Y \to V$ mogen ook verwijderd worden.)

b. Factorisatie toegepast op $X \to eX$ en $X \to eY$ (nieuwe niet-terminaal W):
$X \to eW$ en $W \to X \mid Y$.
(Ook de ketenproducties $W \to X \mid Y$ zouden mogen worden verwijderd.)

De grammatica H geconstrueerd bij de vorige onderdelen heeft producties
$S \to Tf$
$T \to U \quad U \to XdU \mid YeU \mid \Lambda$
$X \to eW \quad W \to X \mid Y$
$Y \to V \quad V \to dV \mid \Lambda$.

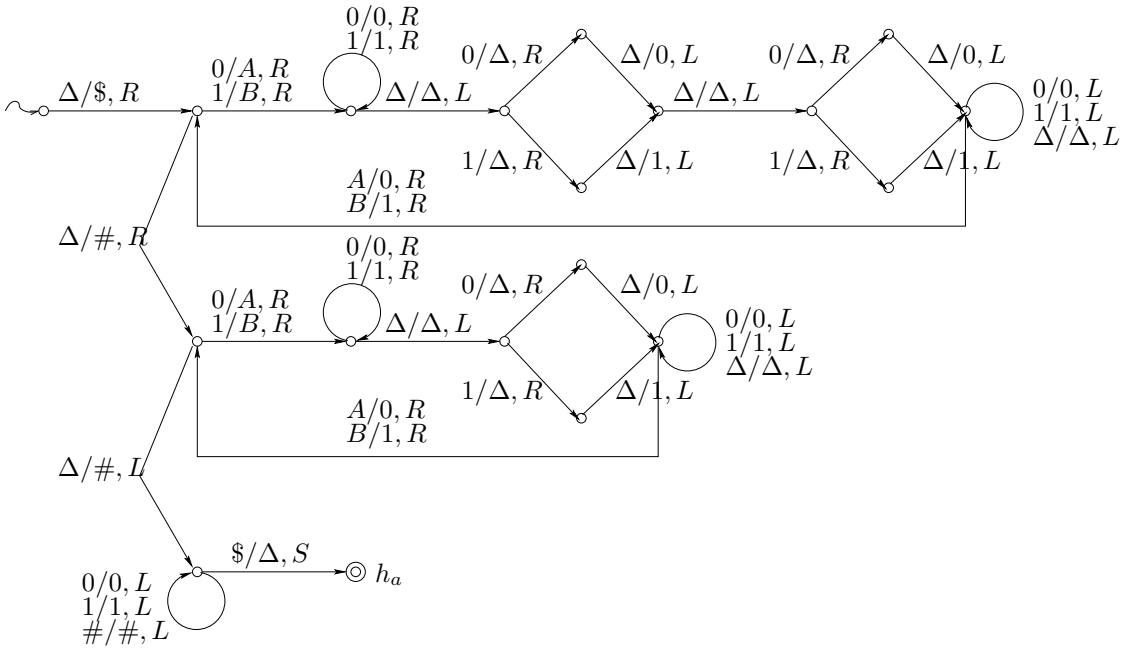
c. Lookahead voor de producties van H:
$\texttt{LA}_1(S \to Tf) = \{d, e, f\}$ — elk van deze 3 symbolen kan de eerste letter van een gegenereerd terminaal woord zijn;
$\texttt{LA}_1(T \to U) = \{d, e, f\}$ — alle lookaheadsymbolen van de producties voor U;

$\texttt{LA}_1(U \to XdU) = \{e\}$ — lookahead van de enige productie voor X;
$\texttt{LA}_1(U \to YeU) = \{d, e\}$ — lookahead van de enige productie voor Y;
$\texttt{LA}_1(U \to \Lambda) = \{f\}$ — achter U staat altijd een f;
$\texttt{LA}_1(X \to eW) = \{e\}$;
$\texttt{LA}_1(W \to X) = \{e\}$ — lookahead van de enige productie voor X;
$\texttt{LA}_1(W \to Y) = \{d, e\}$ — lookahead van de enige productie voor Y;
$\texttt{LA}_1(Y \to V) = \{d, e\}$ — alle lookaheadsymbolen van de producties voor V;
$\texttt{LA}_1(V \to dV) = \{d\}$;
$\texttt{LA}_1(V \to \Lambda) = \{e\}$ — achter (Y en) V staat altijd een e.

d. Zie p.285 van het boek:
Een context-vrije grammatica is LL(1) als het voldoende is om één symbool vooruit te kijken in een te parsen invoerwoord om te bepalen wat de parser moet doen (welke productie voor de meest linkse niet-terminaal moet worden gebruikt). Formeel: voor alle niet-terminalen A geldt, als $A \to \alpha$ en $A \to \beta$ producties zijn met $\alpha \neq \beta$, dan $\texttt{LA}_1(A \to \alpha) \cap \texttt{LA}_1(A \to \beta) = \emptyset$.
De LL(1) eigenschap maakt het mogelijk een bijbehorende deterministische pushdown parser te construeren.

e. H voldoet niet aan de LL(1) eigenschap. Immers, zoals we bij **c** zien, zijn er niet-terminalen met producties die gemeenschappelijke lookahead-symbolen hebben, namelijk U met $U \to XdU$ en $U \to YeU$; en W met $W \to X$ en $W \to Y$.

Opgave 3 [20 pt]



De Turingmachine T met invoeralfabet $\{0, 1\}$ die hierboven is gegeven, verdeelt gegeven invoer w in drie stukken van gelijke lengte gescheiden door #'s.

Gevolgde methode (zie opgave 9.6(f)):
In de eerste fase worden voor elk symbool aan het begin twee symbolen aan het eind een plaats opgeschoven (links van deze twee staat dan dus een Δ). Dit wordt herhaald tot eerste derde deel en laatste twee-derde deel elkaar ontmoeten (op zoek naar het volgende symbool in het eerste derde deel vindt de TM een blanko). Op deze plaats schrijft de TM zijn eerste #.
Merk op: als de lengte van het woord niet door 3 deelbaar is, crasht de machine in deze fase.
In de tweede fase wordt volgens het zelfde principe (maar nu wordt aan het eind steeds een symbool opgeschoven) de cel halverwege het tweede deel gezocht en daar wordt het tweede # geschreven.
Merk op: als het invoerwoord leeg is heeft de TM nu op de band staan: \$##.
De kop loopt terug naar cel 0 (met de \$) schrijft een Δ en stopt met de kop nog op cel 0 (voorgeschreven uitvoerformat, zie Definitie 9.3).

Een andere methode zou gebaseerd kunnen zijn op het volgende idee: Zet 3 hekjes aan begin (of eind) van de invoer en laat ze naar de andere kant lopen in stappen waarbinnen het voorste hekje steeds 3 cellen opschuift, het middelste twee en het derde steeds een cel.

Opgave 4 [20 pt]
Bij a en b is het voldoende om een toestandsdiagram met wat toelichting te geven.
a. T aanpassen zodat de resulterende TM $\{a, b\}^* - L$ accepteert.
Merk eerst op dat er geen invoerwoorden zijn waarop T oneindig lang gaat rekenen. Als een woord niet de vereiste vorm heeft, dan crasht T omdat hij het gewenste symbool niet aantreft.
Maak T volledig door zijn toestandsovergangsfunctie totaal te maken voor alle toegestane invoer (maak reject expliciet door h_r toe te voegen). Drie nieuwe transitios zijn voldoende.
Er geldt nu voor elk invoerwoord dat het ofwel element van L is en leidt tot h_a of geen element van L in welk geval het leidt tot h_r.
Wissel nu h_r en h_a om.

b. Maak van T een Turingmachine die L beslist:
We gaan uit van de versie van T die we net bij a hebben gemaakt voordat we h_a en h_r omwisselden.
Voeg instructies toe zodat je vanuit h_a — nu beschouwd als een gewone toestand — helemaal (over de lege band) naar links loopt tot cel 0, die herkenbaar is aan \$; veeg \$ uit, stap naar rechts print 1 ('yes'), ga terug naar cel 0 en stop in (de nieuw toegevoegde) h_a. Vanuit h_r — nu ook een gewone toestand — loop je naar rechts tot de meest rechtse beschreven cel (eenvoudig te herkennen), dan naar links tot \$ onderwijl de hele band uitvegend. Maak van \$ ook een Δ, stap naar rechts print 0 ('no') ga terug naar cel 0 en stop in (de net nieuw toegevoegde) h_a.
(Zie ook Stelling 10.2: elke TM die altijd stopt, definieert een recursieve taal.

c. 1. Het complement van een recursieve taal is recursief:
Is waar, als een taal recursief is, dan is er een TM die die taal beslist; wissel de antwoorden 'yes' en 'no' (1 en 0) om. De resulterende TM beslist het

complement van L.

2. Het complement van een recursief opsombare taal is recursief opsombaar.
Is niet waar, een tegenvoorbeeld is de taal SA die wel recursief opsombaar is, maar niet recursief (Stelling 11.2). Als het complement van SA recursief opsombaar zou zijn, dan was SA recursief (Stelling 10.5).

3. Er zijn aftelbaar veel recursief opsombare talen over $\{a, b\}$.
Is waar, zie voorbeeld 10.8: dankzij een codering als e correspondeert elke re taal via een gecodeerde Turingmachine met een (of meer) woorden over een binair alfabet als $\{a, b\}$.

4. Elke taal over $\{a, b\}$ is recursief opsombaar.
Nee, NSA is een taal (over een binair alfabet) die niet re is (Stelling 11.1). Een ander argument zou zijn dat er aftelbaar veel re talen zijn over $\{a, b\}$, terwijl er overaftelbaar veel talen zijn over $\{a, b\}$.

Opgave 5 [17 pt]

a. $P_1 \leq P_2$ (P_1 reduceert naar P_2) — Definitie 11.3:
als er een algorithmische procedure (effectieve methode) is om elke instantie I van P_1 te vertalen in een instantie $F(I)$ van P_2 op zo'n manier dat de antwoorden voor I en $F(I)$ overeen komen (dus I is een ja-instantie van P_1 dan en slechts dan als $F(I)$ een ja-instantie van P_2 is).

b. Als P_1 onbeslisbaar is en $P_1 \leq P_2$, dan is P_2 onbeslisbaar — Stelling 11.4:
Neem aan dat P_1 onbeslisbaar is en dat $P_1 \leq P_2$. Stel nu dat P_2 wel beslisbaar zou zijn. Dan hebben we de volgende methode om P_1 te beslissen: gegeven een willekeurige instantie I van P_1; transformeer I tot $F(I)$ (effectief!) en pas het algoritme dat blijkbaar voor P_2 bestaat toe. Het antwoord dat we zo krijgen voor $F(I)$ is ook het antwoord voor I. Dus P_1 zou beslisbaar zijn, wat in tegenspraak is met het gegeven. Dus ook P_2 moet wel onbeslisbaar zijn.

c. Twee beslissingsproblemen AcceptsEverything en NoLoops.

AcceptsEverything:
Gegeven Turingmachine T; accepteert T alle woorden over zijn invoeralfabet?

NoLoops:
Gegeven Turingmachine T; stopt T altijd (al dan niet succesvol) voor alle woorden over zijn invoeralfabet?

AcceptsEverything is onbeslisbaar, dus om aan te tonen dat NoLoops ook onbeslisbaar is, is het volgens het vorige onderdeel voldoende om te laten zien dat AcceptsEverything reduceert naar NoLoops.

Uit een instantie T van AcceptsEverything construeren we een instantie T' van NoLoops met T' een Turingmachine die dezelfde taal accepteert als T maar die nooit woorden verwerpt door (expliciete of impliciete) reject. Deze constructie is inderdaad effectief (zie opgave 9.11: voorkom eerst dat de kop links van de band valt, voeg vervolgens een garbage state toe als vervanging van h_r waarin T' blijft loopen).
Nu geldt als T alle woorden accepteert, dan zal T' nooit loopen, en als er een woord is dat niet wordt geaccepteerd door T, dan zal T' voor dat woord een nooit-stoppende berekening hebben.
Dus AcceptsEverything reduceert naar NoLoops.
