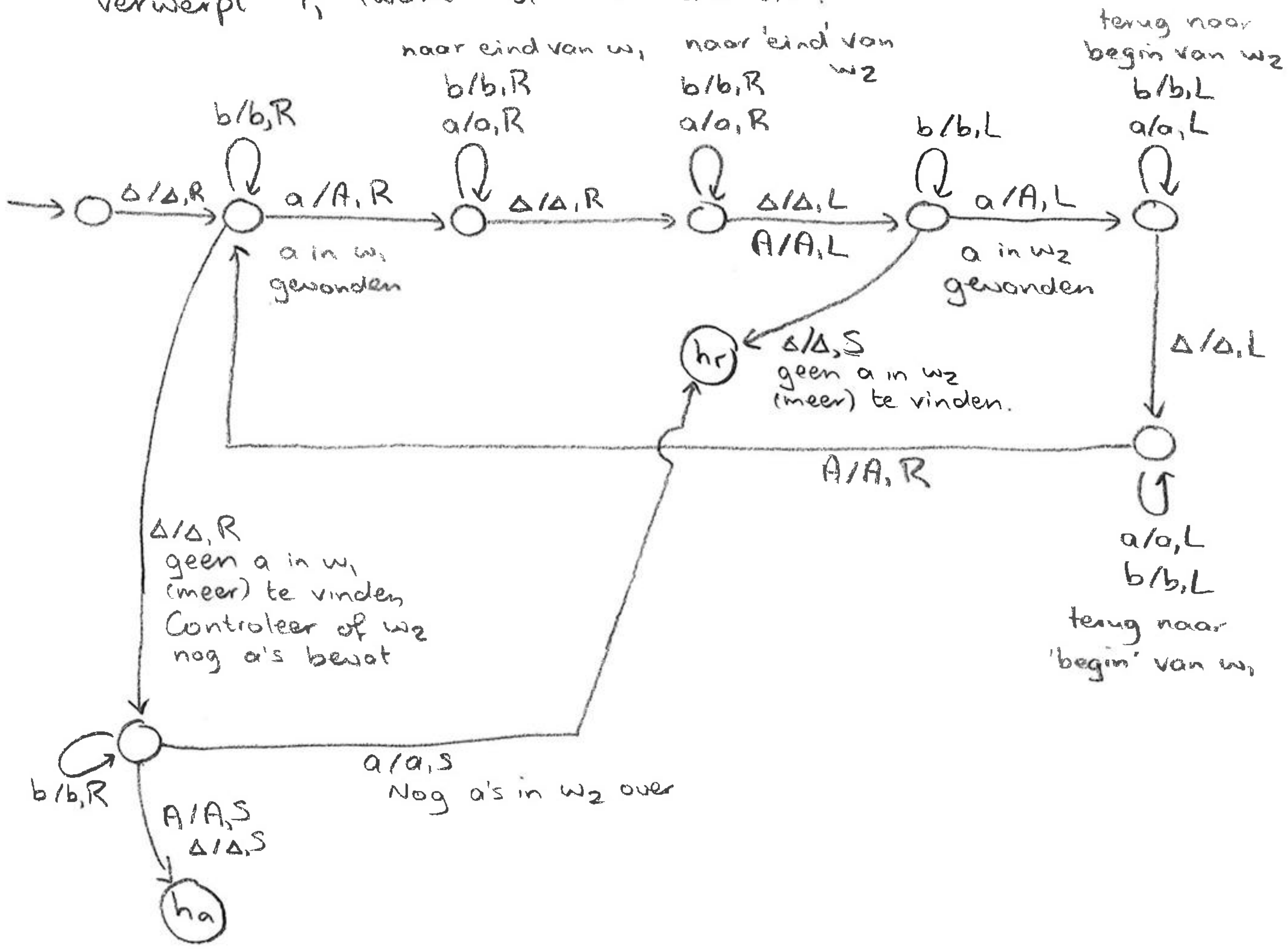
Uitwerking tentamen Fundamentele Informatica 3, vrijdag 3 juni 2016

13:08

1 (a) T_1 zoekt de eerste a in w_1 en markeert die door er een hoofdletter van te maken. Vervolgens zoekt hij de <u>laatste</u> a in w_2 en markeert die op dezelfde manier. Daarna de tweede a van w_1, de een-na-laatste a van w_2, enzovoort.

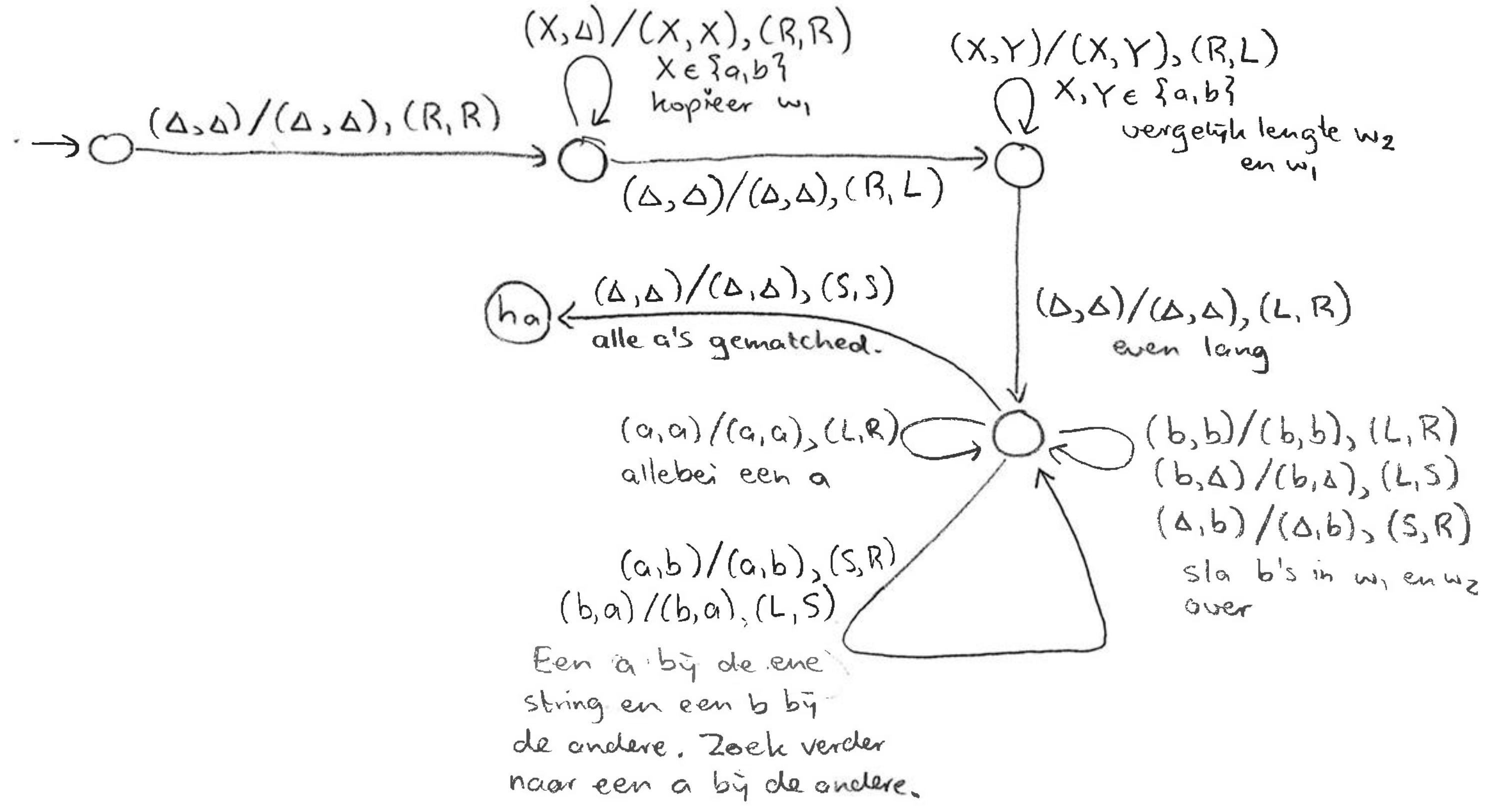
Dit stopt

* als er geen a in w_1 meer te vinden is. In dat geval controleert T_1 of ook de a's in w_2 op zijn. Zo ja, dan accepteert T_1, zo nee, dan verwerpt hij (want $n_a(w_1) < n_a(w_2)$)
* als er in w_2 geen a te vinden is die gemarkeerd kan worden tegen een reeds gemarkeerde a van w_1. In dat geval verwerpt T_1 (want $n_a(w_1) > n_a(w_2)$).



13.24.
14:06

(b) T_2 kopieert w_1 naar tape 2, controleert vervolgens of w_1 (op tape 2) en w_2 (op tape 1) even lang zijn, en zo ja, of w_1 en w_2 evenveel a's bevatten. Als dat het geval is, bevatten ze ook evenveel b's, en is w_2 dus een anagram van w_1 en kan T_2 accepteren.



14.23./25

2 (a) G kent de volgende producties

$S \to S'R$ creëer een variabele S' die letters genereert en een rechterkant R

$S' \to aS'A_1A_2 \mid bS'B_1B_2$ genereer: letters a, b voor x; letters A_1, B_1 voor y; letters A_2, B_2 voor x^r

De letters A_2, B_2 staan al in de goede volgorde ten opzichte van elkaar, vergeleken met de letters a, b voor x. Ze moeten alleen nog naar rechts verhuizen, want er staan letters A_1, B_1 tussen.

$S' \to L$ klaar met genereren letters

$A_2A_1 \to A_1A_2$ $A_2B_1 \to B_1A_2$
$B_2A_1 \to A_1B_2$ $B_2B_1 \to B_1B_2$ } A_2 en B_2 gaan naar rechts

$A_1B_1 \to B_1A_1$ $B_1A_1 \to A_1B_1$ letters A_1 en B_1 mogen van plaats verwisselen, want moeten anagram y gaan vormen

$LA_1 \to aL$ $LB_1 \to bL$ L loopt naar rechts en verandert hoofdletters van y in kleine letters

$A_2R \to Ra$ $B_2R \to Rb$ R loopt naar links en verandert hoofdletters van x^r in kleine letters

$LR \to \Lambda$ Klaar.

14.38

(b) 14.41.

Een afleiding in G voor $x = aababab aa$:

$$S \Rightarrow S'R \Rightarrow^* aabS'B_1B_2A_1A_2A_1A_2R \Rightarrow aabLB_1B_2A_1A_2A_1A_2R \Rightarrow^*$$
$$aabLB_1A_1A_1B_2A_2A_2R \Rightarrow aabLA_1B_1A_1B_2A_2A_2R \Rightarrow^* aababaLB_2A_2A_2R$$
$$\Rightarrow^* aababaLRbaa \Rightarrow aabababaa.$$

14.45.

3] (a) T moet zijn invoer $x \in \Sigma^*$ dus kunnen accepteren $\Leftrightarrow$ x kan gegenereerd worden door G.

T gaat daartoe (in de 2e fase) achter zijn invoer x een afleiding in G simuleren.

In de 1e fase moet T dus voorbij zijn invoer x lopen:

$\underline{\Delta} x \Delta \Delta \dots \xrightarrow[\text{fase 1}]{} \Delta x \underline{\Delta} \Delta \dots$

MovePastInput.

In de 2e fase wordt een string y gegenereerd vanuit S (= startsymbool van G)

$\Delta x \underline{\Delta} \Delta \dots \xrightarrow[\text{fase 2}]{} \Delta x \underline{\Delta} y \Delta \dots$

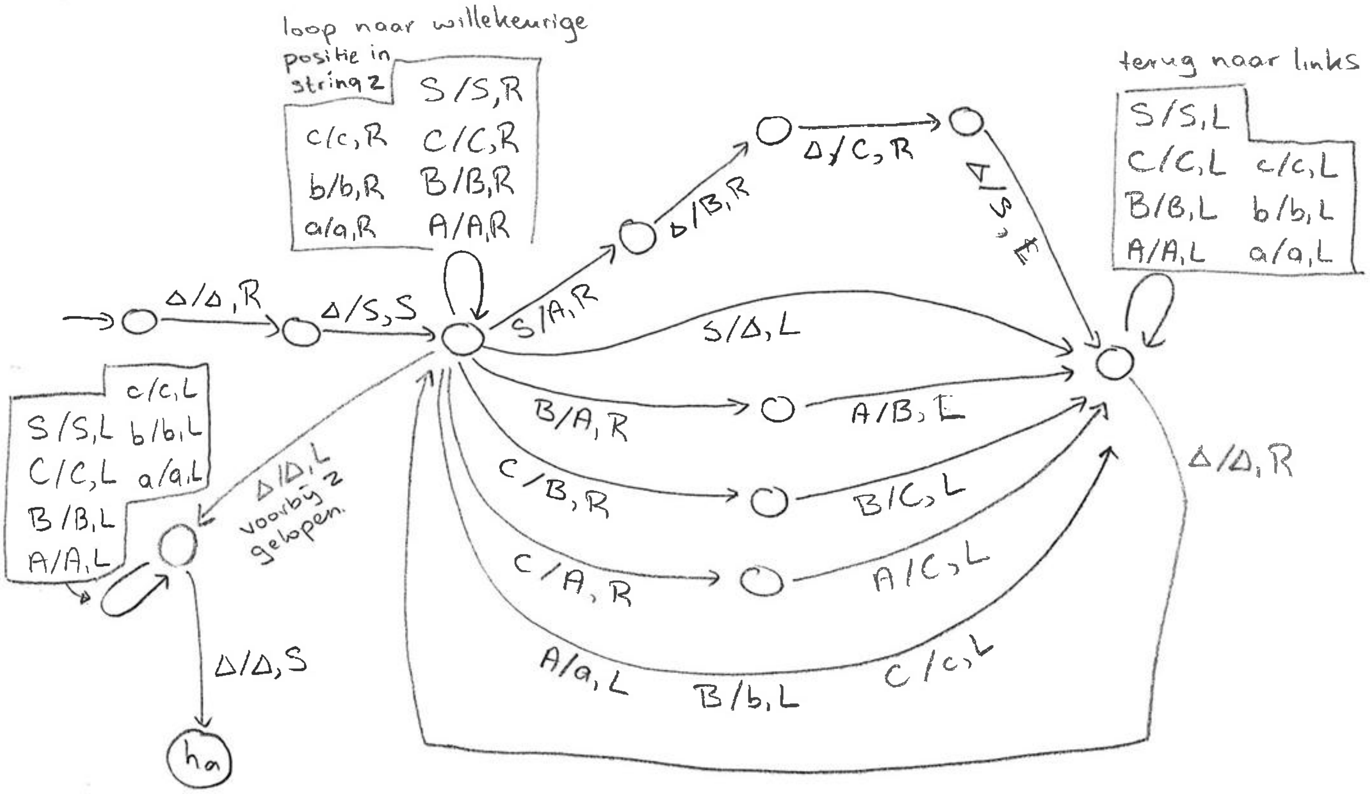
In de 3e fase controleert T of $x = y$. Zo ja, dan accepteert T x, en anders niet.
T loopt daartoe terug naar links en roept Equal aan.

Er geldt: $x \in L(G) \Leftrightarrow$ er is een afleiding $S \Rightarrow^* x$ in $G \Leftrightarrow$ er is een berekening van T die na fase 2 $\Delta x \underline{\Delta} x \Delta \dots$ oplevert $\Leftrightarrow$ er is een berekening van T die x accepteert. $\Leftrightarrow x \in L(T)$.

(b)

T_2 zet eerst het startsymbool S van G op de tape.
Vervolgens probeert T_2 herhaaldelijk een productie van G toe te passen. Daartoe gaat T_2 naar een willekeurige positie in de huidige string z (niet-deterministisch dus). Vanaf die positie leest T_2 de linkerkant van een productie (niet-deterministisch, als hier keuze voor is), onderwijl die linkerkant letter-voor-letter vervangend door de rechterkant van de productie.

Voor bijna elke productie geldt dat de rechterkant even lang is als de linkerkant. Vervangen op de tape is dan makkelijk te doen, zonder Inserts en Deletes.
Alleen de rechterkanten van de producties voor S zijn langer of korter dan S zelf. Omdat S tijdens een afleiding echter altijd helemaal aan het eind van de huidige string z staat ('toevallig' zo bij deze grammatica G), is het ook bij die producties gemakkelijk om de linkerkant door de rechterkant te vervangen.
T_2 stopt, zodra hij bij het zoeken van een positie helemaal voorbij z gelopen is.



4

11.40

(a) Een eigenschap R van Turingmachines is een niet-triviale taaleigenschap, wanneer

* R een taaleigenschap is, dat wil zeggen: als T_1 en T_2 Turingmachines zijn met $L(T_1) = L(T_2)$, dan hebben T_1 en T_2 òf allebei eigenschap R wel òf allebei eigenschap R niet.
* R een niet-triviale eigenschap is, dat wil zeggen: er zijn Turingmachines die eigenschap R wel hebben en er zijn Turingmachines die eigenschap R niet hebben.

11:46

11:47

(b) T_2 ziet er als volgt uit:

$$\to \text{MovePastInput} \to T_1 \to \text{EraseTape} \to \text{BackToSquare } 0 \to T_R$$

$$T_1 \to \cancel{h_a}$$

Hierin is T_R een Turing machine die eigenschap R wel heeft (zo'n Turingmachine bestaat, want R is niet-triviaal).

T_2 krijgt dus een invoer x, maar loopt daar eerst aan voorbij en simuleert dan T_1 op de (voor de rest lege) tape die T_2 daar voor zich ziet. Hierbij wordt uiteraard voorkomen (met een speciaal scheidingssymbool) dat T_2 naar links loopt tot op zijn eigenlijke invoer x. T_2 simuleert dus T_1 voor de lege string als invoer.

Wanneer T_1 zou accepteren, veegt T_2 de tape schoon rechts van zijn oorspronkelijke invoer x. Dit is mogelijk wanneer we bij het simuleren van T_1 een end-of-tape marker gebruiken.

Ten slotte gaat T_2 terug naar vakje 0 van de tape, vóór zijn oorspronkelijke invoer x en simuleert dan T_R op invoer x.

12.01

(c) Er geldt:

T_1 is ja-instantie van Accepts-Λ $\Leftrightarrow$ T_1 accepteert Λ $\Leftrightarrow$ T_1 bereikt h_a als hij met lege tape begint $\Rightarrow$ T_2 zal uiteindelijk T_R gaan simuleren voor invoer x $\Rightarrow$ (T_2 zal invoer x accepteren $\Leftrightarrow$ T_R accepteert x) $\Rightarrow$ $L(T_2) = L(T_R)$. Omdat T_R eigenschap R heeft, en R een taaleigenschap is, heeft T_2 in dit geval ook eigenschap R $\Rightarrow$ T_2 is een ja-instantie van P_R.

En er geldt ook:

T_1 is nee-instantie van Accepts-Λ $\Leftrightarrow$ T_1 accepteert Λ niet $\Leftrightarrow$ T_1 bereikt h_a niet als hij met een lege tape begint (T_1 verwerpt dus, crasht of komt in een oneindige lus) $\Rightarrow$ T_2 accepteert invoer x niet, onafhankelijk van wat x is $\Leftrightarrow$ $L(T_2) = \phi = L(T_0)$. Omdat volgens aanname T_0 eigenschap R niet heeft, en R een taaleigenschap is, heeft T_2 in dit geval ook eigenschap R niet $\Rightarrow$ T_2 is een nee-instantie van P_R

12.12

5(a)

Dat betekent dat

* f $n+1$ argumenten heeft
* en dat
  - voor elke $X \in \mathbb{N}^n$, $f(X,0) = g(X)$
  - voor elke $X \in \mathbb{N}^n$ en $k \geq 0$, $f(X,k+1) = h(X, k, f(X,k))$

(b) 12:15

Er geldt: voor elke $X \in \mathbb{N}^n$,

$m^P(X,0) = 0$, zowel als $\{y \leq 0 \mid P(X,y) \text{ is true}\}$ niet leeg is (want dan is $P(X,0)$ is true en is de verzameling gelijk aan $\{0\}$), als wanneer de verzameling wel leeg is.

De functiewaarde 0 moet gelijk zijn aan $g(X)$, en dat is het geval voor $g = C_0^n$. Dit is een initiële functie, en dus primitief recursief

voor elke $X \in \mathbb{N}^n$ en $k \geq 0$,

$$m^P(X,k+1) = \begin{cases} k+1 & \text{als } P(X,k+1) \text{ is true} \\ m^P(X,k) & \text{als } P(X,k+1) \text{ is false,} \end{cases}$$

zowel als $m^P(X,k)$ een 'echte waarde' is, omdat er een $y \leq k$ is met $P(X,y)$ is true, als wanneer $m^P(X,k)$ de default waarde 0 is, omdat er niet zo'n y is.

De functiewaarde $m^P(X,k+1)$ moet gelijk zijn aan $h(X,k,m^P(X,k))$

Dat is het geval voor

$$h(X, x_2, x_3) = \begin{cases} s(x_2) & \text{als } P(X, s(x_2)) \text{ is true} \\ x_3 & \text{als } P(X, s(x_2)) \text{ is false.} \end{cases}$$

Omdat h het resultaat is van volledige gevalsonderscheiding met primitieve recursieve functies en primitieve recursieve predikaten, is h zelf ook primitief recursief.

Omdat m^P ontstaat uit primitieve recursieve functies g en h door de operatie primitieve recursie, is m^P zelf ook primitief recursief.

12.31 / 12.33