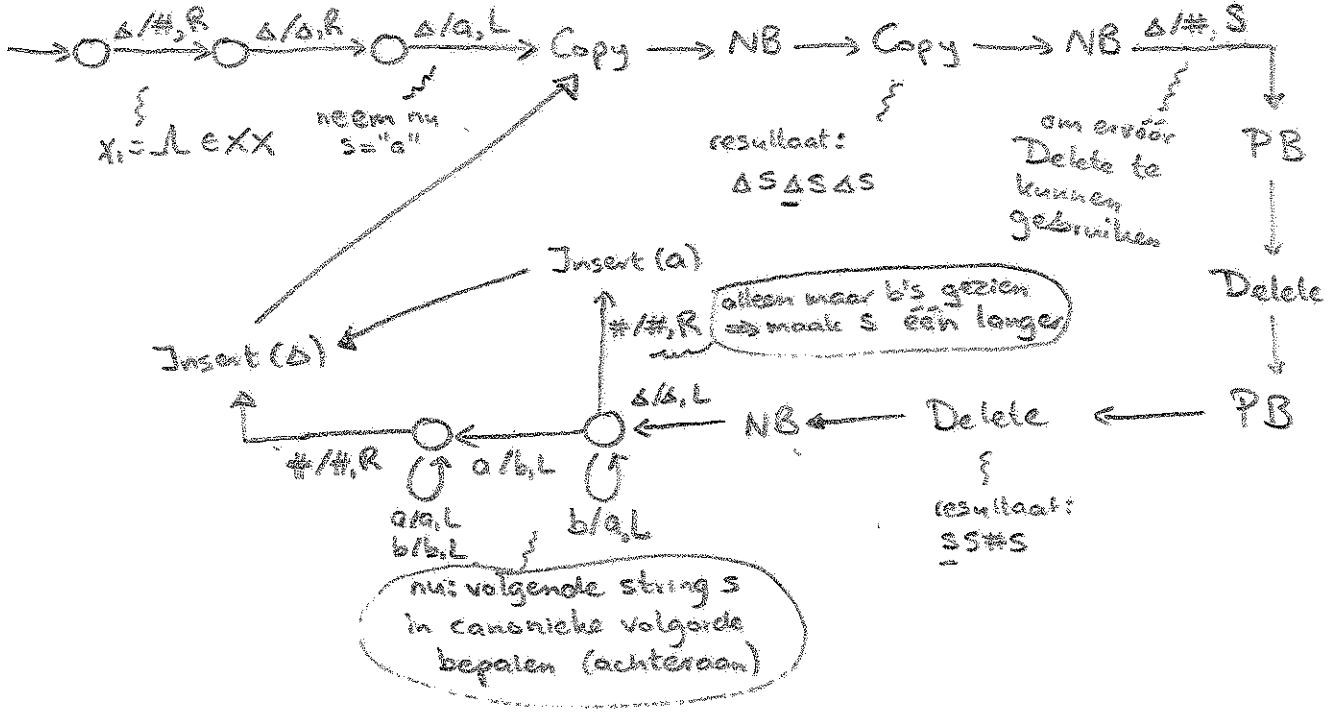
13:30

1(a) T:



13:45

T zet eerst (als het ware) $X_1 = \Lambda$ op de tape, en zet daarna voor $s =$ "a", "b", "aa", "ab", ... (in canonieke volgorde) $X_1 = SS$ op de tape.
Daartoe zet T steeds drie exemplaren van s achter elkaar op de tape (met twee keer Copy). De eerste twee exemplaren worden samengevoegd tot $X_1 = SS$. Het derde exemplaar wordt gebruikt om de opvolger van s in canonieke volgorde te bepalen.
Het bepalen van de opvolger van s in canonieke volgorde gaat van achter naar voren: we zoeken de achterste a in s (waarbij we de b's erachter, die we eerder tegenkomen, veranderen in a), en veranderen die a in b. Als s helemaal geen a bevat, maken we s een positie langer (met een a ervoor).

13:53.
14:15

(b) * Bij 'a TM enumerating a language' gaat de leeskop op de eerste tape nooit naar links. Zo'n TM heeft in de praktijk dan ook vaak meerdere tapes.
Bij de TM T uit onderdeel (a) gaat de leeskop op de (enige) tape wel regelmatig naar links
* Bij 'a TM enumerating a language' wordt een non-blank symbool op de eerste tape nooit overschreven door een ander symbool
Bij de TM T uit onderdeel (a) wordt regelmatig een non-blank symbool op de (enige) tape overschreven door een ander symbool. We zien bijvoorbeeld expliciet transities van de vorm a/b,L of b/a,L. Maar ook in de functies Copy, Insert en Delete worden non-blank symbolen door andere symbolen overschreven.

4:25

2] We gaan de letters in de 'invoer' x een voor een van links naar rechts verdubbelen. Hiertoe sturen we een variabele F van links naar rechts. De verdubbelde letters ($\hat{a}$ voor a en $\hat{b}$ voor b) lopen op hun beurt, achter F aan, naar rechts, waar ze weer veranderen in de corresponderende letters.

$A \rightarrow CF$	produceer verdubbelaar F en verander A in C (zodat we ook niet nogmaals een verdubbelaar kunnen produceren)
$Fa \rightarrow a\hat{a}F$ $Fb \rightarrow b\hat{b}F$	verdubbel letter uit 'invoer' x. F loopt voor $\hat{a}$ / $\hat{b}$ uit, zodat de verdubbelde letters in dezelfde volgorde komen te staan als de oorspronkelijke letters
$FB \rightarrow MD$	alle letters zijn verdubbeld. Creëer een midden M waar de verdubbelde letters overheen moeten springen om weer een gewone letter te kunnen worden
$\hat{a}a \rightarrow a\hat{a}$ $\hat{a}b \rightarrow b\hat{a}$ $\hat{b}a \rightarrow a\hat{b}$ $\hat{b}b \rightarrow b\hat{b}$	loop naar rechts met de verdubbelde letters. De verdubbelde letters $\hat{a}$ / $\hat{b}$ kunnen niet over elkaar heenspringen. Ze komen dus in de 'goede' volgorde rechts aan
$\hat{a}M \rightarrow Ma$ $\hat{b}M \rightarrow Mb$	de verdubbelde letters springen over M heen, en worden gewone letters. Ze blijven in dezelfde, 'goede' volgorde staan.
$M \rightarrow \Lambda$	M verdwijnt en we zijn klaar. Merk op dat dit pas kan/moet gebeuren als alle verdubbelde letters over M heengesprongen zijn. Anders komen we niet meer van de resterende verdubbelde letters (variabelen) af.

3 a] Tussen de haken in de productie $p(\sigma_1 a) \to (\sigma_1 b) q$ staan steeds twee symbolen, omdat we

* aan de ene kant de oorspronkelijke invoer x waarvoor we T simuleren ongeschonden willen bewaren. Als T accepteert voor deze invoer, moet G namelijk x moeten kunnen reconstrueren. Het eerste symbool tussen de haken is dus het symbool dat op de betreffende tape positie stond bij het begin van de berekening van T voor x.
  $\sigma_1 \in \Sigma \cup \{\Delta\}$, want zulke symbolen staan er op de tape bij het begin van een berekening van T.
* aan de andere kant de huidige inhoud van de tape willen bijhouden tijdens het simuleren van T. De huidige inhoud van de tape bepaalt immers (mede) de volgende stappen van T, en dus ook of T de accepting state h_a zal bereiken. Het tweede symbool tussen de haken is dus het symbool dat op de betreffende tape positie staat op het huidige moment in de gesimuleerde berekening

14:53

b] Producties van de eerste soort:

$S \to S(\Delta\Delta) \mid T$ — produceer genoeg Δ's achter de invoer x van T, om de complete berekening van T voor invoer x (als x door T geaccepteerd wordt) te kunnen beschrijven. T kan namelijk tijdens de berekening vakjes op de tape achter x nodig hebben, en daar staan initieel Δ's

$T \to T(\sigma\sigma_1)$ voor $\sigma_1 \in \Sigma$ — genereer een willekeurige invoer $x \in \Sigma^*$ voor T (van rechts naar links)

$T \to q_0(\Delta\Delta)$ — produceer de Δ vóór de invoer x op de tape, en de initiële toestand q_0 om de complete configuratie (tapeinhoud, toestand en (impliciet) positie van leeskop) te beschrijven.

15:00

15:12

c] Producties van de derde soort

$h_a(\sigma_1\sigma_2) \to h_a(\sigma\sigma_2)h_a$

$(\sigma_1\sigma_2)h_a \to h_a(\sigma\sigma_2)h_a$

$\sigma_1 \in \Sigma \cup \{\Delta\}$

$\sigma_2 \in \Gamma \cup \{\Delta\}$

} propageer de accepterende toestand h_a, die bij acceptatie van x door T in de configuratie is gekomen, over de hele string

$h_A(\Delta\sigma_2) \longrightarrow \Lambda$ voor $\sigma_2 \in \Gamma \cup \{\Delta\}$
$h_A(\sigma_1\sigma_2) \longrightarrow \sigma_1$ voor $\sigma_1 \in \Sigma$, $\sigma_2 \in \Gamma \cup \{\Delta\}$ } filter alleen de invoer x uit de verkregen string

Als het eerste symbool tussen de haken Δ is, dan hoorde dit symbool niet tot x (maar het stond initieel voor of achter x op de tape) $\Rightarrow$ schrap (ook) dit symbool
Als het eerste symbool tussen de haken $\sigma_1 \in \Sigma$ is, dan hoorde dit symbool wel tot x $\Rightarrow$ laat (alleen) dit symbool over.

15:21

4 a) We noemen een eigenschap R van Turing machines een niet-triviale taaleigenschap, als

* R een taaleigenschap van Turing machines is, d.w.z.: als TMs T_1 en T_2 dezelfde taal accepteren ($L(T_1) = L(T_2)$), dan geldt
  T_1 heeft eigenschap R $\Leftrightarrow$ T_2 heeft eigenschap R
* als R een niet-triviale eigenschap van Turing machines is, d.w.z.:
  ! er is minstens één TM die eigenschap R wel heeft
  - en er is minstens één TM die eigenschap R niet heeft

15:26

b) AcceptsSubsetXX
Hierbij is aan de voorwaarden van de stelling van Rice voldaan. Immers,
* de instanties van AcceptsSubsetXX zijn Turing machines T
* de eigenschap die getest wordt, is een niet-triviale taaleigenschap van Turing machines. De vraag is namelijk (in andere woorden): is $L(T) \subseteq XX$, en daarvoor geldt:
  - als $L(T_1) = L(T_2)$, dan geldt $L(T_1) \subseteq XX \Leftrightarrow L(T_2) \subseteq XX$
  - • er zijn TMs die de eigenschap wel hebben, bijvoorbeeld een TM T die niets accepteert

want $L(T) = \emptyset \subseteq XX$

  • er zijn TMs die de eigenschap niet hebben, bijvoorbeeld een TM T die alles accepteert

En als het invoeralfabet van T $\Sigma = \{a,b\}$ is, dan is $L(T) = \{a,b\}^* \not\subseteq XX$

15.34

AcceptsIntersectionWithXX

Hierbij is niet aan de voorwaarden van de stelling van Rice voldaan, omdat de instanties van AcceptsIntersectionWithXX zijn tweetallen TMs (T_1, T_2) in plaats van afzonderlijke TMs T

AcceptsUnionIsXX

Hierbij is op dezelfde manier niet aan de voorwaarden van de stelling van Rice voldaan.

15:39
15 uur

c) De stelling van Rice is direct toepasbaar op AcceptsSubsetXX. Dat beslissingsprobleem is derhalve niet beslisbaar

We gaan nu AcceptsSubsetXX reduceren naar AcceptsIntersectionWithXX

	AcceptsSubsetXX	$\leq$	AcceptsIntersectionWithXX
instantie	TM T		TMs (T_1, T_2)

Voor een gegeven instantie T van AcceptsSubsetXX kiezen we $T_1 = T$ en $T_2 = T$.

Het is duidelijk dat (T_1, T_2) op algoritmische wijze uit T geconstrueerd kan worden.

En er geldt:

T is ja-instantie van AcceptsSubsetXX $\iff L(T) \subseteq XX \iff$

$L(T) = L(T) \cap XX \implies$

$L(T_1) = L(T_2) \cap XX \implies$

(T_1, T_2) is ja-instantie van AcceptsIntersectionWithXX

algemeen geldt: $A \subseteq B \iff A = A \cap B$

Er is dus inderdaad aan de eisen van een reductie voldaan.

Omdat dus

* AcceptsSubsetXX niet beslisbaar is
* en AcceptsSubsetXX $\leq$ AcceptsIntersectionWithXX

is volgens Stelling 9.7 uit het boek ook AcceptsIntersectionWithXX niet beslisbaar

15:52

5 a)

$f_4(x_1,x_2) = x_2+1$

$f_5(x_1,x_2) = x_2+2$

Er geldt:

f_6 heeft één argument, want f_1 heeft nul argumenten en
16:30 f_5 heeft twee argumenten.

16:35

$f_6(0) = f_1(\cdot) = 0$

$f_6(k+1) = f_5(k, f_6(k)) = f_6(k)+2$

$f_6(1) = 2$
$f_6(2) = 4$
$f_6(3) = 6$
In het algemeen: $f_6(x) = 2*x$

16:39

b)

Er geldt

$R(y,0) = Mod(0,y) = 0$ voor elke $y \in \mathbb{N}$ (zowel $y=0$, als $y>0$).

$$R(y,k+1) = Mod(k+1,y) = \begin{cases} k+1 & \text{als } y=0 \\ \text{de rest bij deling van } k+1 \text{ door } y & \text{als } y>0 \end{cases}$$

$$= \begin{cases} Mod(k,y)+1 = R(y,k)+1 & \text{als } R(y,k)+1 < y \\ 0 & \text{als } R(y,k)+1 = y \end{cases}$$

Omdat $R(y,x) = Mod(x,y)$ nooit $\geq y$ wordt (als $y \geq 1$),
zal $R(y,x)+1$ nooit $> y$ worden.
De conditie $R(y,k)+1 = y$ is daarom equivalent aan de conditie
16:52 $R(y,k)+1 \geq y$ (als $y \geq 1$).
16:54

We kunnen daarom de formule voor $R(y,k+1)$ herschrijven tot

$$R(y,k+1) = \begin{cases} k+1 & \text{als } y=0 \\ R(y,k)+1 & \text{als } y \geq 1 \text{ en } R(y,k)+1 < y \\ 0 & \text{als } y \geq 1 \text{ en } R(y,k)+1 \geq y \end{cases}$$

16:57
10:30

Deze laatste beschrijving van $R(y,k+1)$ moet van de vorm
$h(y,k,R(y,k))$ zijn.

R wordt dus door primitieve recursie verkregen uit de volgende functies g en h:

$$g(x_1) = C_0^1(x_1)$$

$$h(x_1, x_2, x_3) = \begin{cases} S(x_2) & \text{als } x_1 = 0 \\ S(x_3) & \text{als } x_1 \geq 1 \text{ en } S(x_3) < x_1 \\ 0 & \text{als } x_1 \geq 1 \text{ en } S(x_3) \geq x_1 \end{cases}$$

De functie g is een initiële functie, en dus primitief recursief.
De functie h is het resultaat van (volledige) gevalsonderscheiding met eenvoudige primitieve recursieve functies en primitieve recursieve predikaten, en is daarom ook primitief recursief.
Gevolg is dat de functie R primitief recursief is.

10:40