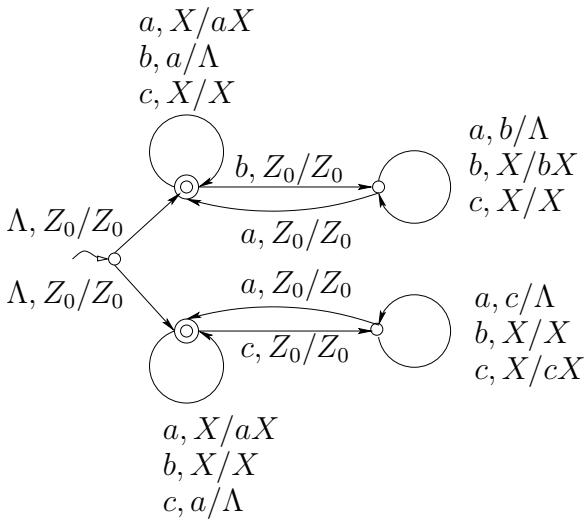
ANTWOORDEN tentamen FUNDAMENTELE INFORMATICA 3
vrijdag 25 januari 2008, 10.00 - 13.00 uur

Opgave 1
$L = \{x \in \{a,b,c\}^* \mid n_a(x) \geq n_b(x)\} \cup \{x \in \{a,b,c\}^* \mid n_a(x) \geq n_c(x)\}$.

a. Een stapelautomaat die L accepteert:



X staat voor elk willekeurig stapelsymbool.
Deze automaat bestaat uit 2 (deterministische) delen; in de begintoestand wordt gegokt (niet-determinisme) of het aantal a's groter/gelijk zal zijn dan het aantal b's of groter/gelijk zal zijn dan het aantal c's. (Woorden waarin allebei geldt, hebben dus zeker 2 accepterende berekeningen.) Na deze keuze [1] zijn er twee toestanden: een eindtoestand waarin het aantal gelezen a's groter/gelijk is dan het aantal gelezen b's (respectievelijk c's) en een toestand waarin het aantal gelezen a's kleiner is dan het aantal gelezen b's (respectievelijk c's).
De getekende automaat noemen we M; zijn begintoestand q_0, en van links naar rechts: de twee toestanden bovenin p_f en p en de twee onderin r_f en r.

b. Accepterende berekeningen van M (één accepterende berekening per woord geven is genoeg)
voor ba:
$(q_0, ba, Z_0) \vdash (p_f, ba, Z_0) \vdash (p, a, Z_0) \vdash (p_f, \Lambda, Z_0)$
of $(q_0, ba, Z_0) \vdash (r_f, ba, Z_0) \vdash (r_f, a, Z_0) \vdash (r_f, \Lambda, aZ_0)$.
voor $babc$:
$(q_0, babc, Z_0 \vdash^3 (r_f, bc, aZ_0) \vdash (r_f, c, aZ_0) \vdash (r_f, \Lambda, Z_0)$.
dit is de enige accepterende berekening van M voor $babc$.

[1] Een alternatieve methode zou na de eerste keuze voor elke gelezen a een symbool bijstapelen (zeg A, mits de top van de stapel een A of Z_0 is) en voor b's een symbool B bijstapelen (mits de top van de stapel een B of Z_0 is). Wanneer een a (b) wordt gelezen met B (A, respectievelijk) als top, dan wordt dit symbool gepopt. Als de top van de stapel A of Z_0 is, mag de automaat (niet-deterministisch) naar de accepterende toestand, vanwaar geen transities meer mogelijk zijn. De zo beschreven PDA heeft minder toestanden, maar meer niet-determinisme dan de hierboven getekende.

c. M is niet deterministisch: in q_0 zijn er twee transities mogelijk als Z_0 de top van de stapel is, zonder dat de keuze wordt bepaald door verschillende invoersymbolen. (Zie ook de opmerkingen bij **a**.)

d. Er bestaat geen DPDA die L accepteert met lege stapel. Immers zowel $babc$ als zijn prefix ba komen voor in L (zie onderdeel **b**). Een deterministische automaat heeft per woord hooguit één berekening. Als na afloop daarvan de stapel leeg is kan er geen verlenging van dat woord ook nog worden geaccepteerd. (Zie opgave 7.15.)

e. Gegeven is dat $K = \{a^i b^j c^k \mid i < j \text{ en } i < k\}$ geen context-vrije taal is.
Dan is L geen deterministische context-vrije taal. Immers, K is de gesorteerde versie van het complement van L: $K = (\{a, b, c\}^* - L) \cap \{a\}^*\{b\}^*\{c\}^*$.
De familie van DCFLs is gesloten onder complement, dus als L een DCFL is dan zijn complement ook. Verder is de familie van DCFLs gesloten onder doorsnijding met reguliere talen. Dus als het complement van L een DCFL is, dan ook K. Dit is in tegenspraak met het gegeven dat K zelfs geen CFL is.

Opgave 2

De context-vrije grammatica G heeft startsymbool S, terminaal alfabet $\{a, b, \$\}$ en producties $S \to T\$ \quad T \to TX \mid ab \mid \Lambda \quad X \to bXaa \mid ba$.

a. Eliminatie van de links-recursie in G (met nieuwe niet-terminaal U):
$T \to TX \mid ab \mid \Lambda$ wordt vervangen door $T \to abU \mid U$ en $U \to XU \mid \Lambda$.

b. Factorisatie toegepast op $X \to bXaa \mid ba$ (met nieuwe niet-terminaal V):
$X \to bV$ en $V \to Xaa \mid a$.

De grammatica H geconstrueerd bij de vorige onderdelen heeft producties
$S \to T\$ \quad T \to abU \mid U \quad U \to XU \mid \Lambda \quad X \to bV \quad V \to Xaa \mid a$.

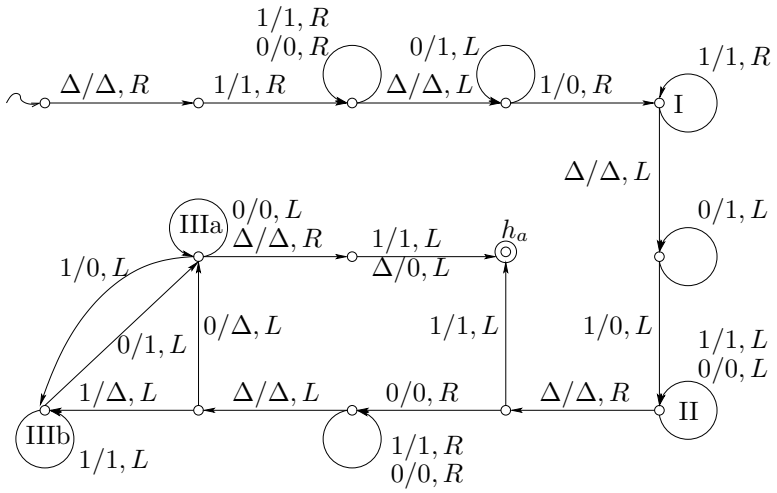
c. Lookahead voor de producties van H:
$\texttt{LA}_1(S \to T\$) = \{a, b, \$\}$
$\texttt{LA}_1(T \to abU) = \{a\}$ en $\texttt{LA}_1(T \to U) = \{b, \$\}$
$\texttt{LA}_1(U \to XU) = \{b\}$ en $\texttt{LA}_1(U \to \Lambda) = \{\$\}$
$\texttt{LA}_1(X \to bV) = \{b\}$
$\texttt{LA}_1(V \to Xaa) = \{b\}$ en $\texttt{LA}_1(V \to a) = \{a\}$.

d. H voldoet aan de LL(1) eigenschap. Immers, zoals we bij **c** zien, geldt voor elke niet-terminaal dat zijn producties geen gemeenschappelijke lookahead-symbolen hebben: voor alle niet-terminalen A geldt, als $A \to \alpha$ en $A \to \beta$ producties zijn met $\alpha \neq \beta$, dan $\texttt{LA}_1(A \to \alpha) \cap \texttt{LA}_1(A \to \beta) = \emptyset$.

Opgave 3

De Turingmachine T met invoeralfabet $\{0, 1\}$ zoals hieronder gegeven berekent de functie $f(n) = n - 2$ door twee keer 1 af te trekken van het gegeven binair gerepresenteerde invoergetal n.
Om $n - 1$ te berekenen als n binair gegeven is, loopt T eerst naar het rechteruiteinde van de invoer (waarbij gelijk gecontroleerd wordt dat de invoer met een 1 begint). Vervolgens beweegt hij naar links onderwijl elke 0 in een 1 veranderend totdat hij een 1 tegenkomt. Deze 1 wordt 0 en dan gaat de kop terug naar rechts (bij I) om deze procedure nog één keer te herhalen. (Als

de invoer 1 oorspronkelijk 1 was, blijft T nu hangen, omdat er geen 1 meer is om in 0 te veranderen.) T loopt helemaal terug naar links (bij II) en als het meest linkse symbool een 1 is, kan hij stoppen in h_a in de configuratie $(h_a, \underline{\Delta}\texttt{bin}(n-2))$.
Als het meest linkse symbool een (leidende) 0 is, dan loopt T helemaal naar rechts om vandaar alle symbolen een plek naar links op te schuiven (bij IIIa wordt onthouden dat een 0 wordt opgeschoven en bij IIIb dat het om een 1 gaat). Wanneer de kop links is aangekomen, ziet hij Δ terwijl hij een 0 onthoudt. Nu moet hij alleen nog controleren of band nu helemaal leeg is en de uitvoer dus 0 moet zijn, omdat de invoer 2 was. Dan kan hij stoppen in h_a.
Merk op dat er nooit meer dan één leidende 0 zal zijn na II; als je dat niet gelooft, dan moet je de opschuifprocedure blijven herhalen tot je na II inderdaad een 1 rechts van Δ ziet.



Een andere methode is om binair 2 (dus 10) af te trekken van (de binaire representatie van) n: loop naar het einde van de string; handhaaf het meest rechtse symbool, stap naar links en trek zoals boven beschreven 1 af van de resterende string. Ook nu moet je verifieren dat invoer en uitvoer de juiste vorm hebben en zorgen dat de uitvoer op de juiste plaats op de band staat.

Opgave 4

a. Definitie 10.1:
Een taal L is recursief opsombaar als er een Turingmachine T is die deze taal accepteert ($L = L(T)$).
L is recursief als er een Turingmachine T is die L herkent (d.w.z. beslist, dus T berekent de karakteristieke functie van L).

b.
Stelling 10.1: Elke taal die recursief is, is ook recursief opsombaar.
Bewijs: een eenvoudige constructie, zie boek.

De bewering:
"elke taal die recursief opsombaar is, is ook recursief"
is niet waar, zoals blijkt uit de taal SA die wel recursief opsombaar is, maar niet recursief (Stelling 11.2).

$K = \{x \in \{a,b\}^* \mid n_a(x) = 2^k \text{ voor een } k \geq 0\}$.

c. Een unrestricted (type 0) grammatica die K genereert, met startsymbool S en producties:
$S \to LARB \mid LAR$
$L \to LX \quad XR \to R \quad XA \to AAX$
$L \to \Lambda \quad R \to \Lambda \quad B \to BB \quad AB \to BA$
$A \to a \quad B \to b$
Deze grammatica kan alleen woorden genereren die minstens één a bevatten en 0 of meer b's. De symbolen L en R fungeren als zender en ontvanger van de boodschap X dat elke A moet verdubbelen. De productie $AB \to BA$ zorgt dat A's en B's door elkaar kunnen worden gezet.
Overigens is het ook niet zo moeilijk om een context-gevoelige grammatica voor deze taal te geven waarin woorden nooit korter worden door herschrijven.

d. K is recursief, want zijn karakteristieke functie is berekenbaar. Immers, het is eenvoudig om na te gaan of een gegeven woord w element is van K.
Ten eerste mag w alleen maar uit a's en b's bestaan. Verder kan je het aantal a's in w tellen en nagaan of dat getal een macht van 2 is (door herhaald door 2 delen). Omdat K recursief is, is K ook recursief opsombaar (zie **b**).

Opgave 5

a. De stelling van Rice (Stelling 11.9):
beslissingsprobleem P_R:

Gegeven Turingmachine T; heeft $L(T)$ eigenschap R?

is onbeslisbaar als R een niet-triviale eigenschap van Turingmachinetalen (de recursief opsombare talen) is (er zijn re talen met die eigenschap en er zijn er zonder die eigenschap).

Gegeven zijn twee beslissingsproblemen:

AcceptsNothing:
Gegeven Turingmachine T; is $L(T) = \emptyset$?

EmptyIntersection:
Gegeven twee Turingmachines T_1 en T_2; is $L(T_1) \cap L(T_2) = \emptyset$?

b. AcceptsNothing reduceert naar EmptyIntersection (zie Definitie 11.3):
Uit een instantie T van AcceptsNothing construeren we een instantie (T, T_0) van EmptyIntersection met T_0 de Turingmachine die alle woorden over het invoeralfabet Σ van T accepteert; dus $L(T_0) = \Sigma^*$. (Je kan ook $T_0 = T$ kiezen.)
Merk op dat deze transformatie inderdaad effectief is: het is eenvoudig om een Turingmachine te construeren met invoeralfabet Σ die Σ^* accepteert; bijvoorbeeld een TM met alleen een instructie van zijn begintoestand gelijk naar h_a ongeacht de invoer. (Instantie T omzetten in (T, T) is uiteraard ook effectief.)
Er geldt: $L(T) = \emptyset$ desdals $L(T) \cap \Sigma^* = \emptyset$ desdals $L(T) \cap L(T_0) = \emptyset$, (en $L(T) = \emptyset$ desdals $L(T) \cap L(T) = \emptyset$) dus
T is een ja-instantie van AcceptsNothing desdals (T, T_0) is een ja-instantie van EmptyIntersection.

c. EmptyIntersection reduceert naar AcceptsNothing (Definitie 11.3):
Uit een instantie (T_1, T_2) van EmptyIntersection construeren we een instantie

T van AcceptsNothing met T eenTuringmachine die $L(T_1) \cap L(T_2)$ accepteert. Hier maken we gebruik van het gegeven dat voor elk tweetal Turingmachinetalen een TM kan worden geconstrueerd die hun doorsnijding accepteert. Deze transformatie is dus effectief.
Er geldt: $L(T_1) \cap L(T_2) = \emptyset$ desdals $L(T) = L(T_1) \cap L(T_2) = \emptyset$
en dus is (T_1, T_2) een ja-instantie van EmptyIntersection desdals T een ja-instantie is van AcceptsNothing

d. Leeg zijn is een niet-triviale eigenschap van Turingmachinetalen: een TM die $\emptyset$ accepteert bestaat (bijvoorbeeld een TM zonder instructies) en er zijn TMs die wel woorden accepteren (zoals bijvoorbeeld T_0 in onderdeel **b**).
Uit de stelling van Rice volgt nu direct dat AcceptsNothing onbeslisbaar is.
Omdat AcceptsNothing naar EmptyIntersection reduceert, volgt dat ook EmptyIntersection onbeslisbaar moet zijn (zo niet dan hadden we op grond van **b** ook een algoritme om AcceptsNothing te beslissen).