HERTENTAMEN FUNDAMENTELE INFORMATICA 3
Maandag 27 mei 2019, 14.00 - 17.00 uur

Dit tentamen bestaat uit zeven opgaven, waarbij steeds tussen [en] staat hoeveel punten er ongeveer mee te verdienen zijn. In totaal zijn er 100 punten te verdienen.
Geef de gevraagde Turingmachines door middel van hun transitiediagram.
Wanneer er bij een vraag om uitleg of motivatie gevraagd wordt, is het belangrijk om die ook te geven.
Als je het antwoord op een onderdeel niet weet, en je hebt dat antwoord nodig bij een later onderdeel, dan kun je het antwoord 'kopen' bij de docent.

---

1. [17 pt] De binaire representatie van een natuurlijk getal is óf de string `0` óf een bitstring die begint met een `1`.

   Construeer een Turingmachine T die de unaire representatie van een natuurlijk getal omzet in de binaire representatie. Om precies te zijn T moet de functie $f : \{1\}^* \to \{0,1\}^*$ berekenen, gedefinieerd door

   $$f(1^n) = \text{ de binaire representatie van } n.$$

   Bijvoorbeeld $f(111111111) = 1001$.

   Hint: bouw de bitstring van achter naar voren op, door herhaaldelijk n mod 2 en n div 2 uit te voeren.

   Leg ook duidelijk uit hoe T werkt.

   Je mag voor T gebruik maken van de componenten *NB*, *PB*, *Insert*(σ) en *Delete* zoals die in het boek beschreven zijn. Andere componenten mag je alleen gebruiken als je ze zelf uitwerkt (dus tekent). Wellicht ten overvloede:

   - *NB* verplaatst de leeskop naar de eerste Δ rechts van de huidige positie,
   - *PB* verplaatst de leeskop (zo mogelijk) naar de eerste Δ links van de huidige positie,
   - *Insert*(σ) verandert de tape-inhoud van $y\underline{z}$ in $y\underline{\sigma}z$ (waarbij z geen Δ bevat),
   - *Delete* verandert de tape-inhoud van $y\underline{\sigma}z$ in $y\underline{z}$ (waarbij z geen Δ bevat).

---

2. [10 pt] Laat $L \subseteq \Sigma^*$ een taal zijn, en laat $T_1 = (Q, \Sigma, \Gamma, q_0, \delta)$ een Turingmachine zijn, zó dat $L(T_1) = L$.

   Leg precies uit (desgewenst ondersteund met een tekening) hoe je T_1 kunt ombouwen naar een Turingmachine T_2 die dezelfde taal L accepteert, maar die alleen nog maar halt voor invoer $x \in L$.

---

3. [11 pt] Laat $L_1 \subseteq \{a, b\}^*$ een taal zijn, en laat T_1 een Turingmachine zijn, zó dat $L(T_1) = L_1$. Construeer een niet-deterministische Turingmachine T_2 die de taal L_2 van de prefixen van L_1 accepteert. Ofwel

$$L(T_2) = L_2 = \{x \in \{a, b\}^* \mid xy \in L_1 \text{ voor zekere } y \in \{a, b\}^*\}$$

Je mag voor T_2 gebruik maken van de componenten *NB*, *PB*, *Insert*(σ) en *Delete* zoals die bij opgave 1 beschreven zijn, en van T_1 als component. Andere componenten mag je alleen gebruiken als je ze zelf uitwerkt (dus tekent).

Leg ook duidelijk uit hoe T_2 werkt, en beredeneer dat T_2 inderdaad de taal L_2 accepteert.

---

4. [19 pt] Laat

$$L = \{a^i b^i c^j \mid i, j \geq 0 \text{ en } i \neq j\}.$$

   (a) Geef de eerste zes elementen van L in de canonieke volgorde.

   (b) Geef een *unrestricted grammar* G, zó dat $L(G) = L$.
   Leg uit wat de functie is van de diverse variabelen en producties in G.

   (c) Geef een afleiding in je grammatica G uit onderdeel (b), voor het woord *aabbc*. Wanneer je in de afleiding een aantal 'gelijksoortige' producties achter elkaar toepast, mag je die stappen samenvatten met behulp van $\Rightarrow^*$.

---

5. [14 pt] In het bewijs van Stelling 8.14 in het boek wordt beschreven, hoe je bij een willekeurige Turingmachine $T = (Q, \Sigma, \Gamma, q_0, \delta)$ een unrestricted grammar $G = (V, \Sigma, S, P)$ kunt construeren, zó dat $L(G) = L(T)$.

   De grammatica G bevat drie soorten producties. De eerste soort is bedoeld om initiële configuraties van de Turingmachine na te bouwen. Wanneer we met G een berekening van T voor de invoer $a_1 \ldots a_n \in \Sigma^*$ willen simuleren, wordt met de producties van de eerste soort de volgende string gegenereerd:

$$q_0(\Delta\Delta)(a_1 a_1) \ldots (a_n a_n)(\Delta\Delta) \ldots (\Delta\Delta)$$

   De tweede soort is voor het simuleren van berekeningen van de Turingmachine. De derde soort is voor het reconstrueren van de invoer van de Turingmachine, aan het eind van de berekening.

   (a) Als T de invoer $a_1 \ldots a_n$ accepteert, hoe ziet de string in de grammatica G er dan uit, direct na het simuleren van de berekening van T? Wees precies in je beschrijving van de string.

   (b) Geef de producties van de derde soort in G, de producties om de invoer van T te reconstrueren als die invoer geaccepteerd wordt. Leg ook uit wat de functie is van de verschillende producties.
   Je mag de producties algemeen beschrijven in termen van elementen σ van Σ en/of Γ en/of $\{\Delta\}$. Als alternatief mag je ook aannemen dat $\Sigma = \{a, b\}$ en dat $\Gamma = \{a, b, c\}$.

---

6. [10 pt] Een Turingmachine met twee bitstrings als invoer kan gebruikt worden om (bitsgewijze) logische operaties te berekenen. Zo kan een Turingmachine de functie $f_{\text{NOR}} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ berekenen, die gedefinieerd wordt door

$$f_{\text{NOR}}(x,y) = \begin{cases} \text{de logische NOR van } x \text{ en } y & \text{als } |x| = |y| \\ \text{niet gedefinieerd} & \text{anders} \end{cases}$$

Bijvoorbeeld: $f_{\text{NOR}}(1001, 0011) = 0100$. Wellicht ten overvloede: NOR staat voor NOT OR.

Ook kan een Turingmachine de functie $f_{\text{AND}} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ berekenen, die gedefinieerd wordt door

$$f_{\text{AND}}(x,y) = \begin{cases} \text{de logische AND van } x \text{ en } y & \text{als } |x| = |y| \\ \text{niet gedefinieerd} & \text{anders} \end{cases}$$

Bijvoorbeeld: $f_{\text{AND}}(1001, 0011) = 0001$.

We kunnen nu de volgende twee beslissingsproblemen definiëren:

> *IsNOR*: Gegeven een Turingmachine T_1 die twee bitstrings als invoer verwacht, berekent T_1 de functie f_{NOR} ?
>
> *IsAND*: Gegeven een Turingmachine T_2 die twee bitstrings als invoer verwacht, berekent T_2 de functie f_{AND} ?

Toon aan dat $\mathit{IsNOR} \leq \mathit{IsAND}$. Laat uiteraard ook zien dat aan alle eisen van een reductie is voldaan.

Z.O.Z.

7. [19 pt]

   (a) Wanneer noemen we een eigenschap R van Turingmachines een niet-triviale *taaleigenschap* (*language property*) ?

   De stelling van Rice luidt als volgt:

   > Als R een niet-triviale taaleigenschap van Turingmachines is, dan is het beslissingsprobleem
   >
   > P_R:
   > Gegeven een Turingmachine T, heeft T eigenschap R ?
   >
   > niet beslisbaar.

   In het boek wordt de stelling van Rice bewezen met behulp van een reductie tussen *Accepts*-Λ en P_R, waarbij *Accepts*-Λ als volgt gedefinieerd is:

   > *Accepts*-Λ: Gegeven een Turingmachine T, is $\Lambda \in L(T)$?

   Gegeven is dat *Accepts*-Λ niet beslisbaar is.

   (b) Moeten we, om de stelling van Rice op deze manier te bewijzen, aantonen dat *Accepts*-$\Lambda \leq P_R$ of dat $P_R \leq$ *Accepts*-Λ ? Motiveer je antwoord.

   (c) In het bewijs in het boek wordt onder meer verwezen naar een Turingmachine T_0 die helemaal niets accepteert. Neem aan dat T_0 eigenschap R niet heeft, en laat T_R een Turingmachine zijn die eigenschap R wel heeft. Dan wordt er bij een instantie T_1 van het ene beslissingsprobleem een instantie T_2 van het andere probleem gemaakt, die als volgt werkt:

   - T_2 krijgt een invoer x,
   - loopt voorbij x,
   - voert dan T_1 uit.
   - Als T_1 h_a bereikt,
     - veegt T_2 de tape rechts van x schoon,
     - gaat T_2 terug naar positie 0 op de tape, links van x,
     - en voert dan T_R uit.

   i. Van welk probleem (*Accepts*-Λ of P_R) moet T_1 een instantie zijn om de bij onderdeel (b) gewenste reductie te krijgen, en van welk probleem wordt T_2 dan een instantie?

   ii. Wat is de taal die T_2 accepteert, afhankelijk van instantie T_1? Motiveer je antwoord.

   iii. Toon aan dat met deze constructie inderdaad geldt dat T_1 een ja-instantie van het ene probleem is, dan en slechts dan als T_2 een ja-instantie van het andere probleem is.

`einde tentamen`