

Oefeningenbundel voor het vak *Automaten, berekenbaarheid & complexiteit*

Jens Bossaert
jens.bossaert@ugent.be
2017–2018

Dit is de oefeningenbundel voor het vak *Automaten, berekenbaarheid & complexiteit*, gegeven aan de Universiteit Gent, academiejaar 2017–2018, tweede semester.

De oefeningen vinden plaats op maandagochtend van 8.30 u. tot 11.15 u. in auditorium A3 of computerzaal Konrad Zuse (campus de Sterre, gebouw S9), volgens onderstaande kalender.

19/02: A3, **26/02:** A3, **05/03:** Zuse, **12/03:** Zuse, **19/03:** A3,
26/03: Zuse, **16/04:** A3, **23/04:** Zuse, **07/05:** A3, **14/05:** A3.

Wijzigingen hierin worden aangekondigd via Minerva.

1. We proberen de meeste opgaven tijdens de oefeningenles te bespreken.
- * 2. Opgaven met een sterretje zijn extra oefeningen en worden niet behandeld in de les.
- ⚡ 3. Opgaven met dit symbool zijn moeilijkere oefeningen, voor wie meer uitdaging wil.

Met vragen kun je steeds terecht tijdens deze oefeningensessies of via een e-mail. Uiteraard mogen ook vragen over de extra of over de uitdagende oefeningen gesteld worden.

Bij sommige opgaven is de software JFLAP (*Java Formal Languages and Automata Package*) handig of noodzakelijk. JFLAP is gratis beschikbaar op www.jflap.org, waar men tevens meer informatie kan vinden.

Gelieve fouten of andere onvolkomenheden in deze bundel te melden via een mailtje.

Reeks 1

* 1. Bewijs de volgende eigenschappen van verzamelingen.

1. $\overline{\emptyset} = \Omega$
2. $\overline{\overline{X}} = X$
3. $\overline{X \cup Y} = \overline{X} \cap \overline{Y}$
4. $\overline{X \cap Y} = \overline{X} \cup \overline{Y}$
5. $|\mathcal{P}(X)| = 2^{|X|}$ voor eindige X (via inductie)

2. Zij Σ een alfabet, dan geldt voor alle $w \in \Sigma^*$ en $n \in \mathbb{N}$ dat $|w^n| = n \cdot |w|$. Bewijs dit via inductie.

Voor $n = 0$ is $w^0 = \varepsilon$ en dus $|w^0| = 0 = 0 \cdot |w|$.

Veronderstel dat de eigenschap voldaan wordt voor n , dus dat $|w^n| = n \cdot |w|$. Dan is

$$|w^{n+1}| = |w^n \cdot w| = |w^n| + |w| = n \cdot |w| + |w| = (n + 1) \cdot |w|,$$

dus geldt de eigenschap ook voor $n + 1$. Per inductie geldt de uitspraak voor alle $n \in \mathbb{N}$.

3. “Bewijs” door middel van inductie op het aantal schapen in een wei: *als minstens één schap wit is, dan zijn ze allemaal wit*. Zowel de basisstap ($n = 0$ of $n = 1$) als de inductiestap worden eenvoudig bewezen; wat gaat er mis?

Het is verleidelijk om de volgende redenering op te bouwen.

Voor $n = 1$ is de opgave inderdaad duidelijk: als één schap wit is, dan zijn alle schapen (namelijk die ene) wit. Onderstel dat de uitspraak geldt voor n en beschouw dan $n + 1$ schapen, waaronder minstens één witte, die wij Castor noemen. Haal een ander schap, Pollux, uit de wei. De inductiehypothese op de overige n schapen (waaronder witte Castor) leert dat deze allen wit zijn. Zet nu Pollux terug en haal Castor weg. Ook deze n schapen zijn dan volgens de inductiehypothese allemaal wit, dus in het bijzonder ook Pollux. Dat betekent dat alle $n + 1$ schapen wit zijn, en de inductiestap is aangetoond.

Waar zit nu de fout? Het probleem schuilt hem in het geval $n = 2$. Als we in dat geval Pollux uit de wei halen, dan blijft slechts Castor over, die per assumptie wit is. Daarna echter, als we Castor weghalen, blijft slechts Pollux over en kunnen we niet langer besluiten dat ook dit schap wit is. Met andere woorden, voor $n = 2$ loopt deze redenering al spaak, en aangezien de gevallen $n > 2$ impliciet op het geval $n = 2$ steunen (door het inductieprincipe) zijn deze eveneens ongeldig.

4. Beschouw de talen $\mathcal{L}_1 = \{x^n y^n : n > 0\}$ en $\mathcal{L}_2 = \{z^n : n > 0\}$. Bepaal voor elk van de volgende woorden of deze tot de geconcateneerde taal $\mathcal{L}_1 \cdot \mathcal{L}_2$ behoort.

1. ε niet
2. $xyzxz$ niet
3. $xyyz$ wel
4. xyz wel

5. Wat is het verschil tussen de taal \emptyset en de taal $\{\varepsilon\}$? Gegeven een taal \mathcal{L} , wat zijn $\mathcal{L} \cdot \emptyset$ en $\mathcal{L} \cdot \{\varepsilon\}$?

De ledige taal \emptyset bevat geen enkel woord. De taal $\{\varepsilon\}$ bevat één woord, namelijk het unieke woord met lengte nul. Dat betekent dat $\mathcal{L} \cdot \emptyset = \emptyset$ en $\mathcal{L} \cdot \{\varepsilon\} = \mathcal{L}$.

* 6. Zijn de volgende eigenschappen (voor willekeurige talen A , B en C) waar of niet waar? Bewijs of geef een tegenvoorbeeld.

1. $(A \cup B)C = AC \cup BC$

$$2. (A \cap B)C = AC \cap BC$$

7. Geef voor de volgende talen een beknopte beschrijving.

1. $\{w \in \{0, 1\}^* : (\exists n \in \mathbb{N})(0 < |w| \leq n)\}$ is de verzameling $\{0, 1\}^* \setminus \{\varepsilon\}$ van alle niet-ledige woorden over het alfabet $\{0, 1\}$.
2. $\{w \in \{a, b\}^* : \text{exact één prefix van } w \text{ eindigt op een } b\}$ is de verzameling van alle woorden over het alfabet $\{a, b\}$ met exact één b , dus $\{a^m b a^n : m \geq 0, n \geq 0\}$.
3. $\{w \in \{a, b\}^* : \text{alle prefixen van } w \text{ eindigen op een } a\}$ is de ledige verzameling \emptyset , aangezien ε een prefix is van ieder woord en niet op a eindigt.

8. Zijn de volgende verzamelingen gesloten onder de gegeven operatie? Indien niet, geef de sluiting.

Merk op: de sluiting van een verzameling X onder f is gelijk aan $X \cup f(X) \cup f(f(X)) \cup \dots$

1. $\{a, b\}$ onder concatenatie is niet gesloten; de sluiting is $\{a, b\}^+$.
2. $\{a, b\}^*$ onder omkering is gesloten.
- * 3. $\{w \in \{a, b\}^* : w \text{ begint met } a\}$ onder omkering is niet gesloten; de sluiting is $\{w \in \{a, b\}^* : w \text{ begint of eindigt met } a\}$.

9. Geef voor de volgende talen een eindige deterministische automaat.

1. \emptyset met alfabet $\Sigma = \{0, 1\}$

We geven een automaat die altijd verwerpt.



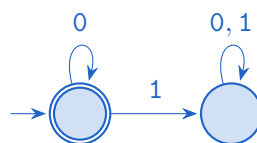
2. Σ^* met alfabet $\Sigma = \{a, b\}$

We geven een automaat die altijd aanvaardt.



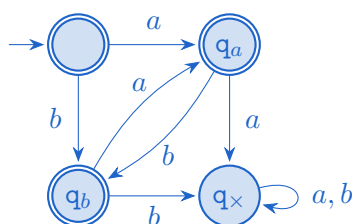
3. $\{w \in \{0, 1\}^+ : \text{geen enkele prefix van } w \text{ eindigt op een } 1\}$

“Geen enkele prefix eindigt op een 1” betekent eenvoudigweg dat de input geen symbolen 1 mag bevatten. Een eenvoudige automaat voor deze taal is dus:



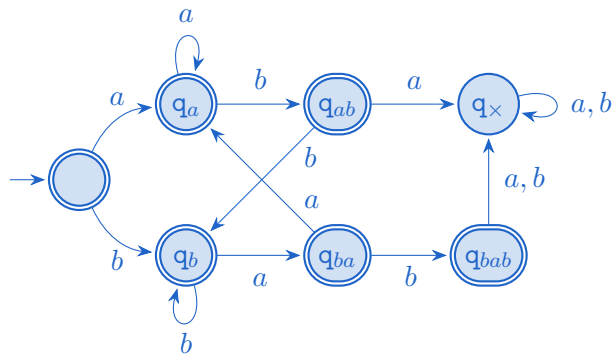
4. $\{w \in \{a, b\}^+ : w \text{ bestaat uit alternerende } a\text{'s en } b\text{'s}\}$

We houden de laatst gelezen letter bij in twee toestanden q_a en q_b . Als we ons in toestand q_a bevinden en nóg een a inlezen, dan moeten we de input verwerpen. Analoog met q_b .



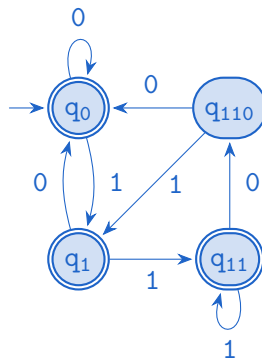
5. $\{w \in \{a, b\}^* : w \text{ bevat zowel } babb \text{ als } aba \text{ niet als deelwoord}\}$

We houden opnieuw in enkele toestanden de relevante laatst gelezen letters bij.



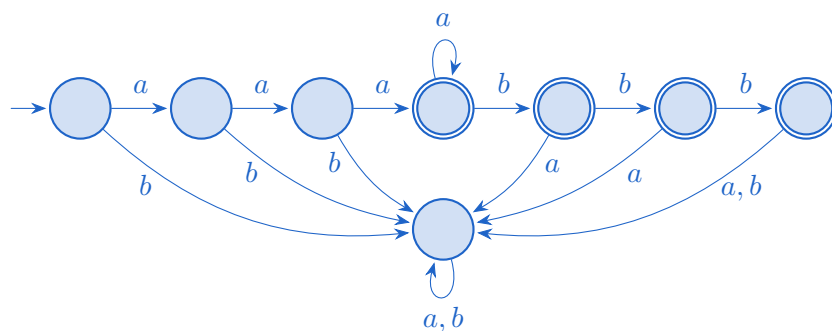
6. $\{w \in \{0, 1\}^* : w \text{ eindigt niet op } 110\}$

We houden opnieuw in enkele toestanden de relevante laatst gelezen letters bij.



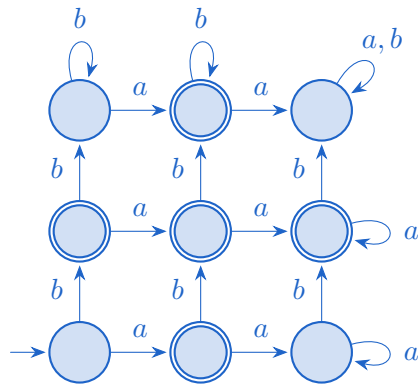
7. $\{a^m b^n : m \geq 3 \text{ en } n \leq 3\}$

We hoeven eerst slechts bij te houden of we nul, één, twee of minstens drie a 's lezen, en dan of we nul, één, twee of minstens drie b 's lezen.



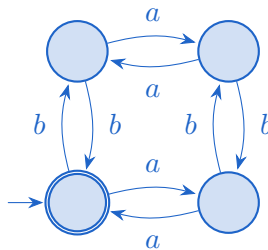
8. $\{w \in \{a, b\}^* : w \text{ bevat precies één } a \text{ of precies één } b\}$

We gebruiken negen toestanden, eentje voor elke combinatie uit $\{\text{nul, één of meerdere } a\}$ en $\{\text{nul, één of meerdere } b\}$.



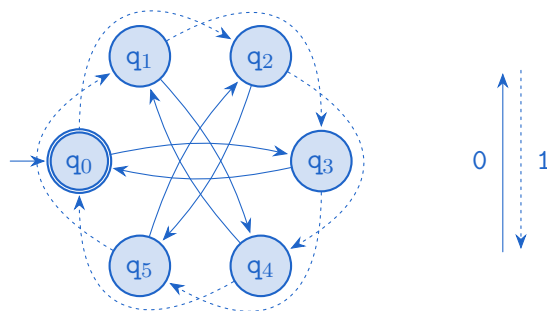
9. $\{w \in \{a, b\}^* : \text{zowel } \#_a(w) \text{ als } \#_b(w) \text{ zijn even}\}$

Deze verloopt vrij gelijkaardig als de vorige oefening, alleen gebruiken we nu combinaties uit {een even of oneven aantal a's} en {een even of oneven aantal b's} en kunnen we "terugkeren".



10. $\{w \in \{0, 1\}^* : 3 \#_0(w) + 2 \#_1(w) \text{ is deelbaar door zes}\}$

We houden met zes toestanden de waarde van $3 \#_0 + 2 \#_1$ van de reeds ingelezen prefix bij, modulo zes. Lezen we een 0, dan moet de waarde met drie verhogen (modulo zes), bij een 1 met twee. Uiteindelijk aanvaarden we als de waarde nul blijkt (modulo zes).

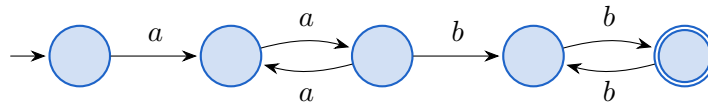


* 10. Geef voor de volgende talen een eindige deterministische automaat.

1. $\{w \in \{0, \dots, 9\}^* : w \text{ is de decimale representatie van een oneven getal}\}$
2. $\{w \in \{a, b\}^* : w \text{ bevat minstens drie } b\text{'s}\}$
3. $\{w \in \{a, b\}^* : w \text{ bevat zowel } ab \text{ als } bba \text{ niet als deelwoord}\}$
4. $\{w \in \{0, 1\}^* : w \text{ bevat de deelwoord } 101\}$
5. $\{w \in \{a, b\}^* : w \text{ bevat precies één } b\}$
6. $\{w \in \{a, b\}^* : w \text{ bevat minstens drie } a\text{'s niet onmiddellijk gevolgd door een } b\}$
7. $\{w \in \{0, 1\}^* : w \text{ bevat hoogstens één paar direct opeenvolgende nullen}\}$
8. $\{w \in \{a, b\}^* : w \text{ bevat minstens drie } b\text{'s en hoogstens drie } a\text{'s}\}$,

9. $\{w \in \{0, 1\}^* : w \text{ bevat de deelwoorden } 010 \text{ én } 101\}$,
10. $\{w \in \{0, 1\}^* : \text{de laatste en voorlaatste nul in } w \text{ staan opeenvolgend}\}$,
11. $\{a^m b a^n : m - n \text{ is deelbaar door } 3\}$
12. $\overline{\{a^m b^n : m > 4 \text{ en } n > 3\}}$

11. Welke taal wordt aanvaard door de volgende automaat?



Deze automaat aanvaardt de taal $\{a^m b^n : m \text{ en } n \text{ zijn even, met } m, n \geq 2\}$.

12. Bewijs dat de volgende taal regulier is:

$$\mathcal{L} = \{w \in \{0, 1\}^* : w \text{ is de binaire representatie van een getal deelbaar door drie}\}.$$

Hierbij stelt het lege woord het getal nul voor, en laten we toe dat w leidende nullen heeft.

Deze opgave bestaat uit twee onderdelen:

1. een eindige deterministische herkenner M voor \mathcal{L} geven;
2. bewijzen dat effectief $\mathcal{L}(M) = \mathcal{L}$.

Constructie van de eindige deterministische herkenner

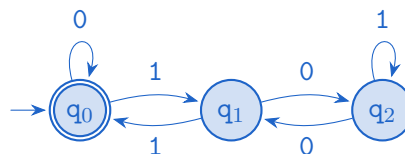
Idee: zij $w = x_1 \cdots x_n$ en noteer $[w]_2$ voor het getal voorgesteld door w . Gebruik drie toestanden, q_0, q_1 en q_2 , om bij te houden wat de rest van de reeds verwerkte invoer bij deling door drie is; als we in toestand q_k eindigen, dan betekent dit dat $[w]_2 \bmod 3 = k$.

Stel dat we reeds $x_1 \cdots x_i$ gelezen hebben en in toestand q_k zitten (zodat $[x_1 \cdots x_i]_2 \bmod 3 = k$). Lezen we dan x_{i+1} dan willen we naar die toestand $q_{k'}$ gaan zodat $[x_1 \cdots x_i \cdot x_{i+1}]_2 \bmod 3 = k'$.

Omdat $[x_1 \cdots x_i \cdot x_{i+1}]_2 = 2 \cdot [x_1 \cdots x_i]_2 + x_{i+1}$ moet de transitiefunctie dus zijn:

$$\delta(q_k, x) = q_{k'} \text{ met } k' = (2k + x) \bmod 3.$$

Zij dus $M = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_0\})$.



Bewijs dat $\mathcal{L} = \mathcal{L}(M)$

We bewijzen (voor $k \in \{0, 1, 2\}$) dat $[w]_2 \bmod 3 = k$ als en slechts als $(q_0, w) \vdash^* (q_k, \varepsilon)$.

We gebruiken inductie op $|w|$.

- Voor $|w| = 0$ is $w = \varepsilon$ en $[w]_2 = 0$, en inderdaad $(q_0, \varepsilon) \vdash^* (q_0, \varepsilon)$.
- Voor $|w| = n + 1$ is $w = vx$ met $v \in \{0, 1\}^*$, $|v| = n$ en $x \in \{0, 1\}$. Dan is $[w]_2 = 2[v]_2 + x$. Uit de inductiehypothese weten we dat $[v]_2 \bmod 3 = k$ als en slechts als $(q_0, v) \vdash^* (q_k, \varepsilon)$. Enerzijds is $(q_0, w) = (q_0, vx) \vdash^* (q_k, x) \vdash (q_{k'}, \varepsilon)$, met $k' = \delta(k, x) = (2k + x) \bmod 3$. Anderzijds is $[w]_2 \bmod 3 = (2[v]_2 + x) \bmod 3 = (2k + x) \bmod 3 = k' \bmod 3$. Dus ook voor woorden w met lengte $n + 1$ geldt de eigenschap dat $[w]_2 \bmod 3 = k'$ als en slechts als $(q_0, w) \vdash^* (q_{k'}, \varepsilon)$, en per inductie dus voor alle woorden.

⚡ 13. Veralgemeen oefening 12 naar de familie van talen

$$\mathcal{L}_{a,b,c} = \{w \in \{0, \dots, a-1\}^* : w \text{ stelt het getal } n \text{ in grondtal } a \text{ voor, met } n \equiv b \pmod{c}\}.$$

Reeks 2

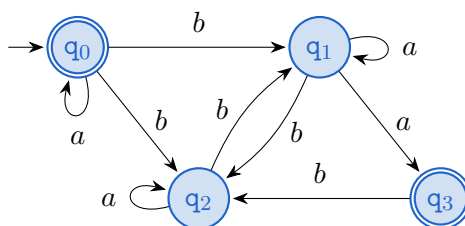
14. Bewijs: een taal \mathcal{L} is regulier als en slechts als het complement $\overline{\mathcal{L}}$ regulier is.

Zij $M = (K, \Sigma, \delta, s, A)$ een EDA met $\mathcal{L}(M) = \mathcal{L}$.

Dan is $\overline{M} = (K, \Sigma, \delta, s, K \setminus A)$ een EDA met $\mathcal{L}(\overline{M}) = \overline{\mathcal{L}}$:

$$\begin{aligned} w \in \mathcal{L}(\overline{M}) &\Leftrightarrow (s, w) \vdash_M^* (q, \varepsilon) \text{ met } q \in K \setminus A \\ &\Leftrightarrow (s, w) \vdash_M^* (q, \varepsilon) \text{ met } q \notin A \\ &\Leftrightarrow w \notin \mathcal{L}(M) \\ &\Leftrightarrow w \in \overline{\mathcal{L}}. \end{aligned}$$

* 15. Beschouw de volgende niet-deterministische automaat M .

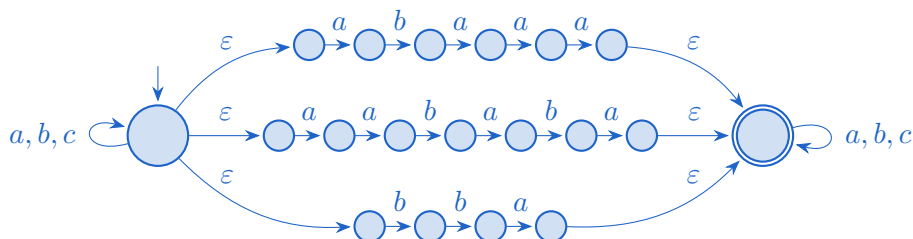


Bepaal of $abab$, $ababa$ en $ababab$ in $\mathcal{L}(M)$ zitten.

16. Construeer een niet-deterministische automaat voor de volgende talen.

- $\{w \in \{a, b, c\}^* : w \text{ bevat minstens één van de deelwoorden } abaaa, aababa \text{ of } bba\}$

De automaat kan “raden” wanneer we beginnen testen op één van de gezochte deelwoorden, en welk deelwoord we precies zoeken.

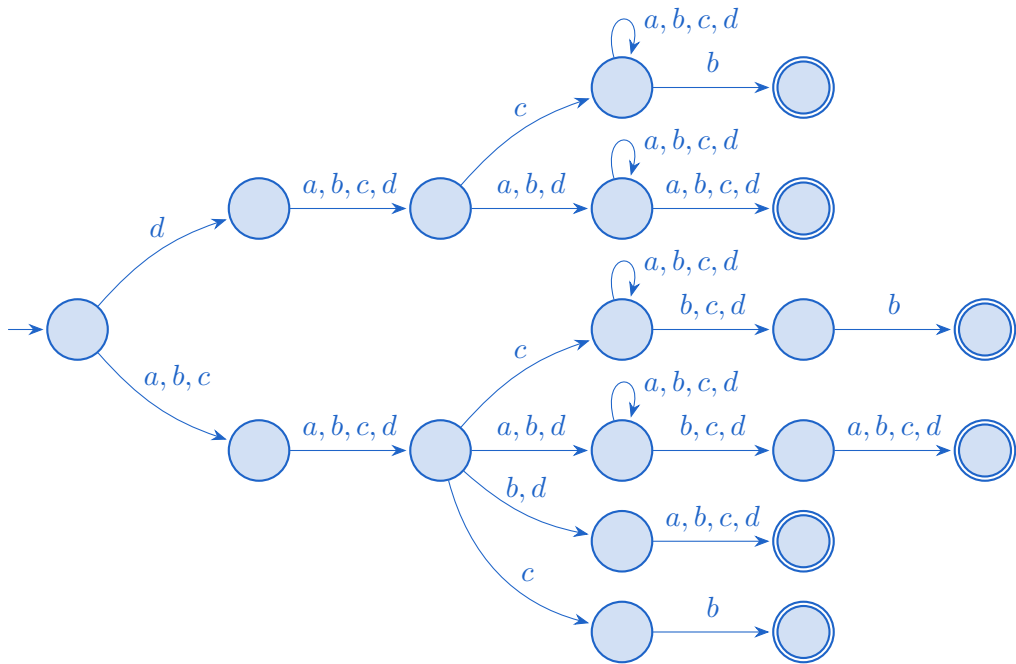


- de taal van alle $w \in \{a, b, c, d\}^*$ waarvoor

- $|w| \geq 4$,
- als de derde letter van w een c is, dan is de laatste letter van w een b , en
- als de voorlaatste letter van w een a is, dan is de eerste letter een d

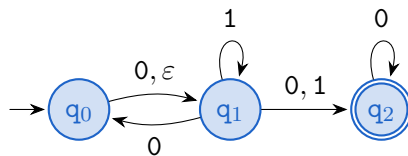
We houden de eerste en derde gelezen letter bij (in enkele vertakkingen). De automaat gokt wanneer de voorlaatste letter verwerkt wordt, en controleert dan of deze en de laatste letter compatibel zijn met de voorwaarden op de taal.

Er treedt nog een lastigheid op als we een woord met precies vier letters te verwerken krijgen, want daar zijn de derde en de voorlaatste letter gelijk; deze moeten we apart behandelen.



- * 3. $\{w \in \{0, 1\}^* : \text{de tweede letter van } w \text{ is een } 0 \text{ en de voorlaatste letter een } 1\}$
- * 4. de taal van alle $w \in \{x, y\}^*$ die minstens één deelstring bevatten van drie identieke letters
- * 5. de taal van alle binaire voorstellingen in $\{0, 1\}^*$ die een deelwoord bevatten dat een veelvoud van vijf voorstelt (verschillend van nul)

17. Construeer een equivalente eindige deterministische automaat voor deze NDA.



We passen de constructie in stelling 2.2 uit de cursus toe, en starten met

$$\text{eps}(q_0) = \{q_0, q_1\}, \quad \text{eps}(q_1) = \{q_1\}, \quad \text{eps}(q_2) = \{q_2\}.$$

De nieuwe starttoestand s wordt dus $\text{eps}(q_0) = \{q_0, q_1\}$.

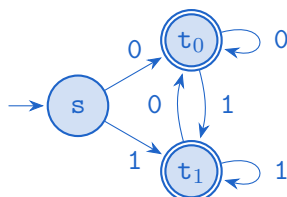
Hoe moeten we $\delta(s, 0)$ definiëren? Toestand $q_0 \in s$ kan onder 0 naar q_1 , en $q_1 \in s$ naar q_0 en q_2 . Met andere woorden, stel $t_0 = \text{eps}(q_0) \cup \text{eps}(q_1) \cup \text{eps}(q_2) = \{q_0, q_1, q_2\}$ en dan is $\delta(s, 0) = t_0$.

Op dezelfde manier zien we dat $q_0 \in s$ onder 1 naar geen enkele toestand kan, en $q_1 \in s$ naar q_1 en q_2 . Noem dus $t_1 = \text{eps}(q_1) \cup \text{eps}(q_2) = \{q_1, q_2\}$ en dan is $\delta(s, 1) = t_1$.

Doen we hetzelfde voor t_0 en t_1 , dan blijkt dat we geen extra toestanden moeten creëren, daar

$$\delta(t_0, 0) = t_0, \quad \delta(t_0, 1) = t_1, \quad \delta(t_1, 0) = t_0, \quad \delta(t_1, 1) = t_1.$$

De aanvaardende toestanden zijn alle toestanden die q_2 bevatten, dus t_0 en t_1 .

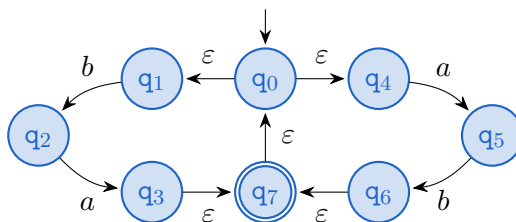


Merk op dat deze EDA de volledige taal $\{0, 1\}^+$ aanvaardt.

Merk op dat we hier een subtiliteit onder de mat hebben geveegd. Vanuit q_0 vertrekt er namelijk geen enkele transitie met label 1. Hieronder wordt verstaan dat er een transitie met label 1 loopt naar een aparte “reject”-toestand q' waar de rest van het woord wordt verwerkt alvorens te verwerpen (ook al kan de machine in feite toch verder, dankzij de ε -transitie). Dit vormt voor de oefening geen probleem, omdat het niet uitmaakt op hoeveel en welke manieren een woord verworpen kan worden—het enige wat telt, is of een woord aanvaard kan worden, en daarop heeft q' geen effect.

Ga zelf na wat de deterministische automaat wordt als we q' er expliciet bij beschouwen, en toon aan dat deze inderdaad equivalent is met de EDA die wij opgesteld hebben.

* 18. Idem voor deze NDA.



19. Bewijs: \mathcal{L} is regulier als en slechts als \mathcal{L}^R regulier is.

Zij $M = (K, \Sigma, \delta, s, A)$ een EDA zodat $\mathcal{L}(M) = \mathcal{L}$.

Constructie van de EDA M^R zodat $\mathcal{L}(M^R) = \mathcal{L}^R$

\mathcal{L}^R bestaat uit dezelfde woorden als \mathcal{L} maar dan omgekeerd. Onze nieuwe machine zal in essentie dezelfde zijn als M behalve dat de pijlen omgedraaid worden, de starttoestand van M aanvaardend wordt, en de aanvaardende toestanden van M starttoestanden. Als $\delta(q, a) = q'$, tekenen we dus een pijl van q' naar q met label a ; m.a.w. $((q', a), q)$ wordt een element van de transitierelatie.

Merk op dat alle aanvaardende toestanden van M zo starttoestanden voor M^R worden; er is dus geen unieke starttoestand meer! Voeg daarom een nieuwe toestand toe met ε -transities naar alle starttoestanden, en laat deze nieuwe toestand de unieke starttoestand zijn. Noteer die met s' .

Merk ook op dat deze nieuwe machine M^R niet-deterministisch is.

Bewijs dat $\mathcal{L}(M^R) = \mathcal{L}^R$

Eerst tonen we aan dat $\mathcal{L}^R \subseteq \mathcal{L}(M^R)$. Beschouw daarvoor een woord $w = x_n \cdots x_1 \in \mathcal{L}^R$, zodat $w^R = x_1 \cdots x_n \in \mathcal{L} = \mathcal{L}(M)$. Dat betekent dat $(s, w^R) \vdash_M^* (q, \varepsilon)$ voor zekere $q \in A$, of voluit

$$(s = q_0, x_1 x_2 \cdots x_n) \vdash_M (q_1, x_2 \cdots x_n) \vdash_M \cdots \vdash_M (q_{n-1}, x_n) \vdash_M (q_n = q, \varepsilon)$$

voor zekere $q_i \in K$. Voor iedere index i is er dus een pijl in M van toestand q_i naar toestand q_{i+1} met label x_{i+1} . Hierdoor geldt dat

$$\begin{aligned} (s', w) = (s', x_n \cdots x_1) & \vdash_{M^R} (q = q_n, x_n \cdots x_1) \\ & \vdash_{M^R} (q_{n-1}, x_{n-1} \cdots x_1) \\ & \vdots \\ & \vdash_{M^R} (q_1, x_1) \\ & \vdash_{M^R} (s = q_0, \varepsilon), \end{aligned}$$

en omdat s aanvaardend is in M^R , is inderdaad $w \in \mathcal{L}(M^R)$.

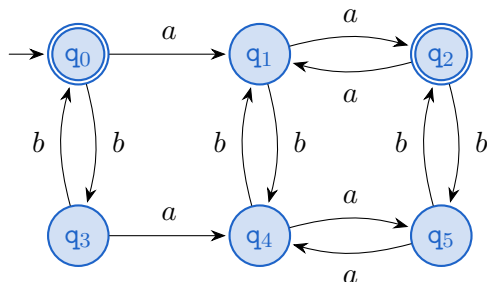
Vervolgens tonen we aan dat $\mathcal{L}(M^R) \subseteq \mathcal{L}^R$. Beschouw daarvoor $w \in \mathcal{L}(M^R)$. Dan geldt er dat $(s', w) \vdash_{M^R}^* (s, \varepsilon)$. Per constructie van s' bestaat er een aanvaardende toestand q van M zodanig dat $(s', w) \vdash_{M^R} (q, w)$ via een ε -transitie, waarna $(q, w) \vdash_{M^R}^* (s, \varepsilon)$. In deze laatste berekening komen we geen ε -transities meer tegen, dus er volgt (analoog aan de vorige stap met de q_i 's) dat $(s, w^R) \vdash_M^* (q, \varepsilon)$ en dus dat $w^R \in \mathcal{L}(M) = \mathcal{L}$, zodat inderdaad $w \in \mathcal{L}^R$.

- ⚡ 20. In de theorie hebben we een proces besproken om een niet-deterministische automaat om te zetten naar een deterministische automaat voor dezelfde taal. Het nadeel van de deterministische variant is dat het aantal toestanden exponentieel toeneemt, van k naar 2^k . Deze oefening toont aan dat we dit niet kunnen vermijden: er bestaan talen (in functie van een parameter k) waarvoor een EDA $\Theta(2^k)$ toestanden nodig heeft, terwijl $\Theta(k)$ toestanden volstaan voor een equivalente NDA.

Beschouw het alfabet $\Sigma_k = \{1, \dots, k\}$ en de taal \mathcal{L}_k bestaande uit alle woorden $w \in \Sigma_k^*$ waarin minstens één letter uit Σ_k ontbreekt. Construeer voor deze taal een niet-deterministische automaat met $k + 2$ toestanden. Toon vervolgens aan dat elke deterministische automaat voor \mathcal{L}_k minstens 2^k toestanden moet hebben. *Hint: beschouw de relatie $\approx_{\mathcal{L}}$ en pas stelling 3.2 uit de cursus toe.*

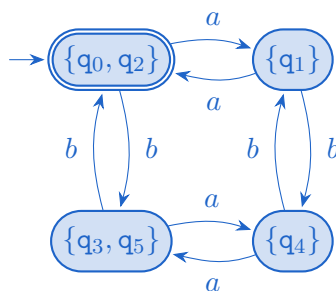
Reeks 3

21. Minimaliseer het aantal toestanden in de volgende automaat.



We passen de constructie uit hoofdstuk 3.3 toe, met de equivalentierelaties \equiv_n . Intuïtief betekent $q_i \equiv_n q_j$ dat de toestanden hetzelfde gedrag vertonen voor alle woorden met lengte hoogstens n ; elk zo'n woord komt vanuit q_i in een aanvaardende toestand terecht als en slechts als dat het geval is vanuit q_j . Inductief is $q_i \equiv_0 q_j$ als en slechts als q_i en q_j beide aanvaardend of beide verwerpend zijn, en dat $q_i \equiv_{n+1} q_j$ als en slechts als $q_i \equiv_n q_j$ en $\delta(q_i, x) \equiv_n \delta(q_j, x)$ voor alle $x \in \Sigma$.

Voor \equiv_0 zijn de equivalentieklassen $\{q_0, q_2\}$ en $\{q_1, q_3, q_4, q_5\}$. Voor \equiv_1 zijn de equivalentieklassen precies $\{q_0, q_2\}$, $\{q_1\}$, $\{q_3, q_5\}$ en $\{q_4\}$. Voor \equiv_2 blijken de equivalentieklassen precies dezelfde als die van \equiv_1 , dus de constructie loopt ten einde en we eindigen met vier toestanden.



22. Geef voor de volgende talen een reguliere uitdrukking.

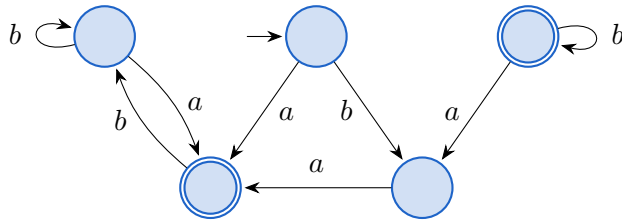
1. $\{w \in \{a, b\}^* : \text{iedere } a \text{ in } w \text{ wordt direct voorgedaan door een } b\}$ wordt $b^*(bab^*)^*$
2. $\{w \in \{a, b\}^* : w \text{ heeft even lengte}\}$ wordt $(aa \cup ab \cup ba \cup bb)^*$
3. $\{w \in \{a, b\}^* : \text{het aantal } a\text{'s in } w \text{ is oneven}\}$ wordt $b^*ab^*(b^*ab^*ab^*)^*$
4. $\{w \in \{a, b\}^* : w \text{ bevat } aa \text{ als deelwoord}\}$ wordt $(a \cup b)^*aa(a \cup b)^*$
5. $\{w \in \{a, b\}^* : w \text{ bevat hoogstens drie } a\text{'s}\}$ wordt $b^*(a \cup \varepsilon)b^*(a \cup \varepsilon)b^*(a \cup \varepsilon)b^*$
6. $\{a^n b^n : n \leq 3\}$ wordt $\varepsilon \cup ab \cup aabb \cup aaabbb$

* 7. de taal uit reeks 1, oefening 12

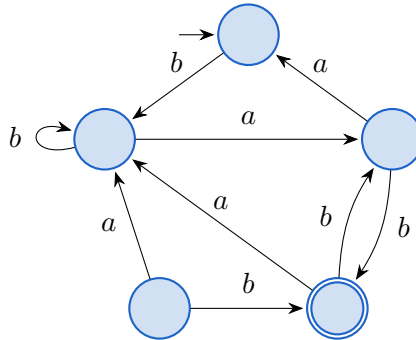
23. Construeer voor volgende reguliere uitdrukkingen een eindige automaat en verklaar in woorden hoe de corresponderende reguliere taal eruit ziet. Volg stelling 4.1 uit de cursus.

1. $(b(b \cup \varepsilon)b)^*$
2. $(a \cup b)^+(ba \cup bba)^*$
- * 3. $(b \cup a)((b \cup \varepsilon)a)^+$
- * 4. $(ab \cup ba)^* \cup (a^*)^*$

24. Construeer voor de volgende deterministische automaat een reguliere uitdrukking.



* 25. Construeer voor de volgende deterministische automaat een reguliere uitdrukking.



* 26. Bewijs met behulp van reguliere uitdrukkingen dat \mathcal{L} regulier is als en slechts als \mathcal{L}^R regulier is. *Hint: beschouw een reguliere uitdrukking ρ zodat $\mathcal{L} = \mathcal{L}(\rho)$. Bewijs dat $\mathcal{L}^R = \mathcal{L}(\rho^R)$ via inductie op de complexiteit van ρ .*

27. Geef voor de volgende talen een reguliere grammatica.

1. $\mathcal{L}((a \cup b)^* bba)$

We zorgen er eerst voor dat het startsymbool S het gedeelte $(a \cup b)^*$ genereert. Daarna gaan we naar andere niet-terminalen A en B die het suffix bba genereren.

$$\begin{cases} S \rightarrow aS \mid bS \mid bA, \\ A \rightarrow bB, \\ B \rightarrow a. \end{cases}$$

2. $\{w \in \{a, b\}^* : w \text{ bevat de deelstring } abb\}$

S genereert de letters vóór abb en T de letters ná abb .

$$\begin{cases} S \rightarrow aS \mid bS \mid aA, \\ A \rightarrow bB, \\ B \rightarrow bT, \\ T \rightarrow aT \mid bT \mid \varepsilon. \end{cases}$$

3. $\{w \in \{0, 1\}^* : w \text{ eindigt niet op } 01\}$

- S betekent “de laatste letters zijn 11 (of initieel 1 of ε)”,
- T betekent “de laatste letter is 0”,
- U betekent “de laatste letters zijn 01”.

$$\begin{cases} S \rightarrow 0T \mid 1S \mid \varepsilon, \\ T \rightarrow 0T \mid 1U \mid \varepsilon, \\ U \rightarrow 0T \mid 1S. \end{cases}$$

4. $\{w \in \{a, b\}^* : \text{als } w \text{ de deelstring } aa \text{ bevat, dan is } |w| \text{ oneven}\}$

- S betekent “nog geen aa gegenereerd, voorlopige lengte even, laatste letter b ”,
- S' betekent “nog geen aa gegenereerd, voorlopige lengte oneven, laatste letter b ”,
- T betekent “nog geen aa gegenereerd, voorlopige lengte even, laatste letter a ”,
- T' betekent “nog geen aa gegenereerd, voorlopige lengte oneven, laatste letter a ”,
- U betekent “reeds aa gegenereerd, voorlopige lengte even”,
- U' betekent “reeds aa gegenereerd, voorlopige lengte oneven”.

$$\begin{cases} S \rightarrow aT' \mid bS' \mid \varepsilon, \\ S' \rightarrow aT \mid bS \mid \varepsilon, \\ T \rightarrow aU' \mid bS' \mid \varepsilon, \\ T' \rightarrow aU \mid bS \mid \varepsilon, \\ U \rightarrow aU' \mid bU', \\ U' \rightarrow aU \mid bU \mid \varepsilon. \end{cases}$$

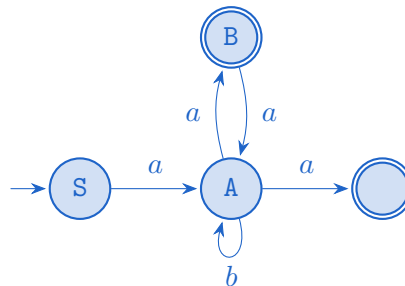
* 28. Geef voor de volgende talen een reguliere grammatica.

1. \emptyset
2. $\{\varepsilon\}$
3. $\mathcal{L}((a \cup b)(ba \cup bba)^*)$
4. $\{a^n b^n : 0 \leq n \leq 3\}$
5. $\{w \in \{a, b\}^* : w \text{ heeft even lengte}\}$
6. $\{w \in \{a, b\}^* : w \text{ bevat hoogstens drie } a\text{'s}\}$
7. $\{w \in \{a, b\}^* : w \text{ bevat een oneven aantal } a\text{'s}\}$

29. Construeer voor de volgende reguliere grammatica een eindige deterministische automaat.

$$\begin{cases} S \rightarrow aA, \\ A \rightarrow bA \mid aB \mid a, \\ B \rightarrow aA \mid \varepsilon. \end{cases}$$

De grammatica omzetten naar een *niet-deterministische* automaat is niet zo moeilijk; zie hieronder. Daarna moet deze nog deterministisch gemaakt worden met de constructie uit de theorie.



30. Gegeven reguliere grammatica's voor de talen \mathcal{L}_1 en \mathcal{L}_2 . Construeer een reguliere grammatica voor de reguliere taal $\mathcal{L}_1 \cdot \mathcal{L}_2$.

Zij S_1 en S_2 de startsymbolen van de respectievelijke grammatica's.

Na een eindig aantal stappen in de eerste grammatica hebben we steeds een woord van de vorm $x_1 \cdots x_n T$, met T een niet-terminaal, en we stoppen pas als de laatste regel van de vorm $T \rightarrow x$ of $T \rightarrow \varepsilon$ is. Als dit gebeurt, willen we overgaan naar de starttoestand van de tweede grammatica.

Regels van de vorm $T \rightarrow x$ worden eenvoudig aangepast: vervang deze gewoon door $T \rightarrow xS_2$.

Regels van de vorm $T \rightarrow \varepsilon$ zijn iets subtieler; we kunnen deze namelijk niet zomaar vervangen door $T \rightarrow S_2$, want dit is geen geldige herschrijfgregel in een reguliere grammatica. We kunnen dit als volgt oplossen. Verwijder de regel $T \rightarrow \varepsilon$, tenzij $S_2 \rightarrow \varepsilon$ een regel is voor de grammatica voor \mathcal{L}_2 . Voeg daarna regels $T \rightarrow y$ toe voor elke regel van de vorm $S_2 \rightarrow y$, en regels $T \rightarrow zU$ voor elke regel van de vorm $S_2 \rightarrow zU$.

- * 31. Gegeven reguliere grammatica's voor de talen \mathcal{L}_1 en \mathcal{L}_2 . Construeer een reguliere grammatica voor de reguliere taal $\mathcal{L}_1 \cup \mathcal{L}_2$.

- 32. Gegeven een reguliere grammatica voor de taal \mathcal{L} . Construeer een reguliere grammatica voor \mathcal{L}^* .

Zij S het startsymbool van de respectievelijke grammatica.

Definieer een nieuw startsymbool S' met de herschrijfgregel $S' \rightarrow \varepsilon$. De regel $S' \rightarrow S$ is opnieuw niet toegestaan; gebruik hetzelfde trucje als in oefening 30 om die legitiem te maken. Vervolgens willen we voor elke regel van de vorm $T \rightarrow \varepsilon$ of $T \rightarrow x$, de regel $T \rightarrow S$ of $T \rightarrow xS$ toevoegen, alleen moeten we nogmaals hetzelfde trucje toepassen om $T \rightarrow S$ legitiem te maken.

- * 33. Vervolledig het bewijs van stelling 5.1 in de theorie: geef een constructie om een EDA om te zetten naar een reguliere grammatica die dezelfde taal genereert.
- * 34. Omschrijf een algoritme om, gegeven een reguliere grammatica voor \mathcal{L} , een reguliere grammatica te construeren voor \mathcal{L}^R .
- * 35. Bewijs dat een grammatica met herschrijfgregels van de vorm $A \rightarrow wB$, met $w \in \Sigma^*$, nog steeds een reguliere taal genereert.

Reeks 4

36. Zijn de volgende talen regulier of niet? Toon aan.

1. $\{a^m b^n : m - n = 3\}$

Deze taal \mathcal{L} is gelijk aan $\{a^{n+3} b^n : n \geq 3\}$, waarin een afhankelijkheid tussen het aantal a 's en b 's optreedt die een onbeperkt geheugen vereist om bij te houden. We verwachten dus dat \mathcal{L} niet regulier zal zijn, en passen het principe van het *pumping lemma* toe.

Stel dat \mathcal{L} regulier is. Het is duidelijk dat \mathcal{L} oneindig groot is. Volgens het pumping lemma bestaat er een constante k zodat voor ieder woord $w \in \mathcal{L}$ met $|w| \geq k$, er een factorisatie $w = xyz$ bestaat met $|xy| \leq k$ en $|y| \neq 0$ waarvoor $w_p = xy^p z \in \mathcal{L}$ voor elke $p \in \mathbb{N}$.

We passen deze eigenschap toe op het concrete woord $w = a^{k+3} b^k \in \mathcal{L}$, waarvoor inderdaad $|w| = 2k + 3 \geq k$. Er geldt dus dat $a^{k+3} b^k = xyz$ met $|xy| \leq k$ en $|y| \neq 0$. Merk op dat xy , als prefix van $a^{k+3} b^k$ met lengte hoogstens k , volledig uit a 's moet bestaan! Dus $x = a^{|x|}$ en $y = a^{|y|}$ (met $|x| \geq 0$ en $|y| > 0$), zodat $z = a^{k+3-|x|-|y|} b^k$.

Beschouw nu de woorden w_p uit het pumping lemma. Voorgaande analyse leert dat

$$w_p = x \cdot y^p \cdot z = a^{|x|} \cdot (a^{|y|})^p \cdot a^{k+3-|x|-|y|} b^k = a^{(p-1) \cdot |y| + k + 3} b^k,$$

terwijl w_p voor alle $p \in \mathbb{N}$ tot \mathcal{L} zou moeten behoren, zodat $((p-1) \cdot |y| + k + 3) - k = 3$. Deze vergelijking wordt echter alleen voldaan voor $p = 1$; een strijdigheid! Daardoor kan \mathcal{L} niet regulier zijn; woorden kunnen niet zomaar op- of neergepompt worden.

2. $\{a^m b^n : m \leq n\}$

Deze taal is niet regulier; pas het pumping lemma opwaarts toe op het woord $w = a^k b^k$.

3. $\{w \in \{a\}^* \{b, c, d\}^* : \#_b(w) \text{ is een veelvoud van zes en } \#_b(w) \geq 3 \#_c(w)\}$

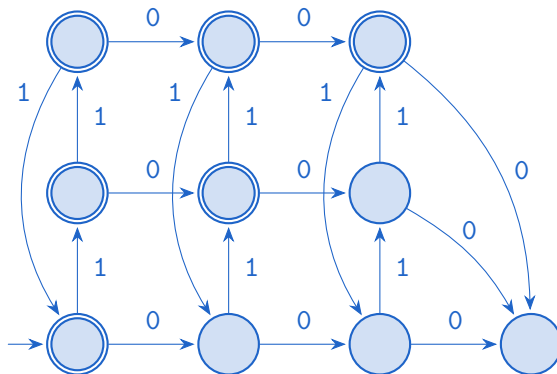
Deze taal \mathcal{L} is niet regulier. Deze opgave wordt het efficiëntst bewezen door gebruik te maken van de sluitingseigenschappen van reguliere talen. Veronderstel immers dat \mathcal{L} regulier is, dan ook de doorsnede met de reguliere taal $b^* c^*$. Nu geldt $\mathcal{L} \cap b^* c^* = \{b^{6m} c^n : 2m \geq n \geq 0\}$. Het pumping lemma neerwaarts toegepast op $w = b^{6k} c^{3k}$ leidt tot een strijdigheid.

4. \mathcal{L}^* waarbij $\mathcal{L} = \{ww^R : w \in \{a, b, c, d\}^*, |w| \leq 2018\}$

Deze taal is regulier. De taal \mathcal{L} is immers eindig, dus zonder twijfel regulier; uit de sluitingseigenschappen weten we dat dan ook de Kleene-ster van \mathcal{L} regulier is.

5. $\{w \in \{0, 1\}^* : \#_0(w) \leq (\#_1(w) \bmod 3)\}$

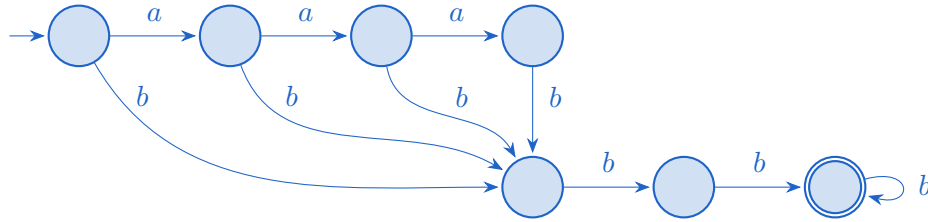
Deze taal is regulier. We moeten immers niet het volledige aantal nullen en enen bijhouden: omdat $(\#_1(w) \bmod 3) \leq 2$, kunnen we woorden met drie of meer nullen zeker verwerpen.



Horizontaal houden we het aantal gelezen nullen bij, verticaal het aantal enen modulo drie.

6. $\{a^m b^n : m \leq 3 \leq n\}$

Deze taal is regulier. Het enige wat bijgehouden hoeft te worden is of we eerst hoogstens drie a 's lezen, en daarna minstens drie b 's—er is geen verdere afhankelijkheid tussen de aantallen a 's en b 's.



* 37. Zijn de volgende talen regulier of niet? Toon aan.

1. $\{w \in \{a, b\}^* : \#_a(w) \neq \#_b(w)\}$
2. $\{w \in \{a\}^* \{b, c, d\}^* : \#_a(w) \text{ is deelbaar door acht en } \#_b(w) \leq 2 \#_c(w)\}$
3. $\{a^m b^n : m + n = 3\}$
4. $\{a^n b^n (bc)^m : n \geq 0, m \geq 0\}$
5. \mathcal{L}^R met $\mathcal{L} = \{w \in \{a, b, c\}^* : \#_a(w) + \#_b(w) > \#_c(w) \text{ en } w \text{ bevat geen deelwoord } aa\}$
6. $\{(ab)^n : n \geq 0\}$
7. $\{(ab)^n (cd)^n : n \geq 0\}$
8. $\{w \in \{a, b\}^* : w = w^R\}$
9. $\{w w^R w : w \in \{a, b\}^+\}$

⚡ 10. $\mathcal{L} \cdot \mathcal{L}$ waarbij $\mathcal{L} = \{w \in \{a, b\}^* : \#_a(w) \neq \#_b(w)\}$

38. Zijn de reguliere talen gesloten onder de volgende operaties? Zo ja, toon aan met een constructie⁽¹⁾. Zo nee, geef een tegenvoorbeeld.

1. $\text{Intersection}(\mathcal{L}_1, \mathcal{L}_2) = \mathcal{L}_1 \cap \mathcal{L}_2$

Gesloten.

Zij $M_1 = (K_1, \Sigma, \delta_1, \mathbf{s}_1, A_1)$ en $M_2 = (K_2, \Sigma, \delta_2, \mathbf{s}_2, A_2)$ twee EDA's met $\mathcal{L}(M_1) = \mathcal{L}_1$ en $\mathcal{L}(M_2) = \mathcal{L}_2$. We construeren een nieuwe automaat M' met $\mathcal{L}(M') = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$.

Stel $K' = K_1 \times K_2$, $\mathbf{s}' = (\mathbf{s}_1, \mathbf{s}_2)$ en $A' = A_1 \times A_2$. Het idee is om in de nieuwe automaat de twee oude tegelijk te simuleren, zodat concreet $(\mathbf{s}', w) = ((\mathbf{s}_1, \mathbf{s}_2), w) \vdash_{M'}^* ((\mathbf{p}_1, \mathbf{p}_2), \varepsilon)$ als en slechts als $(\mathbf{s}_1, w) \vdash_{M_1}^* (\mathbf{p}_1, \varepsilon)$ en $(\mathbf{s}_2, w) \vdash_{M_2}^* (\mathbf{p}_2, \varepsilon)$. Hiertoe definiëren we

$$\delta'((\mathbf{p}_1, \mathbf{p}_2), a) = (\delta_1(\mathbf{p}_1, a), \delta_2(\mathbf{p}_2, a)).$$

Herkent deze machine $M' = (K', \Sigma, \delta', \mathbf{s}', A')$ precies de taal $\mathcal{L}(M_1) \cap \mathcal{L}(M_2)$?

Toon eerst via inductie op de lengte $|w|$ aan dat $((\mathbf{s}_1, \mathbf{s}_2), w) \vdash_{M'}^* ((\mathbf{p}_1, \mathbf{p}_2), \varepsilon)$ als en slechts als $(\mathbf{s}_1, w) \vdash_{M_1}^* (\mathbf{p}_1, \varepsilon)$ en $(\mathbf{s}_2, w) \vdash_{M_2}^* (\mathbf{p}_2, \varepsilon)$ —werk dit zelf uit. Daarna kunnen we besluiten dat $\mathcal{L}(M') = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$:

$$\begin{aligned} w \in \mathcal{L}(M_1) \cap \mathcal{L}(M_2) &\Leftrightarrow w \in \mathcal{L}(M_1) \text{ en } w \in \mathcal{L}(M_2) \\ &\Leftrightarrow \begin{cases} (\mathbf{s}_1, w) \vdash_{M_1}^* (\mathbf{p}_1, \varepsilon) \text{ met } \mathbf{p}_1 \in A_1 \\ (\mathbf{s}_2, w) \vdash_{M_2}^* (\mathbf{p}_2, \varepsilon) \text{ met } \mathbf{p}_2 \in A_2 \end{cases} \\ &\Leftrightarrow ((\mathbf{s}_1, \mathbf{s}_2), w) \vdash_{M'}^* ((\mathbf{p}_1, \mathbf{p}_2), \varepsilon) \text{ met } (\mathbf{p}_1, \mathbf{p}_2) \in A_1 \times A_2 = A' \\ &\Leftrightarrow w \in \mathcal{L}(M'). \end{aligned}$$

⁽¹⁾Met andere woorden, ontwerp een algoritme dat een EDA, reguliere uitdrukking of reguliere grammatica voor \mathcal{L} omzet in een EDA, reguliere uitdrukking of reguliere grammatica voor $f(\mathcal{L})$.

$$2. \text{Join}(\mathcal{L}_1, \mathcal{L}_2) = \{xy : x \in \mathcal{L}_1, y \in \mathcal{L}_2 \text{ en } |x| = |y|\}$$

Niet gesloten.

Een tegenvoorbeeld is $\text{Join}(a^*, b^*) = \{a^n b^n : n \geq 0\}$.

$$3. \text{Permute}(\mathcal{L}) = \{x \in \Sigma^* : (\exists y \in \mathcal{L})(x \text{ is een permutatie van } y)\}$$

Niet gesloten.

Een tegenvoorbeeld is $\text{Permute}((ab)^*) = \{w \in \{a, b\}^* : \#_a(w) = \#_b(w)\}$.

$$4. \text{MaxString}(\mathcal{L}) = \{x \in \mathcal{L} : (\forall y \in \Sigma^+)(xy \notin \mathcal{L})\}$$

Gesloten.

Zij $M = (K, \Sigma, \delta, s, A)$ een EDA met $\mathcal{L}(M) = \mathcal{L}$. We construeren een nieuwe automaat M' met $\mathcal{L}(M') = \text{MaxString}(\mathcal{L})$.

Zij A' de verzameling van alle aanvaardende toestanden waaruit geen enkel pad vertrekt in M' naar een aanvaardende toestand (ook geen lus), dus

$$A' = \{p \in A : \text{voor elke } w \in \Sigma^* \text{ met } 1 \leq |w| \leq |K| \text{ en } (p, w) \vdash_M^* (q, \varepsilon), \text{ geldt dat } q \notin A\}.$$

We tonen aan dat $M' = (K, \Sigma, \delta, s, A')$ voldoet.

$$\underline{\mathcal{L}(M') \subseteq \text{MaxString}(\mathcal{L})}$$

Onderstel dat $w \in \mathcal{L}(M')$. Dan is $(s, w) \vdash_{M'}^* (p, \varepsilon)$ met $p \in A' \subseteq A$, zodat zeker ook $w \in \mathcal{L}$. Neem nu aan dat $w \notin \text{MaxString}(\mathcal{L})$; we zoeken een strijdigheid.

Omwille van de assumptie $w \notin \text{MaxString}(\mathcal{L})$ bestaat er een $y \in \Sigma^+$ met $wy \in \mathcal{L} = \mathcal{L}(M)$. Dit betekent dat er toestanden q_1 en q_2 bestaan zodat $(s, wy) \vdash_M^* (q_1, y) \vdash_M^* (q_2, \varepsilon)$, met $q_2 \in A$. Nu heeft M' dezelfde transitiefunctie als M , dus geldt ook dat

$$(s, wy) \vdash_{M'}^* (q_1, y) \vdash_{M'}^* (q_2, \varepsilon).$$

We weten echter dat $(s, w) \vdash_{M'}^* (p, \varepsilon)$, waaruit volgt dat $q_1 = p \in A'$. Maar dan zal het feit dat $(q_1, y) \vdash_M^* (q_2, \varepsilon)$ met $q_2 \in A$, tot een strijdigheid leiden met de definitie van A' . Als $|y| > |K|$, dan zit er een lus in deze berekening; door de lussen eruit te halen, kunnen we bekomen dat $|y| \leq |K|$. Hieruit zou volgen dat $q_1 = p \notin A'$; een tegenstrijdigheid.

$$\underline{\text{MaxString}(\mathcal{L}) \subseteq \mathcal{L}(M')}$$

Onderstel dat $w \in \text{MaxString}(\mathcal{L})$. Dan is $w \in \mathcal{L}$, dus $(s, w) \vdash_M^* (q_1, \varepsilon)$ met $q_1 \in A$. We weten dat $wy \notin \mathcal{L}$ voor alle $y \in \Sigma^+$. Hieruit volgt dat voor alle $y \in \Sigma^+$

$$(s, wy) \vdash_M^* (q_1, y) \vdash_M^* (q_2, \varepsilon),$$

met $q_2 \notin A$. Dit betekent per definitie van A' dat de toestand $q_1 \in A'$. In M' vinden we dus dat $(s, w) \vdash_{M'}^* (q_1, \varepsilon)$ met $q_1 \in A'$, zodat inderdaad $w \in \mathcal{L}(M')$.

$$* 5. \text{Copy}(\mathcal{L}) := \{ww : w \in \mathcal{L}\}$$

$$* 6. \text{Fold}(\mathcal{L}) = \{xy^R : xy \in \mathcal{L} \text{ en } |x| = |y|\}$$

$$* 7. \text{Prefixes}(\mathcal{L}) = \{x \in \Sigma^* : (\exists y \in \Sigma^+)(xy \in \mathcal{L})\}$$

$$* 8. \text{InvPrefixes}(\mathcal{L}) = \{w \in \Sigma^* : \text{elke prefix van } w \text{ behoort tot } \mathcal{L}\}$$

$$\text{⚡ } 9. \text{Interlace}(\mathcal{L}_1, \mathcal{L}_2) = \{x_1 y_1 x_2 y_2 \cdots x_n y_n : x_1, \dots, x_n \in \mathcal{L}_1 \text{ en } y_1, \dots, y_n \in \mathcal{L}_2\}$$

$$\text{⚡ } 10. \text{RiffleShuffle}(\mathcal{L}_1, \mathcal{L}_2) = \{x_1 y_1 x_2 y_2 \cdots x_n y_n : (x_1 \cdots x_n) \in \mathcal{L}_1 \text{ en } (y_1 \cdots y_n) \in \mathcal{L}_2\}$$

$$\text{⚡ } 11. \text{Cycle}(\mathcal{L}) = \{yx : xy \in \mathcal{L}\}$$

39. Als $\text{MaxString}(\mathcal{L})$ (zie oefening 38.4) regulier is, geldt dan steeds dat \mathcal{L} ook regulier is? Leg uit.

Nee! Een tegenvoorbeeld wordt gegeven door $\mathcal{L} = \{a^m b^m c^n : m, n \geq 0\}$. Deze is duidelijk niet regulier (beschouw de doorsnede met $a^* b^*$), terwijl $\text{MaxString}(\mathcal{L}) = \emptyset$ wél (triviaal) regulier is. Ook de talen $\{a^p : p \text{ is priem}\}$ en $\{a^m b a^n : m \leq n\}$ zijn tegenvoorbeelden.

40. Beschrijf algoritmes voor de volgende problemen.

We verwijzen naar hoofdstuk 7 in de cursus voor een overzicht van de algoritmes die reeds gekend zijn, zoals een beslissingsprocedure $\text{equalFSM}(M, N)$ die `True` teruggeeft indien $\mathcal{L}(M) = \mathcal{L}(N)$ en `False` anders.

1. Gegeven twee EDA's M en N , bepaal of $\mathcal{L}(M) = \mathcal{L}(N)^R$.
 - Construeer een EDA N' zodanig dat $\mathcal{L}(N') = \mathcal{L}(N)^R$; zie oefeningenreeks 2.
 - Gebruik de procedure equalFSM om te bepalen of $\mathcal{L}(M) = \mathcal{L}(N')$.
2. Gegeven twee EDA's M en N , bepaal of $|\mathcal{L}(M)| < |\mathcal{L}(N)|$.

Uit de theorie weten we dat een taal ofwel eindig, ofwel aftelbaar oneindig is. We kennen ook reeds een procedure $\text{isInfinite}(M)$ om te bepalen of een EDA M al dan niet een oneindige taal genereert (zie de cursus).

- Controleer met $\text{isInfinite}(M)$ of $\mathcal{L}(M)$ oneindig is; indien ja, return `False`.
- Controleer met $\text{isInfinite}(N)$ of $\mathcal{L}(N)$ oneindig is; indien ja, return `True`.
- Als we tot hier geraken, zijn zowel $\mathcal{L}(M)$ als $\mathcal{L}(N)$ eindig.

Noteer het aantal toestanden van M als k_M . Merk op M slechts woorden w waarvoor $|w| \leq k_M$ kan aanvaarden; mocht namelijk $|w| > k_M$ en $w \in \mathcal{L}(M)$, dan moet tijdens het verwerken van w de automaat zich twee keer in dezelfde toestand bevinden (duivenhokprincipe), en door die lus naar believen te overlopen, vinden we een oneindig aantal woorden in $\mathcal{L}(M)$ —een strijdigheid. Uiteraard geldt op dezelfde manier dat $|w| \leq k_N$ voor alle $w \in \mathcal{L}(N)$.

Overloop dus alle woorden in Σ^* met lengte ten hoogste k_M (dit is een eindig aantal!) en houdt met een teller c_M het aantal woorden in $\mathcal{L}(M)$ bij. Doe vervolgens hetzelfde met de woorden begrensd door k_N , testend of deze in $\mathcal{L}(N)$ zitten, bijhoudend in c_N .

Uiteindelijk, indien $c_M < c_N$, return `True`, en anders `False`.

3. Zij $\Sigma = \{a, b\}$. Gegeven twee reguliere uitdrukkingen α en β , bepaal of er een $w \in \{a, b\}^*$ bestaat zodanig dat $w \in \mathcal{L}(\alpha)$, $w \in \mathcal{L}(\beta)$, $wa \in \mathcal{L}(\alpha)$, maar $wa \notin \mathcal{L}(\beta)$.
 - Construeer eerst twee EDA's M en N waarvoor $\mathcal{L}(M) = \mathcal{L}(\alpha)$ en $\mathcal{L}(N) = \mathcal{L}(\beta)$.
 - Construeer hieruit EDA's M' en N' waarvoor $w \in \mathcal{L}(M')$ als en slechts als $wa \in \mathcal{L}(\alpha)$, en $w \in \mathcal{L}(N')$ als en slechts als $wa \in \mathcal{L}(\beta)$; de procedure volgt verderop.
 - Construeer een EDA N'' zodanig dat $\mathcal{L}(N'') = \overline{\mathcal{L}(N')}$.
 - Construeer nu een EDA O zodanig dat $\mathcal{L}(O) = \mathcal{L}(M) \cap \mathcal{L}(N) \cap \mathcal{L}(M') \cap \mathcal{L}(N'')$.
 - Controleer tot slot de procedure uit de cursus of $\mathcal{L}(O) \neq \emptyset$.

We moeten enkel nog een procedure beschrijven die uit een automaat M voor een reguliere expressie ρ , een EDA M' construeert zodanig dat $w \in \mathcal{L}(M')$ als en slechts als $wa \in \mathcal{L}(\rho)$. Initialiseer daarvoor M' gelijk aan de machine M , met als verschil dat we voor M' volgende aanvaardende toestanden kiezen:

$$A' = \{p \in K : (p, a) \vdash_M (q, \varepsilon) \text{ met } q \in A\}.$$

Is dit algoritme correct? Met andere woorden, is $w \in \mathcal{L}(M')$ als en slechts als $wa \in \mathcal{L}(M)$?

Veronderstel dat $w \in \mathcal{L}(M')$, zodat $(s, w) \vdash_{M'}^* (p, \varepsilon)$ met $p \in A'$. Dan is $(s, wa) \vdash_{M'}^* (p, a)$. Omdat M' en M dezelfde transitiefunctie hebben, betekent dit dat ook $(s, wa) \vdash_M^* (p, a)$.

Per definitie van A' geldt dat $(p, a) \vdash_M (q, \varepsilon)$ met $q \in A$, zodat $(s, wa) \vdash_M^* (q, \varepsilon)$ met $q \in A$. We besluiten dat inderdaad $wa \in \mathcal{L}(M)$.

Veronderstel omgekeerd dat $wa \in \mathcal{L}(M)$. Dan bestaat er toestanden p en q zodanig dat

$$(s, wa) \vdash_M^* (p, a) \vdash_M (q, \varepsilon),$$

met $q \in A$. Dit leert dat $p \in A'$. Opnieuw omdat M en M' dezelfde transitiefunctie hebben, betekent dit dat $(s, wa) \vdash_{M'}^* (p, a)$, of dus dat $(s, w) \vdash_{M'}^* (p, \varepsilon)$, waarin $p \in A'$. We besluiten dat inderdaad $w \in \mathcal{L}(M')$.

4. Gegeven een reguliere grammatica \mathcal{G} , bepaal of $\mathcal{L}(\mathcal{G})$ regulier is.

We weten dat een reguliere grammatica *altijd* een reguliere taal genereert, dus return True.

- * 5. Gegeven een reguliere grammatica \mathcal{G} en een reguliere expressie α , bepaal of $L(\mathcal{G}) = L(\alpha)$.

- * 6. Zij $\Sigma = \{a, b\}$. Gegeven twee reguliere uitdrukkingen α en β , bepaal of

$$(L(\beta) = a^*) \vee (\forall w \in \{a, b\}^*) (|w| \text{ even} \Rightarrow w \in L(\alpha)).$$

- * 7. Gegeven een reguliere grammatica \mathcal{G} , bepaal of deze woorden met even lengte genereert.

- ⚡ 41. Gegeven een zekere taal \mathcal{L} , definiëren we de *vierkantswortel* $\sqrt{\mathcal{L}}$ als de taal $\{w \in \Sigma^* : ww \in \mathcal{L}\}$. Zo bevat de vierkantswortel van het Nederlands onder meer de “woorden” pa, ma, jo, bon, ker, cous, frou, pili... Bewijs dat als \mathcal{L} regulier is, dan ook $\sqrt{\mathcal{L}}$.

Reeks 5

42. Geef voor de volgende talen een contextvrije grammatica.

1. $\{ww^R : w \in \{a, b\}^*\}$

Voor deze taal volstaan $S \rightarrow aSa \mid bSb \mid \varepsilon$.

2. $\{a^k b^l c^m d^n : k \neq l \text{ of } m \neq n\}$

We construeren alvast de herschrijfgeregels

- $A \rightarrow aA \mid \varepsilon$,
- $B \rightarrow bB \mid \varepsilon$,
- $C \rightarrow cC \mid \varepsilon$,
- $D \rightarrow dD \mid \varepsilon$,

die reeksen van a 's, b 's, c 's of d 's voortbrengen. Vanuit het startsymbool onderscheiden we twee mogelijkheden: ofwel moeten we $k \neq l$ forceren (via de niet-terminaal X) en mogen m en n willekeurig zijn, ofwel forceren we $m \neq n$ (via Y) en zijn k en l willekeurig.

- $S \rightarrow XCD \mid ABY$.

Voor X zijn er opnieuw twee mogelijkheden: ofwel is $k > l$, ofwel $k < l$. In beide gevallen genereren we eerst evenveel a 's als b 's tot $\min(k, l)$ bereikt wordt. Daarna voegen we ofwel een (niet-nul) aantal a 's toe, ofwel een aantal b 's. De uitwerking van Y verloopt analoog.

- $X \rightarrow aXb \mid aA \mid bB$,
- $Y \rightarrow cYd \mid cC \mid dD$.

3. $\{a^k b^l c^m d^n : k + l = m + n\}$

We genereren van buiten naar binnen evenveel a 's als d 's. Daarna zijn er drie mogelijkheden: ofwel mogen er nog a 's bijkomen, ofwel d 's, ofwel geen van beide meer. Dit resulteert in

$$\begin{cases} S \rightarrow aSd \mid aXc \mid bYd \mid bZc \mid \varepsilon, \\ X \rightarrow aXc \mid bZc \mid \varepsilon, \\ Y \rightarrow bYd \mid bZc \mid \varepsilon, \\ Z \rightarrow bZc \mid \varepsilon. \end{cases}$$

4. $\{a^m b^n : m - n \text{ is positief en oneven}\}$

Merk op dat $a^m b^n = a^{m-n} a^n b^n$.

We schrijven $S \rightarrow XY$, waarbij X het gedeelte a^{m-n} (met $m - n$ oneven) zal genereren, en Y het gedeelte $a^n b^n$.

$$\begin{cases} S \rightarrow XY, \\ X \rightarrow aaX \mid a, \\ Y \rightarrow aYb \mid \varepsilon. \end{cases}$$

* 5. $\{a^m b^n : 2m \neq 3n + 1\}$

* 6. $\{w \in \{a, b\}^* : \#_a(w) = 2 \#_b(w)\}$

* 7. $\{a^i b^j c^k : i \leq j \text{ of } j \leq k\}$

43. Geef voor de taal $\{a^m b^n : 2m = 3n + 1\}$ een contextvrije grammatica en bewijs de correctheid.

De relatie $2m = 3n + 1$ betekent dat als we drie a 's aan een geldig woord toevoegen, er twee b 's moeten bijkomen. Het kortste woord in \mathcal{L} is aab . Een mogelijke grammatica is dus

$$\mathcal{G} = \begin{cases} S \rightarrow aaTb, \\ T \rightarrow aaaTbb \mid \varepsilon, \end{cases}$$

of alternatief

$$\mathcal{G}' = \{ S \rightarrow aaaSbb \mid aab \}.$$

We bewijzen nu rigoureuus dat $\mathcal{L}(\mathcal{G}) = \mathcal{L}$ (voor de grammatica \mathcal{G}' gaat het bewijs gelijkaardig).

Is $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}$?

We tonen via inductie op de lengte van de afleiding aan dat elke tussenstap x van de vorm $x = S$, $x = a^m T b^n$, of $x = a^m b^n$ is, waarin $2m = 3n + 1$. De inductiebasis is triviaal: als we nog geen herschrijfgeregels hebben toegepast, is inderdaad $x = S$. Beschouw nu zo'n x van deze vorm, pas een herschrijfgregel uit \mathcal{G} toe, en noem het resultaat y . Er zijn drie mogelijkheden te onderzoeken.

- We pasten $S \rightarrow aaTb$ toe. Dit is enkel mogelijk voor $x = S$, zodat $y = aaTb$.
- We pasten $T \rightarrow aaaTbb$ toe. Dan was $x = a^m T b^n$ met $2m = 3n + 1$, zodat $y = a^{n+3} T b^{n+2}$ waarin geldt dat $2(n+3) = 3(n+2) + 1$.
- We pasten $T \rightarrow \varepsilon$ toe. Dan was $x = a^m T b^n$ met $2m = 3n + 1$, zodat $y = a^m b^n$.

In elk van de gevallen blijkt y inderdaad opnieuw van de gezochte vorm te zijn. Zij nu $w \in \mathcal{L}(\mathcal{G})$ een volledig uitgewerkt woord. Aangezien in w geen niet-terminalen S en T meer mogen staan, moet het geval $w = a^m b^n$ met $2m = 3n + 1$ optreden, zodat inderdaad $w \in \mathcal{L}$.

Is $\mathcal{L} \subseteq \mathcal{L}(\mathcal{G})$?

We bewijzen dit via inductie op $|w|$. Het kortste woord in \mathcal{L} is $w = aab$, en deze wordt duidelijk gegenereerd door \mathcal{G} . Zij nu $w = a^m b^n \in \mathcal{L}$ verschillend van aab . Dan bevat w minstens twee b 's, zodat $2m = 3n + 1 \geq 7$. Er zijn dus minstens vier a 's! We kunnen dan schrijven dat $w = a^3 w' b^2$ met $w' = a^{m-3} b^{n-2}$. Merk nu op dat

$$2 \#_a(w') = 2(m-3) = 2 \#_a(w) - 6 = 3 \#_b(w) - 5 = 3(n-2) + 1 = 3 \#_b(w') + 1,$$

dus $w' \in \mathcal{L}$. Aangezien $|w'| < |w|$, mogen we uit de inductiehypothese besluiten dat $w' \in \mathcal{L}(\mathcal{G})$; met andere woorden, \mathcal{G} genereert het woord w' . De laatste regel in een afleiding van w' kan enkel $T \rightarrow \varepsilon$ zijn; dit is namelijk de enige waar rechts geen niet-terminalen meer staan. De laatste stap was dus $a^{m-3} T b^{n-2} \Rightarrow_{\mathcal{G}} a^{m-3} b^{n-2} = w'$. Door eerst de herschrijfgregel $T \rightarrow aaaTbb$ nog eens toe te passen, kunnen we dan ook

$$S \Rightarrow_{\mathcal{G}}^* a^{m-3} T b^{n-2} \Rightarrow_{\mathcal{G}} a^{m-3} (aaaTbb) b^{n-2} \Rightarrow_{\mathcal{G}} a^{m-3} (aaabb) b^{n-2} = a^m b^n$$

afleiden, zodat inderdaad $a^m b^n = w \in \mathcal{L}(\mathcal{G})$.

44. Een woord $w \in \{a, b\}^*$ heet *gebalanceerd* indien $\#_a(w) = \#_b(w)$ en indien $\#_a(x) \geq \#_b(x)$ voor elke prefix x van w . Geef voor de taal van de gebalanceerde woorden een contextvrije grammatica en toon de correctheid aan. *Opmerking: in de praktijk is vooral de taal van de gebalanceerde haakjes, met $a = '('$ en $b = ')'$, relevant.*

We zullen bewijzen dat de grammatica \mathcal{G} met regels $S \rightarrow SS \mid aSb \mid \varepsilon$ volstaat.

Is $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}$?

Onderstel $S \Rightarrow_{\mathcal{G}}^* w$. We bewijzen via inductie op de lengte van de afleiding dat w gebalanceerd is. Concreet zullen we bewijzen dat elke stap in de afleiding een woord in $\{a, b, S\}^*$ is, met evenveel a 's als b 's, waarin elke prefix minstens zoveel a 's als b 's heeft—ook al treden er extra letters S op. Voor de inductiebasis is $w = S$ duidelijk oké. Beschouw vervolgens een afleiding met lengte $n+1$ en schrijf $S \Rightarrow_{\mathcal{G}}^* w' \Rightarrow w$. Volgens de inductiehypothese is w' gebalanceerd, met extra S tussenin. We onderscheiden drie mogelijkheden voor de laatste toegepaste regel op w' om w te bekomen.

- We pasten $S \rightarrow SS$ toe. Aangezien één S in w' vervangen door SS duidelijk geen effect heeft op het aantal a 's en b 's of op de verdeling ervan, blijft w gebalanceerd.
- We pasten $S \rightarrow aSb$ toe. Schrijf $w' = uSv$ voor zekere $u, v \in \{a, b, S\}^*$ zodat $w = uaSbv$. Het is duidelijk dat ook $\#_a(w) = \#_b(w)$. Beschouw een willekeurige prefix x van w ; er zijn opnieuw een aantal gevallen te onderscheiden.

- x is een prefix van u , en dus ook van w' : dan volgt meteen uit de inductiehypothese dat inderdaad $\#_a(x) \geq \#_b(x)$.
- x is gelijk aan ua : de inductiehypothese stelt dat $\#_a(u) \geq \#_b(u)$, zodat

$$\#_a(ua) = \#_a(u) + 1 \geq \#_b(u) + 1 = \#_b(ua) + 1.$$
- x is gelijk aan uaS : de inductiehypothese stelt dat $\#_a(u) \geq \#_b(u)$, zodat

$$\#_a(uaS) = \#_a(u) + 1 \geq \#_b(u) + 1 = \#_b(uaS) + 1.$$
- x is gelijk aan $uaSb$: de inductiehypothese stelt dat $\#_a(u) \geq \#_b(u)$, zodat

$$\#_a(uaSb) = \#_a(u) + 1 \geq \#_b(u) + 1 = \#_b(uaSb).$$
- x is gelijk aan $uaSbv'$, met v' een bepaalde prefix van v : de inductiehypothese stelt dat $\#_a(uSv') \geq \#_b(uSv')$ (want uSv' is een prefix van w'), zodat

$$\#_a(uaSbv') = \#_a(uSv') + 1 \geq \#_b(uSv') + 1 = \#_b(uaSbv').$$

We concluderen dat voor elke mogelijke prefix x van w inderdaad geldt dat $\#_a(x) \geq \#_b(x)$.

- *We pasten $S \rightarrow \varepsilon$ toe.* Dit geval verloopt volledig analoog als het vorige.

We mogen dus uit het inductieprincipe besluiten dat w gebalanceerd is als $S \Rightarrow_{\mathcal{G}}^* w$ (met de caveat dat er nog extra S mogen staan in w). Zij nu $w \in \mathcal{L}(\mathcal{G})$, dus concreet $S \Rightarrow_{\mathcal{G}}^* w$ en het resultaat w bestaat volledig uit terminalen. Dan is w een gebalanceerd woord zonder S , zodat $w \in \mathcal{L}$.

Is $\mathcal{L} \subseteq \mathcal{L}(\mathcal{G})$?

We gebruiken inductie op $|w|$. De inductiebasis is opnieuw duidelijk: als $|w| = 0$, dan is inderdaad $w = \varepsilon \in \mathcal{L}(\mathcal{G})$. Onderstel vervolgens dat $|w| > 0$. We willen w opsplitsen in twee *gebalanceerde* woorden van kleinere lengte; dan kunnen we immers de inductiehypothese toepassen. Definieer $g(i) = \#_a(w(i)) - \#_b(w(i))$, waarin $w(i)$ de prefix van w met lengte i voorstelt. Het is duidelijk dat $g(|w|) = 0$. Toon nu zelf de volgende claim aan: *als voor een bepaalde i geldt dat $g(i) = 0$, dan zijn zowel de prefix $x = w(i)$ als de bijhorende suffix y (met $w = xy$) gebalanceerde woorden!*

Er zijn dus twee gevallen.

- *Er bestaat een $0 < i < |w|$ waarvoor $g(i) = 0$.* Dan is $w = xy$, met x en y twee gebalanceerde, *niet-ledige* woorden. Omdat de lengtes van x en y dan strikt kleiner zijn dan de lengte van w , leert de inductiehypothese dat $x, y \in \mathcal{L}(\mathcal{G})$. Er bestaan dus afleidingen $S \Rightarrow_{\mathcal{G}}^* x$ en $S \Rightarrow_{\mathcal{G}}^* y$. Door nu eerst de regel $S \Rightarrow_{\mathcal{G}} SS$ toe te passen, kunnen we zo het woord $xy = w$ genereren, en we besluiten dat $w \in \mathcal{L}(\mathcal{G})$.
- *Voor alle $0 < i < |w|$ is $g(i) \neq 0$, en dus $g(i) > 0$.* Merk op dat ieder gebalanceerd woord w moet beginnen met een a (anders bevat diens prefix $w(1) = b$ meer b 's dan a 's) en eindigen met een b (anders bevat $w(|w| - 1)$ meer b 's dan a 's). In elk geval is w dus van de vorm $aw'b$. Toon nu zelf de volgende claim aan: *als overal $g(i) > 0$, dan is ook w' gebalanceerd.*

Bemerk dat deze voorwaarde op g niet weggelaten kan worden, want bijvoorbeeld het woord $w = abab$ (waarvoor $g(2) = 0$) is wel gebalanceerd, maar diens $w' = ba$ niet.

Aangezien $|w'| < |w|$, leert de inductiehypothese dat $w' \in \mathcal{L}(\mathcal{G})$, dus beschouw een afleiding $S \Rightarrow_{\mathcal{G}}^* w'$. Door deze te laten voorafgaan door de herschrijfgregel $S \rightarrow aSb$, kunnen we ook $aw'b = w$ genereren met de grammatica, zodat $w \in \mathcal{L}(\mathcal{G})$.

In elk geval blijkt dat ook $w \in \mathcal{L}(\mathcal{G})$, zodat via inductie $\mathcal{L} \subseteq \mathcal{L}(\mathcal{G})$.

45. Verwijder alle nutteloze regels van de volgende grammatica's.

$$1. \begin{cases} S \rightarrow aS \mid A, \\ A \rightarrow bD \mid abE \mid baF, \\ B \rightarrow D \mid bA \mid \varepsilon, \\ C \rightarrow A, \\ E \rightarrow EE, \\ F \rightarrow bF \mid a. \end{cases}$$

We passen het algoritme `removeUnproductive` uit hoofdstuk 8 toe.

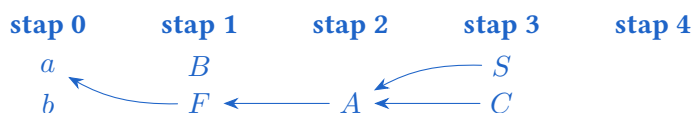
We markeren eerst alle niet-terminale symbolen $\{S, A, B, C, D, E, F\}$ als niet-productief, en alle terminalen $\{a, b\}$ als productief.

Nu zoeken we alle regels waarin rechts enkel productieve symbolen staan, terminaal of niet. Dit zijn de regels $B \rightarrow \varepsilon$ (merk op dat rechts in feite geen enkel symbool staat!) en $F \rightarrow a$, en we markeren ook B en F als productief.

Herhalen we deze procedure, dan vinden we bijkomend de twee regels $A \rightarrow baF$ en $F \rightarrow bF$. We markeren ook A als productief; F is reeds gemarkeerd.

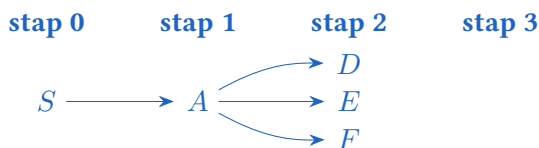
Nogmaals herhalen leidt tot de regels $S \rightarrow A$, $B \rightarrow bA$ en $C \rightarrow A$, dus ook het startsymbool is productief (gelukkig maar!), net als B .

Nogmaals herhalen leidt enkel tot de regel $S \rightarrow aS$, maar S is al gemarkeerd. In deze stap markeren we geen nieuwe symbolen meer en het algoritme stopt hier. De enige gemarkeerde (en dus productieve) niet-terminalen zijn dus $\{S, A, B, C, F\}$; D en E zijn niet-productief.



We passen vervolgens het algoritme `removeUnreachable` uit hoofdstuk 8 toe.

We markeren eerst enkel het startsymbool S als bereikbaar en lezen af dat de enige nieuwe niet-terminaal bereikbaar uit S , A is. Daarna blijken uit A ook D , E en F bereikbaar. Hier stopt het algoritme, en we besluiten dat de enige bereikbare niet-terminalen $\{S, A, D, E, F\}$ zijn; B en C zijn niet bereikbaar.



Een vereenvoudigde grammatica, zonder niet-productieve of onbereikbare niet-terminalen, wordt dus

$$* 2. \begin{cases} S \rightarrow aS \mid AB, \\ A \rightarrow baF \mid a, \\ B \rightarrow AA, \\ C \rightarrow bA. \end{cases}$$

46. Verwijder alle nutteloze, ε - en 1-regels uit de volgende grammatica's.

$$1. \begin{cases} S \rightarrow aaB \mid AaB, \\ A \rightarrow \varepsilon, \\ B \rightarrow bbA \mid \varepsilon. \end{cases}$$

Alle niet-terminalen zijn productief én bereikbaar.

Bovendien zijn er geen 1-regels (van de vorm $X \rightarrow Y$).

We moeten wel de ε -regels ($A \rightarrow \varepsilon$ en $B \rightarrow \varepsilon$) wegwerken. Via het algoritme `findNullable` (dat sterk lijkt op `removeUnproductive`) blijken inderdaad A en B *nullable*, maar S niet. Nu passen we `removeEps` toe.

De eerste aanpasbare regel $S \rightarrow aaB$ wordt door `removeEps` herwerkt tot de regel $S \rightarrow aa$ (maar merk op dat we de originele regel niet schrappen). Dan wordt $S \rightarrow AaB$ omgezet in $S \rightarrow aB$ (met A als *nullable*) en in $S \rightarrow Aa$ (met B als *nullable*). Deze twee regels zijn zelf opnieuw aanpasbaar, en worden nogmaals herwerkt tot $S \rightarrow a$ (beide dezelfde). $B \rightarrow bbA$ tot slot wordt $B \rightarrow bb$. Schrappen we nu nog $A \rightarrow \varepsilon$ en $B \rightarrow \varepsilon$, dan eindigen we met

$$\begin{cases} S \rightarrow aaB \mid aa \mid AaB \mid Aa \mid aB \mid a, \\ B \rightarrow bbA \mid bb. \end{cases}$$

Omdat $\varepsilon \notin \mathcal{L}(\mathcal{G})$, hoeven we de hulpprocedure `atMostOneEps` niet toe te passen.

Merk op dat we na toepassen van `removeEps` een grammatica krijgen die niet-productieve niet-terminalen bevat! A is immers niet-productief. We werken deze weg en krijgen

$$\begin{cases} S \rightarrow aaB \mid aa \mid aB \mid a, \\ B \rightarrow bb. \end{cases}$$

We kunnen deze grammatica nog wat verder vereenvoudigen tot $S \rightarrow aabb \mid aa \mid abb \mid a$.

$$2. \begin{cases} S \rightarrow aSa \mid B \mid D, \\ B \rightarrow bbC \mid bb, \\ C \rightarrow \varepsilon \mid cCC, \\ D \rightarrow dD, \\ E \rightarrow FC, \\ F \rightarrow f. \end{cases}$$

E en F zijn niet bereikbaar, D is niet-productief, en C is *nullable*. Dit wegwerken geeft

$$\begin{cases} S \rightarrow aSa \mid B, \\ B \rightarrow bbC \mid bb, \\ C \rightarrow cCC \mid cC \mid c. \end{cases}$$

We zien dat ook na wegwerken van de ε -regels geen onbereikbare of niet-productieve niet-terminalen worden geïntroduceerd. Er blijft wel nog een 1-regel ($S \rightarrow B$) over. We werken deze via de procedure `removeUnits` weg tot

$$\begin{cases} S \rightarrow aSa \mid bbC \mid bb, \\ B \rightarrow bbC \mid bb, \\ C \rightarrow cCC \mid cC \mid c. \end{cases}$$

Nu hebben we echter B onbereikbaar gemaakt! We vinden finaal de grammatica

$$\begin{cases} S \rightarrow aSa \mid bbC \mid bb, \\ C \rightarrow cCC \mid cC \mid c. \end{cases}$$

$$* 3. \begin{cases} S \rightarrow a \mid aA \mid B \mid C, \\ A \rightarrow aB \mid \varepsilon, \\ B \rightarrow Aa, \\ C \rightarrow cCD, \\ D \rightarrow ddd. \end{cases}$$

$$* 4. \begin{cases} S \rightarrow ABC, \\ A \rightarrow aC \mid D, \\ B \rightarrow bB \mid \varepsilon \mid A, \\ C \rightarrow Ac \mid \varepsilon \mid Cc, \\ D \rightarrow aa. \end{cases}$$

* 47. Vervolledig het bewijs van stelling 11.4: bewijs dat de contextvrije talen gesloten zijn onder unie, concatenatie en Kleene-ster.

⚡ 48. In de cursus zagen we dat alle *syntactisch correcte* Javaprogramma's een contextvrije taal vormen. In deze oefening tonen we aan dat de verzameling van alle *semantisch correcte* Javaprogramma's echter *niet* contextvrij is. Beschouw volgend programma $Q(a, b, c)$ met drie parameters a, b en c .

$$Q(a, b, c) = \left\{ \begin{array}{l} \text{public class } Q \{ \\ \quad \text{public static void main(String[] args) } \\ \quad \quad \text{int } x^a = 0; \\ \quad \quad x^b = x^c; \\ \quad \} \\ \} \end{array} \right.$$

Merk op dat dit programma foutloos compileert als en slechts als $a = b = c$.

1. Toon aan dat $\{Q(a, b, c) : a, b, c \geq 1\}$ een reguliere taal is.
2. Toon aan dat $\{Q(a, b, c) : a, b, c \geq 1, Q(a, b, c) \text{ compileert zonder fouten}\}$ niet contextvrij is.
3. Besluit dat de taal van alle Javaprogramma's die foutloos compileren, niet contextvrij is.

⚡ 49. Veronderstel dat we de capaciteit van een PDA willen beperken door slechts een eindig geheugen te gebruiken. Concreet laten we op de stapel slechts $t + 1$ beschrijfbaar posities $0, \dots, t$ toe, voor zekere constante $t \in \mathbb{N}$. De PDA crasht als deze een $(t + 1)$ -de element op zijn stapel wil pushen; uiteraard wordt de input dan verworpen.

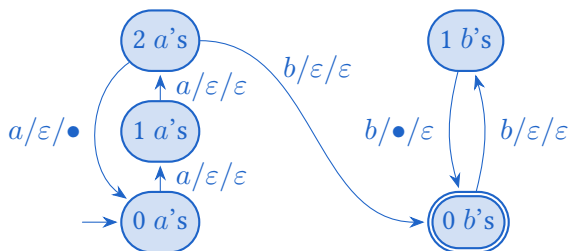
Bewijs dat dergelijke automaten (voor eender welke t) precies de reguliere talen aanvaarden.

Reeks 6

50. Geef PDA's voor de volgende talen.

1. $\{a^m b^n : 2m = 3n + 1\}$

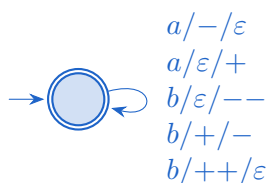
Deze taal is precies gelijk aan de taal $\{a^{3k+2} b^{2k+1} : k \geq 0\}$. Het idee is om de a 's in drietallen te verwerken, en per gelezen drietal een symbool \bullet op de stapel te zetten. Vervolgens lezen we de b 's in tweetallen en halen we per tweetal een \bullet van de stapel. Onthoud dat een woord uiteindelijk slechts aanvaard wordt indien we in een aanvaardende toestand belanden én de stapel leeg is.



Merk op: de automaat gaat slechts over naar de rechterhelft door een b te lezen wanneer er al $3k + 2$ letters a verwerkt zijn (k zijnde het aantal \bullet 's op dat moment op de stapel). Daarna moeten er nog precies $2k$ letters b worden gelezen om aanvaard te worden, dus halen we per twee b 's een \bullet van de stapel. Indien de automaat zich in de aanvaardende toestand bevindt én de stapel leeg is, hebben we in totaal $3k + 2$ a 's en $2k + 1$ b 's gelezen.

2. $\{w \in \{a, b\}^* : \#_a(w) = 2 \#_b(w)\}$

Het idee is om op de stapel met twee symbolen $+$ en $-$ te werken, waarbij de totale "waarde" op elk moment gelijk is aan $\#_a - 2 \#_b$ van de verwerkte input. Aangezien we niet weten in welke volgorde we de a 's en b 's te lezen krijgen, kunnen we positieve en negatieve waarden met willekeurig grote absolute waardes tegenkomen; daarom kunnen we niet werken met een enkel symbool zoals in de vorige oefening.



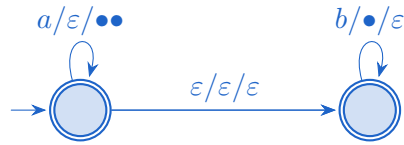
Voor iedere a die we verwerken, dient de waarde op de stapel met één te verhogen. Dit lukt op twee manieren: ofwel zetten we $+$ op de stapel, ofwel halen we (indien mogelijk) $-$ van de stapel. Voor iedere b die we verwerken, moet de waarde op de stapel met twee verlagen. Dit kan op twee manieren: ofwel zetten we $--$ op de stapel, ofwel halen we $++$ eraf, maar vergeet niet dat ook $+$ eraf halen en $-$ erop zetten kan. Uiteindelijk aanvaardt de automaat als het volledige woord gelezen en de stapel leeg is.

Merk op dat er meerdere mogelijkheden zijn om een woord te verwerken. De automaat kan bijvoorbeeld domweg voor elke gelezen letter $+$ of $--$ op zijn stapel zetten, en nooit iets eraf halen, in welk geval de input niet aanvaard zal worden (ongeacht het aantal a 's en b 's). Er is ook een gretige manier van verwerken, die steeds zoveel mogelijk van de stapel probeert te halen in plaats van toe te voegen. Omdat de automaat niet-deterministisch is, volstaat het dat woorden op één manier aanvaard kunnen worden; toon zelf aan dat de gretige manier wél alle woorden van de taal accepteert.

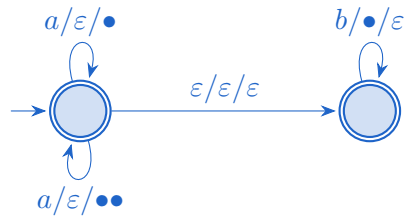
Nog een laatste opmerking: dat we ook de transitie $b/+/-$ nodig hebben, blijkt bijvoorbeeld uit het woord $abab$, dat anders niet aanvaard zou kunnen worden.

3. $\{a^n b^{2n} : n \geq 0\}$

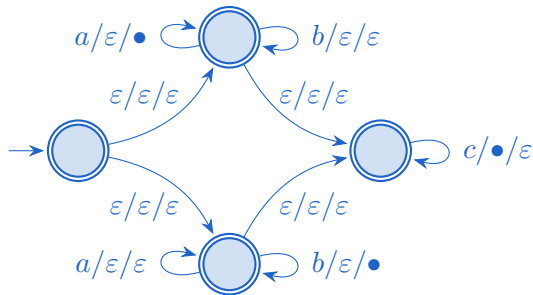
Deze lukt veel eenvoudiger dan de vorige oefening.



4. $\{a^m b^n : m \leq n \leq 2m\}$

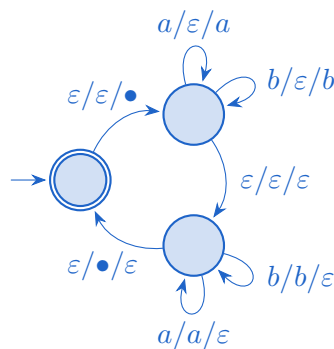


5. $\{x \in \{a, b\}^* \{c\}^* : \#_a(x) = \#_c(x) \text{ of } \#_b(x) = \#_c(x)\}$



6. \mathcal{L}^* waarbij $\mathcal{L} = \{xx^R : x \in \{a, b\}^*\}$

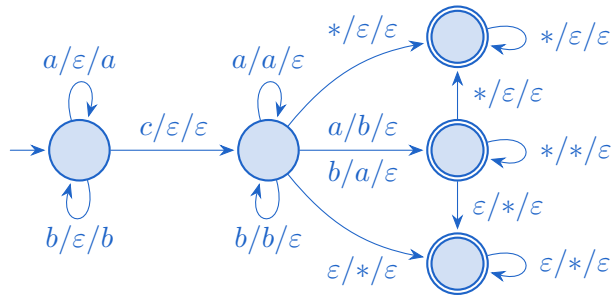
Deze is een beetje *tricky*. We gebruiken een speciaal symbool \bullet dat we onderaan op de stapel zetten, waaraan we kunnen herkennen of de stapel leeg is. Op die manier kunnen we onze stapel meerdere keren gebruiken in de zekerheid dat die telkens terug leeg is.



Het rechtergedeelte is de ondertussen vertrouwde automaat die \mathcal{L} aanvaardt; de constructie met de lus (en de \bullet !) genereren de Kleene-ster van \mathcal{L} .

7. $\{xycy : x, y \in \{a, b\}^* \text{ en } x \neq y^R\}$

De eerste tak houdt het geval $|x| < |y|$ bij, de tweede tak $|x| = |y|$ en de derde tak $|x| > |y|$.



Om de tekening niet al te zwaar te maken, gebruiken we een symbool $*$ voor a of b (niet ϵ !); de notatie $*/*/\epsilon$ bijvoorbeeld is dan een afkorting voor de vier transities $(a/a/\epsilon)$, $(a/b/\epsilon)$, $(b/a/\epsilon)$ en $(b/b/\epsilon)$.

- * 8. $\{w \in \{a, b\}^* : w \neq xx^R \text{ voor elke } x \in \{a, b\}^*\}$
- * 9. $\{a^m b^n c^p d^q : m + n = p + q\}$
- * 10. $\{a^i b^j c^k : i + k = j\}$
- * 11. $\{ab(ab)^n b(ba)^n : n \geq 0\}$
- * 12. $\{0^i 1^j 2^k : i \neq j \text{ of } j \neq k\}$
- * 13. $\{xycy : x, y \in \{a, b\}^* \text{ en } |x| \neq |y|\}$
- * 14. $\{a^m b^n : m \geq n \text{ en } m - n \text{ is even}\}$

51. Bepaal van de volgende talen of ze regulier of niet-regulier zijn, en contextvrij of niet-contextvrij. Bewijs je claim.

1. $\{a^{n^4} : n \geq 0\}$

is niet regulier of contextvrij.

We passen het *pumping lemma* voor contextvrije talen toe. Omdat de taal duidelijk oneindig groot is, bestaat er een getal k met alle eigenschappen van het pumping lemma. Beschouw nu $w = a^{k^4} \in \mathcal{L}$. Dan is $w = uvxyz$ met $|vxy| \leq k$, $|vy| > 0$, en $w_p = uv^p xy^p z \in \mathcal{L}$ voor alle $p \in \mathbb{N}$. Zij nu concreet $p = 2$, dan is

$$w_2 = uvvxyyz = a^{|u|} a^{2|v|} a^{|x|} a^{2|y|} a^{|z|} = a^{|w|+|vy|} = a^{k^4+|vy|}$$

opnieuw een woord van de taal, dus moet $k^4 + |vy|$ opnieuw een vierdemacht zijn. Echter,

$$k^4 < k^4 + |vy| \leq k^4 + |vxy| \leq k^4 + k < (k+1)^4,$$

dus vinden we een strijdigheid.

Omdat de taal al niet contextvrij is, is ze zeker niet regulier.

2. $\{a^m b^n : m \geq (n-1)^3\}$ is niet contextvrij.

Onderstel van wel; we passen het *pumping lemma* voor contextvrije talen toe. Omdat de taal duidelijk oneindig groot is, bestaat er een getal k met alle eigenschappen van het pumping lemma. Beschouw nu $w = a^{k^3} b^{k+1} \in \mathcal{L}$. Dan is $w = uvxyz$ met $|vxy| \leq k$, $|vy| > 0$, en $w_p = uv^p xy^p z \in \mathcal{L}$ voor alle $p \in \mathbb{N}$. We onderscheiden enkele mogelijkheden.

- Het woord v of het woord y bevat zowel a 's als b 's.
Dan is het opgepompte woord w_2 niet langer van de vorm $a^* b^*$; een strijdigheid.
- Zowel het woord v als het woord y bevatten geen a 's.
Opwaarts pompen (met $p = 2$) leert dat

$$w_2 = a^{k^3} b^{k+1+|v|+|y|} \in \mathcal{L},$$

maar $k^3 \not\geq (k + |v| + |y|)^3$; een strijdigheid.

- Zowel het woord v als het woord y bevatten geen b 's.
Neerwaarts pompen (met $p = 0$) leert dat

$$w_0 = a^{k^3 - |v| - |y|} b^{k+1} \in \mathcal{L},$$

maar $k^3 - |v| - |y| \not\geq k^3$; een strijdigheid.

- Het woord v bevat enkel a 's en het woord y enkel b 's.
We mogen aannemen dat v en y niet ledig zijn, want dan zitten we in een vorig geval.
Hier in dit geval is

$$w_p = a^{k^3 + (p-1) \cdot |v|} b^{k+1 + (p-1) \cdot |y|} \in \mathcal{L},$$

en dat voor alle $p \in \mathbb{N}$. Voor $p = 0$ staat er dat $w_0 = a^{k^3 - |v|} b^{k - |y| + 1} \in \mathcal{L}$, zodat

$$k^3 - |v| \geq (k - |y|)^3 = k^3 - 3k^2|y| + 3k|y|^2 - |y|^3.$$

Voor $p = 2$ staat er dat $w_2 = a^{k^3 + |v|} b^{k + |y| + 1} \in \mathcal{L}$, zodat

$$k^3 + |v| \geq (k + |y|)^3 = k^3 + 3k^2|y| + 3k|y|^2 + |y|^3.$$

Tellen we nu deze beide vergelijkingen op en delen we door twee, dan bekomen we dat $k^3 \geq k^3 + 3k|y|^2$, wat duidelijk een strijdigheid is omdat $3k|y|^2 > 0$.

3. $\{a^p : p \text{ is een priemgetal}\}$ is niet contextvrij.
4. $\{w \in \{a, b, c\}^* : \#_a(w) \leq \#_b(w)^2 \text{ en } \#_c(w) > 2\}$ is niet contextvrij.
5. $\{w \in \Sigma^* : (\exists n \in \mathbb{N})(|w| = n^3)\}$ (voor een willekeurig alfabet Σ) is niet contextvrij.

Beschouw een enkele letter $x \in \Sigma$; definieer \mathcal{L}' als de doorsnede met de reguliere taal $\{x\}^*$. Deze doorsnede is precies gelijk aan $\{x^{n^3} : n \geq 0\}$ en de rest volgt uit een argument analoog als in opgave 2.

6. $\{(ab)^n a^n b^n : n > 0\}$ is niet contextvrij.
7. $\{xy : |x| = 2|y| \text{ en } x, y \in a^+ b^+\}$ is contextvrij, maar niet regulier.
8. $\{a^n b^n : n \neq 5\}$ is contextvrij, maar niet regulier.

Dit volgt meteen uit stelling 11.9 uit de theorie, want de taal $\{a^5 b^5\}$ is eindig (dus regulier).

Dat $\{a^n b^n : n \neq 5\}$ niet regulier is, is eveneens eenvoudig in te zien: anders zou ook de unie met de reguliere taal $\{a^5 b^5\}$ regulier zijn, maar zo vinden we het vertrouwde tegenvoorbeeld $\{a^n b^n : n \geq 0\}$ terug.

9. $\{w \in \{0, 1\}^* : \text{de eerste letter en de laatste letter van } w \text{ zijn gelijk}\}$ is regulier.

Een reguliere expressie voor deze taal is $\varepsilon \cup 0 \cup 1 \cup 0(0 \cup 1)^* 0 \cup 1(0 \cup 1)^* 1$.

10. $\{uvv^R w^R : u, v, w \in \{0, 1\}^+\}$ is regulier.

Een reguliere expressie voor deze taal is $(0 \cup 1)^+(00 \cup 11)(0 \cup 1)^+$.

11. $\{a^i b^j c^k : i \leq k \leq j \text{ en } i \neq j\}$ is niet contextvrij.

12. $\{w \in \{0, 1\}^* : \#_0(w) = \#_1(w) \text{ en } w \text{ bevat geen deelwoord } 010\}$ is contextvrij.

We weten immers dat $\mathcal{L}_1 = \{w \in \{0, 1\}^* : \#_0(w) = \#_1(w)\}$ contextvrij is, en $\mathcal{L}_2 = \{w \in \{0, 1\}^* : w \text{ bevat geen deelwoord } 010\}$ regulier, zodat ook de doorsnede contextvrij is.

De taal is duidelijk niet regulier. De doorsnede met $0^* 1^*$ is immers de taal $\{0^n 1^n : n \geq 0\}$, en de substitutie $0 \mapsto a$ en $1 \mapsto b$ geeft het vertrouwde tegenvoorbeeld $\{a^n b^n : n \geq 0\}$.

* 13. $\{xy : x, y \in \{a, b\}^* \text{ en } |x| = |y|\}$

* 14. $\{w \in \{a, b, c\}^* : \#_a(w) \text{ is een veelvoud van zeven en } abc \text{ is een deelwoord}\}$

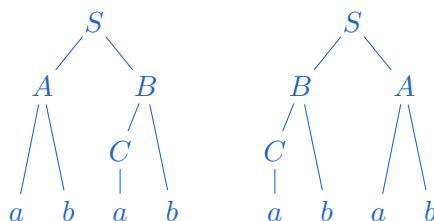
- * 15. $\{a^m b^n : m, n > 0 \text{ en } (m = n \text{ of } m = 2n)\}$
- * 16. $\{a^m b^{2m} c^n : m, n \geq 0\} \cap \{a^m b^n c^{2n} : m, n \geq 0\}$
- * 17. $\{w w^R w : w \in \{a, b\}^*\}$
- * 18. $\{w \in \{a, b, c, d\}^* : \#_a(w) = \#_b(w) \cdot (\#_c(w) - 1)^2\}$

Reeks 7

52. Toon aan dat volgende grammatica's dubbelzinnig zijn, en geef een equivalente, ondubbelzinnige grammatica.

$$1. \begin{cases} S \rightarrow AB \mid BA, \\ A \rightarrow aA \mid ab, \\ B \rightarrow Cb, \\ C \rightarrow aC \mid a. \end{cases}$$

We zien in dat zowel A als B woorden van de vorm a^+b genereren, dus de uiteindelijke taal is eenvoudigweg a^+ba^+b . Dat betekent ook dat de grammatica dubbelzinnig is, omdat A en B dezelfde rol spelen. Voor het woord $abab$ bijvoorbeeld bestaan er twee *parse trees*:



Een voor de hand liggende suggestie voor een equivalente ondubbelzinnige grammatica is

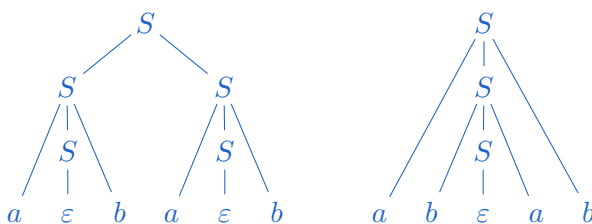
$$\mathcal{G} = \begin{cases} S \rightarrow AA, \\ A \rightarrow aA \mid ab. \end{cases}$$

We tonen nu aan dat \mathcal{G} inderdaad ondubbelzinnig is. Kies een willekeurig woord $w \in \mathcal{L}(\mathcal{G})$; we tonen aan dat deze een unieke linkse afleiding heeft. Bekijk een willekeurige tussenstap in een afleiding van w . We zitten op dat moment met een woord in $\{a, b, S, A\}^*$. Beschouw de meest linkse niet-terminaal in deze tussenstap; de vraag is nu of er op deze niet-terminaal precies één herschrijfgregel toegepast kan worden om w te bekomen, of meer dan één. Als er slechts één mogelijkheid is, dan is er een *unieke* linkse afleiding (en dus ook een *unieke* parse tree) voor w .

- Stel dat S de meest linkse niet-terminaal is. Dit kan maar optreden in de allereerste stap, en dan is er inderdaad slechts één regel toepasbaar om w af te leiden (nl. $S \rightarrow AA$).
- Stel dat A de meest linkse niet-terminaal is. De enige mogelijke regels die we dan kunnen toepassen, zijn $A \rightarrow aA$ en $A \rightarrow ab$. Deze eerste regel creëert extra a 's in de huidige tussenstap, en de tweede regel sluit dit gedeelte af met ab . Afhankelijk van w (concreet van de parameters m en n in $w = a^m b a^n b$) en de vorm van de huidige tussenstap, is het duidelijk dat we geen keuze hebben welke regel we moeten toepassen als we uiteindelijk het woord w willen vinden.

2. $S \rightarrow \varepsilon \mid aSa \mid bSb \mid aSb \mid bSa \mid SS$.

De dubbelzinnigheid zit 'm hier in de regel $S \rightarrow SS$. Het woord $abab$ bijvoorbeeld heeft twee parse trees.



De taal voortgebracht door deze grammatica is net de taal van alle woorden in $\{a, b\}^*$ met even lengte. We kunnen dus eigenlijk gewoon de regel $S \rightarrow SS$ weglaten! Probeer nu zelf in te zien waarom deze nieuwe grammatica, zonder deze ene regel, ondubbelzinnig is.

$$* 3. \begin{cases} S \rightarrow AB, \\ A \rightarrow AA \mid a, \\ B \rightarrow Cb, \\ C \rightarrow aC \mid a. \end{cases}$$

$$* 4. S \rightarrow aSb \mid aaSb \mid \varepsilon.$$

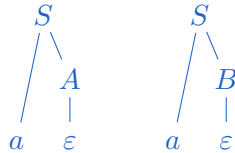
$$* 5. S \rightarrow aSb \mid bSa \mid SS \mid \varepsilon.$$

53. Is een reguliere grammatica altijd ondubbelzinnig?

Neen. Beschouw bijvoorbeeld de volgende reguliere grammatica voor de taal $\{a\}^+$;

$$\begin{cases} S \rightarrow aA \mid aB, \\ A \rightarrow aA \mid \varepsilon, \\ B \rightarrow aB \mid \varepsilon. \end{cases}$$

Dan is het woord a op twee manieren te verkrijgen.



54. Beschouw $\mathcal{L} = \{w \in \{a, b, \cup, \varepsilon, (,), *, +\}^* : w \text{ is een syntactisch correcte reguliere uitdrukking}\}$. Schrijf een ondubbelzinnige contextvrije grammatica die \mathcal{L} genereert. *Hint: je kunt je baseren op voorbeeld 8.1.12 in de cursus.*

Opmerking: in deze oefening kan er verwarring optreden omtrent het ledige woord. Indien we $S \rightarrow \varepsilon$ schrijven, bedoelen we normaal gezien dat S het lege woord genereert. Nu willen we ε echter als een terminaal symbool zien, omdat dit een reguliere uitdrukking is! Let dus op, in deze oefening stelt ε niet het ledige woord voor, maar een letter van het alfabet (een terminaal symbool).

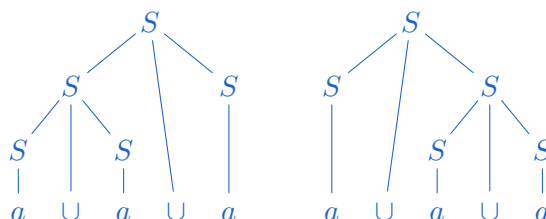
Voor deze oefening baseren we ons op de recursieve definitie van reguliere uitdrukkingen:

- als α en β reguliere uitdrukkingen zijn, dan ook $\alpha \cup \beta$ en $\alpha\beta$;
- als α een reguliere uitdrukking is, dan ook α^* , α^+ en (α) ;
- a , b en ε zijn reguliere uitdrukkingen.

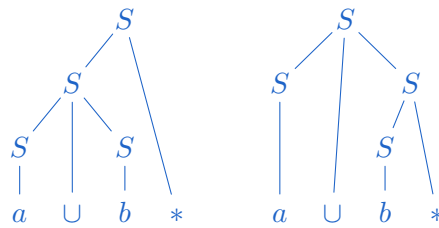
In eerste instantie construeren we dus de volgende grammatica, met één niet-terminaal S .

$$\begin{cases} S \rightarrow S \cup S \mid SS, \\ S \rightarrow S^* \mid S^+ \mid (S), \\ S \rightarrow a \mid b \mid \varepsilon. \end{cases}$$

Deze grammatica is echter niet ondubbelzinnig, zoals bijvoorbeeld $a \cup a \cup a$ uitwijst:



Of, dramatischer, $a \cup b^*$:



Merk op dat de afleiding links inhoudelijk $(a \cup b)^*$ voorstelt, maar rechts $a \cup (b^*)$!

Net zoals in voorbeeld 8.1.12 uit de cursus moeten we nog rekening houden met de prioriteit van de bewerkingen. De reguliere uitdrukking $aab \cup ba^*$ lezen we bijvoorbeeld als $aab \cup b(a)^*$, en niet als $(aab \cup ba)^*$ of $aab \cup (ba)^*$. Dit betekent dat we eerst de unie en de concatenatie moeten genereren en dan pas de Kleene-ster. Algemeen is de volgorde van belangrijkheid

1. allereerst hetgeen tussen haakjes staat,
2. daarna de Kleene-ster α^* en α^+ ,
3. daarna de concatenatie $\alpha\beta$,
4. tot slot de unie $\alpha \cup \beta$.

De bewerkingen $*$ en $+$ hebben dus een hogere prioriteit dan concatenatie, die op zijn beurt weer een hogere prioriteit heeft dan \cup . We definiëren de grammatica in levels, waarbij we van laagste prioriteit naar hoogste prioriteit toe werken (dus eerst \cup en pas later concatenatie, $*$ en $+$). Let op het gebruik van haakjes, die terug de hoogste prioriteit vereisen.

$$\left\{ \begin{array}{l} S \rightarrow S \cup X \mid X \\ X \rightarrow XY \mid Y \\ Y \rightarrow Y^* \mid Y^+ \\ Y \rightarrow (S) \\ Y \rightarrow a \mid b \mid \varepsilon. \end{array} \right.$$

Deze grammatica is ondubbelzinnig; toon dit zelf aan.

55. Beschrijf algoritmes voor de volgende problemen.

1. Gegeven een reguliere uitdrukking α en een PDA M , bepaal of $\mathcal{L}(M) \subseteq \mathcal{L}(\alpha)$.

Controleren of $\mathcal{L}(M) \subseteq \mathcal{L}(\alpha)$ is precies hetzelfde als controleren of $\mathcal{L}(M) \cap \overline{\mathcal{L}(\alpha)} = \emptyset$, en daarvoor kennen we een procedure:

- Construeer een EDA N zodat $\mathcal{L}(N) = \mathcal{L}(\alpha)$.
- Construeer een EDA N' zodat $\mathcal{L}(N') = \overline{\mathcal{L}(N)}$.
- Construeer een PDA O zodat $\mathcal{L}(O) = \mathcal{L}(M) \cap \mathcal{L}(N')$ via `intersectPDAandFSM`. Dit is de procedure die in het bewijs van stelling 11.8 werd gebruikt.
- Construeer een grammatica \mathcal{G} zodat $\mathcal{L}(\mathcal{G}) = \mathcal{L}(O)$ via de procedure `PDAtoCFG`.
- Controleer of $\mathcal{L}(\mathcal{G}) = \emptyset$ via de procedure `decideCFLEmpty`.

2. Gegeven een contextvrije grammatica \mathcal{G} , bepaal of \mathcal{G} een woord van even lengte genereert.

- Construeer een PDA M zodat $\mathcal{L}(M) = \mathcal{L}(\mathcal{G})$ via de procedure `CFGtoPDAtopdown`.
- Bepaal $\Sigma_{\mathcal{G}}$ uit de input \mathcal{G} .
- Construeer een EDA N zodat $\mathcal{L}(N) = \{w \in \Sigma_{\mathcal{G}}^* : |w| \text{ is even}\}$.
- Construeer een PDA O zodat $\mathcal{L}(O) = \mathcal{L}(M) \cap \mathcal{L}(N)$ via `intersectPDAandFSM`.

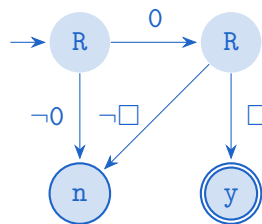
- Construeer een grammatica \mathcal{H} zodat $\mathcal{L}(\mathcal{H}) = \mathcal{L}(O)$ via PDA to CFG.
- Controleer of $\mathcal{L}(H) = \emptyset$ via `decideCFLEmpty`. Zo ja, return `False`; zo nee, return `True`.

- * 3. Gegeven een reguliere grammatica \mathcal{G} , bepaal of $\mathcal{L}(G)$ contextvrij is.
- * 4. Gegeven een contextvrije grammatica \mathcal{G} en woorden x en y , bepaal of $xy \in \mathcal{L}(\mathcal{G})$.

56. Geef voor de volgende talen een Turingmachine die de taal beslist.

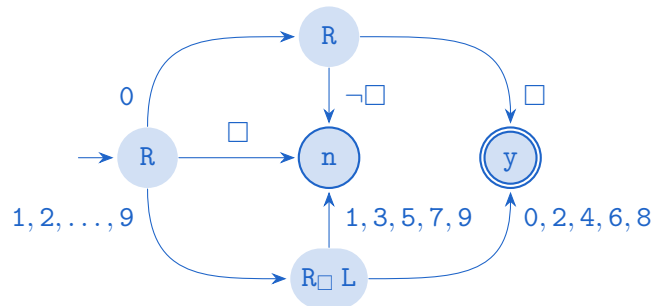
Let op: in macronotatie betekent L_{\square} (ga naar links totdat je een blanco vindt) iets helemaal anders dan $L \square$ (ga naar links en schrijf een blanco), dus probeer je notatie op papier netjes te houden!

1. $\{0\}$



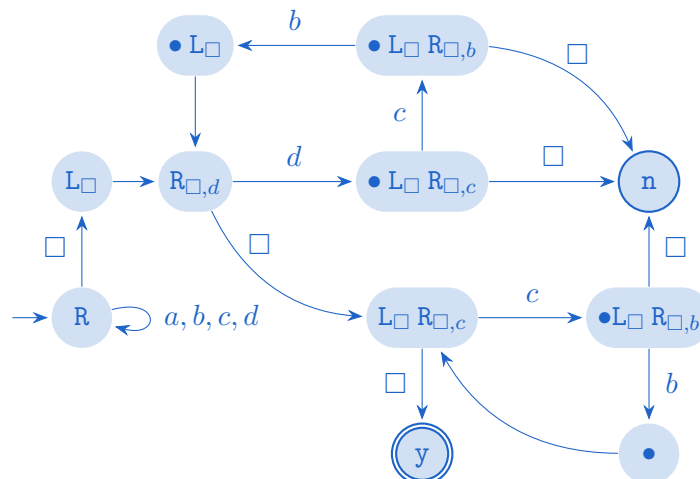
2. $\{w \in \{0, 1, \dots, 9\}^* : w \text{ is een even natuurlijk getal (zonder leidende nullen)}\}$

Deze taal is regulier; een implementatie zou dus niet bijzonder ingewikkeld mogen zijn.



3. $\{w \in \{a, b, c, d\}^* : \#_b(w) \geq \#_c(w) \geq \#_d(w) \geq 0\}$

In eerste instantie trachten we om voor iedere d in de input, ook één c en één b te schrappen. Lukt dat niet, dan verwerpen we w . Daarna zijn er geen d 's meer over (of hebben we de input reeds verworpen) en gaan we naar een tweede subroutine waar we voor elke overgebleven c nog één b trachten te schrappen.



* 6. $\{a^m b^n c^m d^n : m, n \geq 0\}$

* 7. $\{a^k b^l c^m : m > \max(k, l)\}$

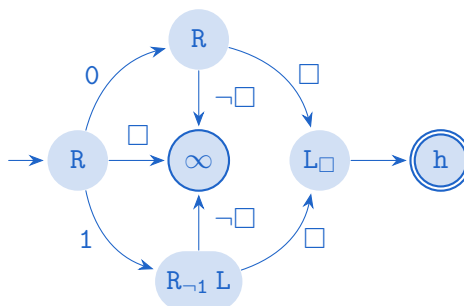
* 57. Beschouw een willekeurige EDA M . Geef een Turingmachine die de taal $\mathcal{L}(M)$ beslist.

Reeks 8

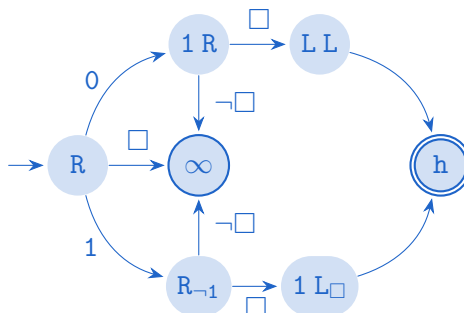
- * 58. Onderstel dat een Turingmachine niet alleen de instructies \leftarrow en \rightarrow kent, maar ook de instructie \perp die uitdrukt dat de lees-/schrijfkop blijft staan. Toon aan dat zo een Turingmachine omgezet kan worden in een Turingmachine waarin de kop enkel \leftarrow en \rightarrow kent. *Opmerking: de Turingmachines in JFLAP hebben een optie S (stay) voor de lees-/schrijfkop.*
- * 59. In de definitie wat het betekent voor een Turingmachine M om een functie f te berekenen, hebben we geëist dat M maar één stoptoestand heeft. Toon nu aan dat M ook meerdere stoptoestanden mag hebben. Met andere woorden, bewijs dat volgende twee uitspraken equivalent zijn, gegeven een functie f met domein A :
- Er bestaat een Turingmachine M met starttoestand s en juist één stoptoestand h zodat voor elke $w \in \Sigma^*$ geldt: als $w \in A$, dan $(s, \sqcup w) \vdash_M^* (h, \sqcup f(w))$, en als $w \notin A$, dan stopt M niet.
 - Er bestaat een Turingmachine M' met starttoestand s' zodat voor elke input $w \in \Sigma^*$ geldt: als $w \in A$, dan bestaat er een stoptoestand h met $(s', \sqcup w) \vdash_{M'}^* (h, \sqcup f(w))$, en als $w \notin A$, dan stopt M' niet.
60. Gebruik de unaire representatie voor de natuurlijke getallen: het woord 0 stelt het getal nul voor, het woord $1^n \in \{1\}^+$ het getal n .

1. Geef een Turingmachine die de identiteitsfunctie $f(n) = n$ berekent.

Veel moeten we niet doen, maar we moeten wel controleren of de input van de goede vorm is; als we onverwachte invoer tegenkomen, moet de machine in een oneindige lus belanden.

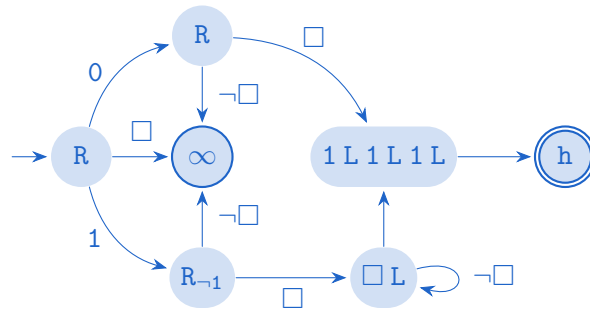


2. Geef een Turingmachine die de functie $f(n) = n + 1$ berekent.



3. Zij $c \in \mathbb{N}$ willekeurig; geef een Turingmachine die de constante functie $f(n) = c$ berekent.

We illustreren het geval $c = 3$. Een andere constante werkt volledig analoog; er hoeft slechts één subroutine aangepast te worden (op de voor de hand liggende manier).

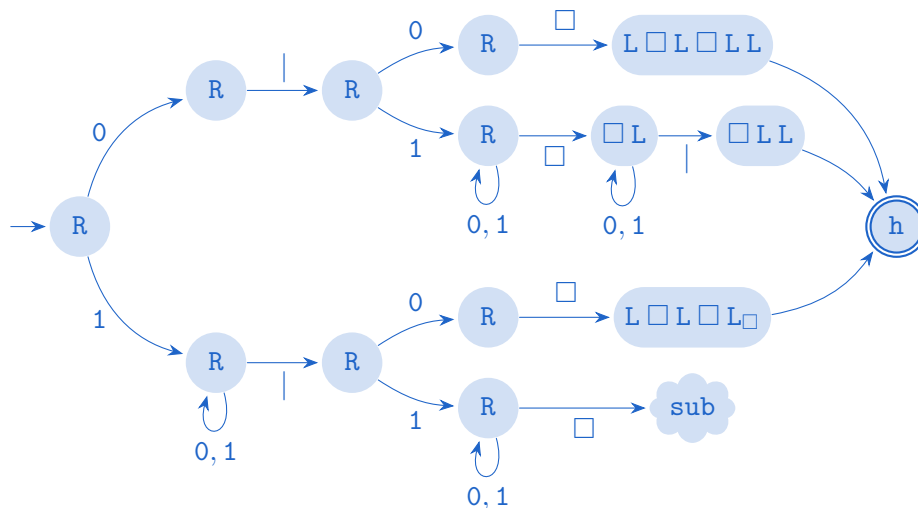


61. Geef een Turingmachine M (met één band) die de volgende functie monus berekent, waarbij we de natuurlijke getallen binair voorstellen.

$$\text{monus}(m, n) = \begin{cases} m - n & \text{als } m > n, \\ 0 & \text{anders.} \end{cases}$$

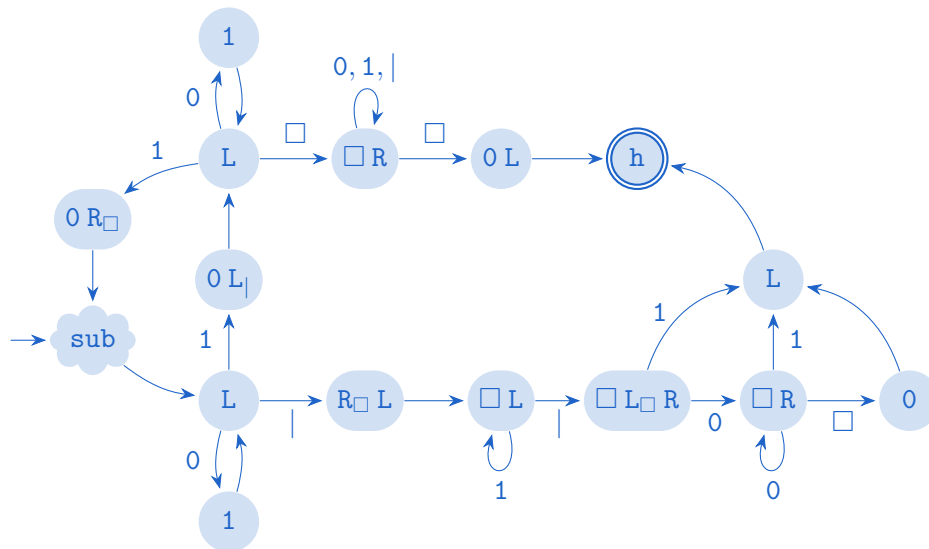
Bijvoorbeeld: als M vertrekt op bandinhoud ($\square 101 \mid 11$), dan moet M eindigen op ($\square 10$).

Eerst controleren we of de input van de juiste vorm is; we laten geen leidende nullen toe. Wanneer $m = 0$ of $n = 0$, is de vereiste output niet moeilijk te genereren en geven we deze meteen terug, maar wanneer $m > 0$ én $n > 0$ hebben we nog wat werk voor de boeg, in een subroutine sub.



Eenmaal in sub beland, trachten we $m - n$ te berekenen door telkens van zowel m als n één af te trekken tot een van de beide nul wordt, steunend op de observatie dat $m - n = (m - 1) - (n - 1)$. Als tijdens de poging om één af te trekken van de huidige n deze reeds nul blijkt, dan weten we dat de huidige m de gezochte uitkomst is. Blijkt m nul, dan is de uitkomst ook nul (ongeacht n).

Binaire getallen met één aftrekken is best eenvoudig: een woord $w10^k$ (met $w \in \{0, 1\}^*$ en $k \geq 0$) moeten we omzetten in $w01^k$ (mocht er geen 1 in staan, is het getal reeds nul). Uiteindelijk leidt dit tot de volgende lus.



- * 62. Geef een Turingmachine (met één band) die de volgende functie `subtract3` berekent, waarbij we de natuurlijke getallen binair voorstellen.

$$\text{subtract3}(n) = \begin{cases} n - 3 & \text{als } n \geq 3, \\ 0 & \text{anders.} \end{cases}$$

- * 63. Beschrijf een Turingmachine (met één band) die de som berekent van twee natuurlijke getallen, binair voorgesteld. Bijvoorbeeld: de invoer ($\square 101 \mid 11$) moet verwerkt worden tot ($\square 1000$).
64. Gebruik de unaire representatie voor de natuurlijke getallen: het woord 0 stelt het getal nul voor, het woord $1^n \in \{1\}^+$ het getal n . Geef een Turingmachine die de faculteit $n!$ berekent.

We pakken dit als volgt aan. Beschouw bij wijze van voorbeeld de input (1111). In eerste instantie willen we $4! = 4 \cdot 3 \cdot 2 \cdot 1$ berekenen door te beginnen met een enkele 1, die te verviervoudigen, daarna te verdrievoudigen, en tot slot nog eens te verdubbelen. Merk op dat een getal in unaire voorstelling *ver- n -voudigen* neerkomt op dit getal $n - 1$ keer kopiëren. Daarom vormen we eerst de input om tot ($\square 111 \mid 1$), waar het linkergetal *drie* betekent dat we het rechtergetal nog drie keer moeten kopiëren (om dus *vier* symbolen 1 te bekommen). Als dit gelukt is, willen we ($\square \square 11 \mid 1111$) bekommen, en moeten we dus 1111 nog *twee* keer kopiëren. Dan staat er

$$(\square \square \square 1 \mid 1111 \ 1111 \ 1111)$$

(de spaties staan hier opnieuw enkel voor de duidelijkheid), en tot slot kopiëren we het rechtergetal nog één keer tot

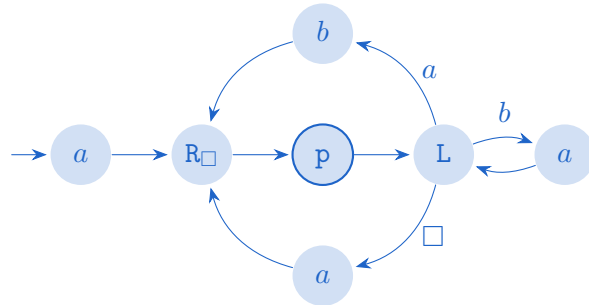
$$(\square \square \square \square \mid 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111).$$

Op dit moment staan er rechts $4 \cdot 3 \cdot 2 = 4!$ symbolen 1 en hoeven we enkel nog de puntkomma te verwijderen.

Hoe implementeren we dit kopiëren nu concreet? We zullen stap voor stap een 1 links markeren (met een \bullet) en daarna rechts een kopie aanmaken (in symbolen \circ om het onderscheid te behouden met de originele symbolen 1 die we moeten kopiëren). Voor iedere kopie moeten we telkens een 1 rechts markeren (we gebruiken daarvoor eveneens \bullet) en daarna een \circ bijschrijven, totdat elke 1 gemarkeerd is. Dan resetten we de rechtse \bullet 's terug tot 1's, markeren we een volgende linkse 1, en maken we rechts een volgende kopie aan. Als alle 1 gemarkeerd zijn links, werd het rechtergedeelte zoveel keer gekopieerd als nodig en kunnen we aan de volgende stap beginnen, nadat we de \bullet 's en \circ 's terug op 1 stellen en één 1 links verwijderen. De eerste stappen worden

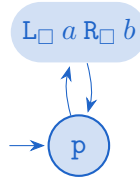
huidige woord wab^k (met $w \in \{a, b\}^*$ en $k \geq 0$) ombouwen naar het volgende woord wba^k terwijl we van b^k het woord a^{k+1} maken.

Verklaar zelf waarom deze procedure werkt, je inspirerend op het binair optellen van getallen.



2. $\{a^n b^n : n \geq 0\}$

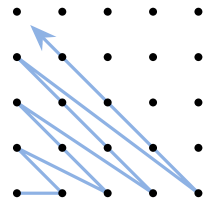
We voegen telkens links een a en rechts een b toe.



3. $\{a^n b^n c^m : n, m \geq 1\}$

We moeten de koppels natuurlijke getallen (m, n) proberen overlopen op zodanige wijze dat we ieder volgend koppel zonder veel moeite kunnen afleiden uit het huidige. Zoals iemand in de oefeningenles opmerkte, bestaat er een vrij eenvoudige methode om deze aan te pakken, door in elke stap het volgende te doen (vertrekkend vanuit $(1, 1)$):

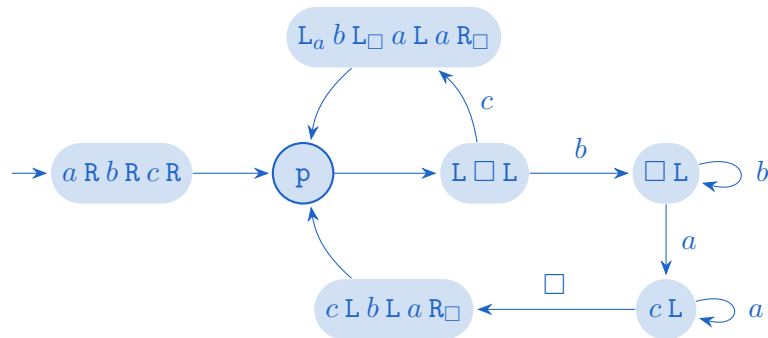
$$(m, n) \mapsto \begin{cases} (m - 1, n + 1) & \text{als } n > 1, \\ (m + 1, 1) & \text{als } n = 1. \end{cases}$$



Met andere woorden, we zullen de woorden als volgt opsommen:

$$abc, abcc, aabbc, abccc, \dots, abc^k, aabbc^{k-1}, \dots, a^k b^k c, abc^{k+1}, \dots$$

De volgende Turingmachine klaart de klus.



- * 4. $\mathcal{L}((ab)^+c)$
- * 5. $\{w \in \{a, b\}^* : \#_a(w) = \#_b(w) \geq 1\}$
- * 6. $\{a^n b^{2n} c^n : n \geq 1\}$

* 68. Onderstel dat de functie $f : \mathbb{N} \rightarrow \mathbb{N}$ Turingberekenbaar is.

- Bewijs dat de taal $\mathcal{L} = \{a^{f(n)} : n \in \mathbb{N}\}$ semibeslisbaar is.
- Pas je bewijs aan voor berekenbare $f : \mathbb{N}^2 \rightarrow \mathbb{N}$, met bijhorende taal $\{a^{f(m,n)} : m, n \in \mathbb{N}\}$.
- Onderstel daarenboven dat f een *strikt* stijgende functie is, dus dat $f(m) < f(n)$ als $m < n$. Bewijs dat \mathcal{L} beslisbaar is.
- Blijft \mathcal{L} beslisbaar als f slechts *monotoon* stijgend is (dus $f(m) \leq f(n)$ als $m < n$)?
- ⚡ • Kun je bewijzen dat \mathcal{L} niet noodzakelijk beslisbaar hoeft te zijn, ook al is f berekenbaar?

Hint: beschouw bijvoorbeeld de functie

$$f : \mathbb{N}^2 \rightarrow \mathbb{N} : f(m, n) = \begin{cases} m & \text{als } m = \langle M \rangle \text{ een Turingmachine codeert} \\ & \text{die in hoogstens } n \text{ stappen stopt op input } \varepsilon, \\ 0 & \text{anders.} \end{cases}$$

Reeks 9

* 69. Geef een Turingmachine met meerdere banden die de inhoud tussen twee blanco's \square op band i kopieert naar band j .

70. Geef een Turingmachine M die de functie $\max_{a,b} : \{a, b\}^* \rightarrow \mathbb{N}$ berekent, met

$$\max_{a,b}(w) = \text{de unaire representatie voor het getal } \max(\#_a(w), \#_b(w)).$$

Vanuit $(\square abaab)$ moet M bijvoorbeeld eindigen op $(\square 111)$. Je mag meerdere banden gebruiken.

We omschrijven een Turingmachine met drie banden waar de invoer w op de eerste band staat.

1. Als de eerste band leeg blijkt, geef dan output 0 terug.
2. Lees het woord w op de eerste band in door de kop telkens één stap naar rechts te bewegen. Bij het lezen van een a schrijven we op de tweede band een marker (\bullet), bij een b schrijven we die op de derde band. Ondertussen verwijderen we de gelezen inhoud van de eerste band en zetten we op de tweede resp. derde band een stap naar rechts.
3. Wanneer de eerste band leeg is, staan er $\#_a(w)$ markers op band twee, $\#_b(w)$ op band drie. Loop nu de tweede en derde band tegelijk terug naar links af. Als we op de beide een marker vinden, dan schrijven we een 1 op de eerste band en gaan we overal een stap naar links. Als de tweede of derde band leeg blijkt, blijven we hetzelfde doen. Pas wanneer beide banden leeg blijken, stoppen we; dan staan er op de eerste band precies $\max(\#_a(w), \#_b(w))$ symbolen 1 met de kop er vlak voor.

De transitiefunctie is eenvoudig genoeg om concreet uit te schrijven. Hierbij is s de starttoestand, r de "inleestoestand" (uit stap 2), t de "terugkeertoestand" (uit stap 3), en h onze halting toestand. De toestand s' is een hulptoestand nodig om de ledige input te kunnen detecteren.

$$\delta(s, (\square, \square, \square)) = (s', (\square, \square, \square), (\rightarrow, \perp, \perp))$$

$$\delta(s', (\square, \square, \square)) = (h, (0, \square, \square), (\leftarrow, \perp, \perp))$$

$$\delta(s', (a, \square, \square)) = (r, (\square, \bullet, \square), (\rightarrow, \rightarrow, \perp))$$

$$\delta(s', (b, \square, \square)) = (r, (\square, \square, \bullet), (\rightarrow, \perp, \rightarrow))$$

$$\delta(r, (a, \square, \square)) = (r, (\square, \bullet, \square), (\rightarrow, \rightarrow, \perp))$$

$$\delta(r, (b, \square, \square)) = (r, (\square, \square, \bullet), (\rightarrow, \perp, \rightarrow))$$

$$\delta(r, (\square, \square, \square)) = (t, (\square, \square, \square), (\leftarrow, \leftarrow, \leftarrow))$$

$$\delta(t, (\square, \bullet, \bullet)) = (t, (1, \square, \square), (\leftarrow, \leftarrow, \leftarrow))$$

$$\delta(t, (\square, \bullet, \square)) = (t, (1, \square, \square), (\leftarrow, \leftarrow, \perp))$$

$$\delta(t, (\square, \square, \bullet)) = (t, (1, \square, \square), (\leftarrow, \perp, \leftarrow))$$

$$\delta(t, (\square, \square, \square)) = (h, (\square, \square, \square), (\perp, \perp, \perp))$$

Hoe zouden we deze Turingmachine moeten aanpassen om $\min(\#_a(w), \#_b(w))$ te berekenen?

Probeer ook eens een Turingmachine met slechts één band uit te werken voor dezelfde functie!

71. Beschouw de taal $\mathcal{L} = \{\langle M \rangle : \text{de Turingmachine } M \text{ aanvaardt minstens twee woorden}\}$.

1. Omschrijf (in woorden) een Turingmachine die \mathcal{L} semibeslist.

Controleer eerst of de input van de vorm $\langle M \rangle$ is. Indien niet, kunnen we meteen verwerpen. Indien wel, moeten we kunnen controleren of M ten minste twee woorden aanvaardt.

Een voor de hand liggende "oplossing" is om gewoon alle woorden (lexicografisch) af te gaan, totdat we twee woorden hebben gevonden die M aanvaardt. Helaas werkt dit niet, want het is mogelijk dat M voor bepaalde woorden in een oneindige lus belandt. Dit probleem kunnen we oplossen via het *dovetailing*-principe.

Som alle woorden $w_1, w_2, w_3, \dots \in \Sigma^*$ lexicografisch op, en voer de volgende lus uit.

- Laat M hoogstens één instructie draaien op het eerste woord w_1 .
- Laat M hoogstens één instructie draaien op w_2 en hoogstens twee instructies op w_1 .
- Laat M hoogstens één instructie draaien op w_3 , dan hoogstens twee instructies op w_2 en hoogstens drie instructies op w_1 .
- ...

Merk op dat we “hoogstens” schrijven, want het is uiteraard mogelijk dat M al vroeger stopt. Op die manier kunnen we alle woorden op alle aantallen instructies nagaan, zonder het risico in een oneindige lus te belanden. Aanvaardt de machine twee woorden w_i en w_j na m_i en m_j instructies, dan komen we dat in de $\max(i, j, m_i, m_j)$ -de iteratie van de lus te weten en dan kunnen we $\langle M \rangle$ aanvaarden. Zo niet, blijven we oneindig lang doorzoeken.

2. Beschouw de variant $\mathcal{L}' = \{\langle M \rangle : \text{de Turingmachine } M \text{ aanvaardt precies twee woorden}\}$. Leg intuïtief uit waarom deze taal niet semibeslisbaar is.

Het verschil met de vorige oefening is dat we daar een stopcriterium hadden, en nu niet. We kunnen (op semibeslisbare wijze) controleren of M *minstens* twee woorden aanvaardt, want zodra we twee woorden vinden, kunnen we stoppen met zoeken—het maakt niets uit wat M met de rest van de woorden doet.

Als we echter willen nagaan of M *precies* twee woorden aanvaardt, moeten we *alle* woorden controleren! Zelfs al vinden we twee woorden die M aanvaardt, we kunnen niet zonder meer besluiten dat $\langle M \rangle \in \mathcal{L}'$, want het is niet uitgesloten dat M nog een derde woord aanvaardt.

- * 3. Toon aan dat \mathcal{L} niet beslisbaar is.

72. Toon via een reductie (niet via de stelling van Rice) aan dat de volgende talen in $SD \setminus D$ zitten.

1. $\{\langle M, w \rangle : M \text{ weigert } w\}$

Construeer een machine die test of de input van de vorm $\langle M, w \rangle$ is, dan de werking van M op w simuleert, en aanvaardt als M het woord w weigert. Deze machine semibeslist de taal.

We gebruiken vervolgens een reductie om aan te tonen dat deze taal niet beslisbaar is, meer bepaald een reductie van het halting problem $\mathcal{H} = \{\langle M, w \rangle : M \text{ stopt op input } w\}$ naar \mathcal{L} . Daartoe willen we een Turingmachine construeren die de eigenschap “ M stopt op input w ” van een input $\langle M, w \rangle$ vertaalt naar een eigenschap “ M' weigert w' ” van een input $\langle M', w' \rangle$. We zouden dan een potentiële belisser voor de taal \mathcal{L} kunnen gebruiken om ook het halting-probleem op te lossen en een strijdigheid te bekomen.

Dus, we willen een Turingmachine R ontwerpen die $\langle M, w \rangle$ zodanig verwerkt tot $\langle M', w' \rangle$ dat M stopt op input w als en slechts als M' input w' weigert. Definieer die als volgt.

De machine R controleert allereerst of zijn input van de vorm $\langle M, w \rangle$ is. Indien niet, laat R de input ongemoeid; indien wel, stelt R de codering op van een nieuwe Turingmachine M' die het volgende doet:

- (1) maak de band leeg,
- (2) schrijf w op de band,
- (3) laat M lopen op de nieuwe input w ,
- (4) weiger (ongeacht de originele input).

Daarna geeft R als output de codering $\langle M', \varepsilon \rangle$ terug.

Merk op dat R dus (coderingen van) programmacodes inleest en teruggeeft. Overtuig jezelf ervan dat deze functionaliteit effectief op een Turingmachine geïmplementeerd kan worden, ook al is het zeker niet de meest aangename oefening om tot in alle details uit te werken!

De reductiemachine R voldoet inderdaad aan de gewenste eigenschappen (met $w' = \varepsilon$).

- Als $\langle M, w \rangle \in \mathcal{H}$, dus als M stopt op w , dan zal M' altijd weigeren (ongeacht de input), dus dan zal in het bijzonder M' input ε weigeren, dus $\langle M', \varepsilon \rangle \in \mathcal{L}$.
- Als $\langle M, w \rangle \notin \mathcal{H}$, dus als M niet stopt op w , dan blijft ook M' oneindig lang doorlopen (ongeacht de input) en zal in het bijzonder $\langle M', \varepsilon \rangle \notin \mathcal{L}$.

Kunnen we nu besluiten dat \mathcal{L} niet beslisbaar is? Veronderstel van wel, dus dat er een orakel-machine Ω zou bestaan die \mathcal{L} beslist. Dan beslist de samenstelling $\Omega(R(\langle M, w \rangle))$ het halting probleem, voor arbitraire $\langle M, w \rangle$! We weten echter dat \mathcal{H} onbeslisbaar is; een strijdigheid.

2. $\{\langle M \rangle : M \text{ aanvaardt minstens één input}\}$

Semibeslisbaarheid verloopt analoog als in oefening 71.

Voor de reductie (opnieuw van \mathcal{H} naar deze \mathcal{L}) beschouwen we de Turingmachine R die op input $\langle M, w \rangle$ een codering van een nieuwe machine M' opstelt die het volgende doet;

- (1) maak de band leeg,
- (2) schrijf w op de band,
- (3) laat M lopen op de nieuwe input w ,
- (4) aanvaard (ongeacht de originele input).

Tot slot geeft R de codering $\langle M' \rangle$ van deze nieuwe machine terug. Dan zal inderdaad gelden dat $R(\langle M, w \rangle) = \langle M' \rangle \in \mathcal{L}$ als en slechts als $\langle M, w \rangle \in \mathcal{H}$. We besluiten dat \mathcal{L} onbeslisbaar moet zijn, want anders zouden we samen met R ook het halting probleem kunnen beslissen.

3. $\{\langle M_1, M_2, w \rangle : M_1 \text{ stopt op het lege woord en } M_2 \text{ aanvaardt } w\}$

In deze oefening worden twee condities opgegeven die allebei nogal onbeslisbaar uitschijnen. Het zal volstaan om te reduceren naar één van die condities; we kiezen de eerste.

We definiëren de reductie R vanuit de niet-beslisbare taal $\mathcal{H}_\varepsilon = \{\langle M \rangle : M \text{ stopt op input } \varepsilon\}$ door een input $\langle M \rangle$ te verwerken tot $\langle M, M', \varepsilon \rangle$, waar M' een machine voorstelt die vanuit de starttoestand meteen aanvaardt (ongeacht de input).

Dan zal $R(\langle M \rangle) = \langle M, M', \varepsilon \rangle \in \mathcal{L}$ als en slechts als M stopt op het ledige woord, of met andere woorden als en slechts als $M \in \mathcal{H}_\varepsilon$. We weten dat deze laatste taal onbeslisbaar is.

* 4. $\{\langle M, v, w \rangle : M \text{ aanvaardt } vw\}$

* 5. $\{\langle M_1, M_2 \rangle : \text{er bestaat minstens één input waarop zowel } M_1 \text{ als } M_2 \text{ stoppen}\}$

73. Voor reguliere talen kennen we een algoritme dat beslist of een gegeven taal al dan niet eindig is. Waarom is dit niet in tegenspraak met de stelling van Rice?

De stelling van Rice heeft betrekking op *alle* Turingmachines, niet op eindigetoestandsautomaten!

Ook al kunnen we een EDA als een bijzondere Turingmachine zien, die gewoon telkens een letter van z'n input leest en naar rechts opschuift (zonder de band aan te passen), dan nog zegt de stelling van Rice enkel iets over eigenschappen van *talen* (onder het volledige bereik van Turingmachines). De taal

$$\{\langle M \rangle : \mathcal{L}(M) \text{ is eindig}\}$$

is wegens Rice inderdaad niet beslisbaar, maar over de taal

$$\{\langle M \rangle : M \text{ stelt een EDA voor en } \mathcal{L}(M) \text{ is eindig}\}$$

kan Rice niets zeggen.

Anders gezegd, het gegeven dat M van een heel specifieke vorm is (die van een EDA) kunnen we gebruiken om de taal $\{\langle M \rangle : \mathcal{L}(M) \text{ is eindig}\}$ *bepert tot EDA's* te beslissen. Bij de algemene taal voor *alle Turingmachines* hebben we echter geen zo'n informatie die we kunnen benutten.

74. Bestaat er een algoritme voor recursief opsombare talen dat bepaalt of een taal het woord

$$10001 (0111110)^{2018} 10001$$

bevat?

Nee. De vraag is of de taal $\{\langle M \rangle : 10001 (0111110)^{2018} 10001 \in \mathcal{L}(M)\}$ al dan niet beslisbaar is. “Het woord $10001 (0111110)^{2018} 10001$ bevatten” is een niet-triviale eigenschap op de verzameling van recursief opsombare talen. Wegens de stelling van Rice is deze taal dus niet beslisbaar.

Merk op dat deze taal wél semibeslisbaar is; toon dit zelf aan.

* 75. Bestaat er een algoritme voor recursief opsombare talen dat bepaalt of een taal een singleton is?

* 76. Bestaat er een algoritme voor recursief opsombare talen dat bepaalt of een taal contextvrij is?

⚡ 77. De volgende oefening gaat over het invloedrijke *tiende probleem van Hilbert*. In 1900 stelde Hilbert een lijst voor met 23 belangrijke wiskundige problemen, die volgens hem de loop van de wiskunde in de twintigste eeuw zouden bepalen. Nummer tien op zijn lijst was de vraag naar een algoritme voor de volgende taak.

Gegeven een veeltermvergelijking met gehele coëfficiënten, in een onbepaald aantal variabelen; bepaal of deze vergelijking oplossingen heeft over de gehele getallen.

Het was mede door dit probleem dat wiskundigen⁽²⁾ op zoek gingen naar een exacte betekenis van “algoritme” en aldus de berekenbaarheidstheorie inluidden. Merk op dat een veeltermvergelijking in een enkele onbekende x steeds gecodeerd kan worden als een woord in $\{0, 1, x, +, -, \times, =\}^*$ met coëfficiënten in unaire vorm. Meerdere variabelen kunnen we coderen als $x1, x11, x111$, etc. De vergelijking $3x^2 - 2xy + y^2z = 7$ wordt dan bijvoorbeeld voorgesteld als

$$111 \times x1 \times x1 - 11 \times x1 \times x11 + 1 \times x11 \times x11 \times x111 = 1111111$$

of zelfs kortweg (aangezien er toch geen ambiguïteit mogelijk is) als

$$111x1x1 - 11x1x11 + 1x11x11x111 = 1111111.$$

Zij \mathcal{L} de taal van alle woorden die een veeltermvergelijking voorstellen met een oplossing over \mathbb{Z}^k en \mathcal{L}' de deeltaal van al deze vergelijkingen in één enkele variabele.

1. Beschrijf in woorden waarom \mathcal{L} en \mathcal{L}' semibeslisbaar zijn.

2. Toon aan dat \mathcal{L}' beslisbaar is. *Hint: zoek een onder- en bovengrens op de mogelijke oplossingen van een veeltermvergelijking in één onbekende in functie van de coëfficiënten en de graad.*

Pas in 1970 werd bewezen (door Yuri Matiyasevich, Julia Robinson, Martin Davis, Hilary Putnam) dat \mathcal{L} onbeslisbaar is, en het gezochte algoritme in de vraag van Hilbert dus niet kan bestaan.

⁽²⁾In het bijzonder Alan Turing, Alonzo Church, Stephen Kleene, Kurt Gödel, Rózsa Péter, Emil Post.

Reeks 10

78. Beschouw de taal $\mathcal{L} = \{\xi\}$, waarin $\xi = 0$ als $P = NP$, of $\xi = 1$ als $P \neq NP$. Is \mathcal{L} beslisbaar?

Deze taal is zeer zeker beslisbaar, want \mathcal{L} is eindig!

Deze opgave is misschien een strikvraag, maar wijst wel een belangrijke subtiliteit uit omtrent het concept "berekenbaarheid". Een taal is (per definitie) beslisbaar als er een Turingmachine bestaat die de taal beslist. Dat we niet weten welke machine dit zou moeten zijn, doet niks ter zake!

In deze situatie kunnen we makkelijk twee Turingmachines construeren, één die de taal $\{0\}$ beslist en één die de taal $\{1\}$ beslist. Een van deze twee is een beslisser voor \mathcal{L} dus is deze taal beslisbaar. Welke van de twee is het? Daar hebben we tot op vandaag het raden naar!

* 79. Bewijs dat de complexiteitsklasse P gesloten is onder eindige unies en doorsnedes.

80. Bewijs dat de complexiteitsklasse P gesloten is onder de Kleene-ster.

Zij $\mathcal{L} \in P$. We moeten aantonen dat ook $\mathcal{L}^* \in P$.

Een voor de hand liggend idee is om voor alle decomposities van w te verifiëren of elk deelwoord in \mathcal{L} zit. Dit zal niet werken, want er bestaan $2^{|w|-1}$ decomposities van w in niet-ledige woorden; de totale tijdscomplexiteit zal dus niet langer polynomiaal zijn in $|w|$ maar exponentieel.

We gebruiken het idee van dynamisch programmeren in de plaats. Stel de input $w = x_1x_2 \cdots x_n$ met elke $x_i \in \Sigma$. We kunnen via dynamisch programmeren de volgende verzameling A berekenen, in polynomiale tijd:

$$A = \{i : \text{de prefix van } w \text{ met lengte } i \text{ behoort tot } \mathcal{L}^*\}.$$

Uiteindelijk is $w \in \mathcal{L}^*$ als en slechts als $|w| = n \in A$. In pseudocode loopt het algoritme als volgt; overtuig jezelf dat al deze stappen op een Turingmachine geïmplementeerd kunnen worden.

```
A = {0}
for i in {1, ..., n}:
    for j in {1, ..., i}:
        if j-1 in A and x_j ... x_i in L:
            A = A union {i}
if n in A:
    return True
else:
    return False
```

Merk op dat alle lussen en condities in dit programma polynomiale tijd in $|w|$ vereisen (het aantal oproepen of deelwoorden van w tot \mathcal{L} behoren, is kwadratisch in $|w|$). We houden niet bij welke componenten van w uiteindelijk in \mathcal{L} zitten, maar dat hoeft ook niet; het enige wat van belang is, zijn de indices in A .

81. Bewijs of geef een tegenvoorbeeld voor de volgende bewering: gegeven drie talen $\mathcal{L}_1 \subseteq \mathcal{L} \subseteq \mathcal{L}_2$, met $\mathcal{L}_1, \mathcal{L}_2 \in P$, dan zit ook $\mathcal{L} \in P$.

Niet correct. Beschouw bij wijze van tegenvoorbeeld de talen $\mathcal{L}_1 = \emptyset$ en $\mathcal{L}_2 = \Sigma^*$. Deze behoren duidelijk tot P (ze zijn zelfs in constante tijd te beslissen!), maar eender welke taal over Σ voldoet aan de inclusies $\emptyset \subseteq \mathcal{L} \subseteq \Sigma^*$. Eender welke $\mathcal{L} \notin P$ (zoals $\mathcal{L} = \mathcal{H}$) vervolledigt het tegenvoorbeeld.

82. Bewijs rechtstreeks vanuit de definitie dat de volgende taal NP-compleet is:

$$\{\langle M, w, n \rangle : M \text{ is een niet-deterministische machine die } w \text{ in hoogstens } n \text{ stappen aanvaardt}\}.$$

Hierbij stellen we $n \in \mathbb{N}$ voor in unaire notatie.

De taal behoort tot NP: op input $\langle M, w, n \rangle$ volstaat het immers op een universele Turingmachine n stappen van M op input w te simuleren. Aanvaardt deze berekening, dan aanvaardden we ook de

input $\langle M, w, n \rangle$, en anders verwerpen we. Er zijn meerdere mogelijkheden voor deze berekening, dus deze beslisser is eveneens niet-deterministisch. Merk op dat beslissen lukt in polynomiale tijd t.o.v. de invoergrootte $|\langle M, w, n \rangle|$, want dit lukt in tijd

$$\mathcal{O}(|\langle M, w, n \rangle|) \cdot \mathcal{O}(n) = \mathcal{O}(|\langle M, w, n \rangle|^2).$$

Vervolgens tonen we aan dat de taal NP-compleet is, m.a.w. dat eender welk NP-probleem ernaar kan worden gereduceerd. Beschouw daartoe een willekeurige taal $\mathcal{L} \in \text{NP}$. Per definitie bestaat er een niet-deterministische machine M en een veelterm p zodat M in tijd $p(|w|)$ beslist of $w \in \mathcal{L}$.

De reductie R gaat nu als volgt: op input w , bereken $p(|w|)$ en schrijf $\langle M, w, p(|w|) \rangle$ op de band. Inderdaad, dan is $w \in \mathcal{L}$ als en slechts als $\langle M, w, p(|w|) \rangle$ tot de taal uit de opgave behoort.

Men kan aantonen dat het berekenen van $p(|w|)$ in polynomiale tijd lukt, dus de reductie zelf kost polynomiale tijd (constante tijd voor M , lineaire tijd voor w en polynomiale tijd voor $p(|w|)$).

Opmerking: we hoeven (de codering van) de Turingmachine M niet live te "berekenen" op de machine die de reductie uitvoert. We krijgen op voorhand een taal \mathcal{L} gegeven, waarvan we weten dat er een M bestaat, en deze M kunnen we hardcoden in de reductiemachine. In het bijzonder claimen we niet dat het makkelijk is om $\langle M \rangle$ expliciet te vinden uit een gegeven NP-taal \mathcal{L} , of dat een Turingmachine dit zou kunnen, maar dat hoeft ook niet; het volstaat gewoon te weten dat M bestaat.

83. Toon aan dat het volgende probleem tot P behoort:

$$\text{EULERIAN-CYCLE} = \{ \langle G \rangle : G \text{ is een Euleriaanse graaf} \}.$$

Herinner je dat een graaf Euleriaans wordt genoemd als er een cykel bestaat die alle bogen precies één keer aandoet.

Een *samenhangende* graaf is Euleriaans als en slechts als elke top even graad heeft. Een algemene graaf is Euleriaans als deze samenhangend Euleriaans is, op eventueel enkele geïsoleerde toppen na. Deze karakterisaties zijn in polynomiale tijd te controleren; de pariteit van een graad checken spreekt voor zich, voor de samenhang verwijzen we naar de cursus.

84. Toon aan dat het volgende probleem NP-compleet is:

$$3\text{SAT} = \{ \langle \phi \rangle : \phi \text{ is een vervulbare Booleaanse expressie in cnf met drie literalen per clause} \}.$$

Je mag steunen op de stelling van Cook–Levin dat SAT en CNF-SAT beide NP-compleet zijn, met

$$\text{CNF-SAT} = \{ \langle \phi \rangle : \phi \text{ is een vervulbare Booleaanse expressie in conjunctieve normaalvorm} \}.$$

We dienen drie beweringen aan te tonen: dat 3SAT in NP zit, dat CNF-SAT reduceert naar 3SAT, en dat deze reductie in polynomiale tijd lukt.

3SAT behoort tot NP.

Dit werd besproken in de theorie. We kunnen dezelfde *decider* of *verifier* als voor SAT loslaten op de taal, en hoeven alleen bijkomend te controleren of de input in conjunctieve normaalvorm staat met drie literalen per clause, maar deze extra controle kost slechts lineaire tijd.

CNF-SAT \leq_m 3SAT.

We zoeken een constructie die een gegeven Booleaanse formule ϕ transformeert in een formule met drie literalen per clause. Noteer $\phi = \rho_1 \wedge \rho_2 \wedge \dots \wedge \rho_n$ waar de ρ_i de clauses zijn. Elke ρ_i zullen we omzetten naar een conjunctie van nieuwe clauses met drie literalen, met als variabelen enerzijds de variabelen uit ρ_i en anderzijds nieuwe variabelen die enkel in deze nieuwe clauses optreden, met als enige doel garanderen dat deze nieuwe formule dezelfde vervulbaarheid heeft. De precieze constructie hangt af van het aantal literalen in ρ_i dus schrijf $\rho_i = \lambda_{i1} \vee \lambda_{i2} \vee \dots \vee \lambda_{ik_i}$ (met λ_{ij} de literalen) en maak volgend onderscheid naargelang de waarde van $k = k_i$.

$k = 1$. Voor een clause $\rho_i = \lambda_{i1}$ met één litaal voeren we twee hulpvariabelen z_{i1} en z_{i2} in. Vorm de nieuwe clauses

$$\tilde{\rho}_i = (\lambda_{i1} \vee z_{i1} \vee z_{i2}) \wedge (\lambda_{i1} \vee \neg z_{i1} \vee z_{i2}) \wedge (\lambda_{i1} \vee z_{i1} \vee \neg z_{i2}) \wedge (\lambda_{i1} \vee \neg z_{i1} \vee \neg z_{i2}).$$

$k = 2$. Voor een clause $\rho_i = \lambda_{i1} \vee \lambda_{i2}$ met twee literalen voeren we één hulpvariabele z_{i1} in. Vorm de nieuwe clauses

$$\tilde{\rho}_i = (\lambda_{i1} \vee \lambda_{i2} \vee z_{i1}) \wedge (\lambda_{i1} \vee \lambda_{i2} \vee \neg z_{i1}).$$

$k = 3$. Voor een clause $\rho_i = \lambda_{i1} \vee \lambda_{i2} \vee \lambda_{i3}$ met drie literalen hoeven we niks te doen, dus stel dan gewoon $\tilde{\rho}_i = \rho_i$.

$k > 3$. Voor een clause $\rho_i = \lambda_{i1} \vee \lambda_{i2} \vee \dots \vee \lambda_{ik}$ met méér dan drie literalen voeren we $k - 3$ hulpvariabelen $z_{i1}, \dots, z_{i,k-3}$ in. Vorm de nieuwe clauses

$$\begin{aligned} \tilde{\rho}_i = & (\lambda_{i1} \vee \lambda_{i2} \vee z_{i1}) \wedge (\neg z_{i1} \vee \lambda_{i3} \vee z_{i2}) \wedge (\neg z_{i2} \vee \lambda_{i4} \vee z_{i3}) \wedge \dots \\ & \wedge (\neg z_{i,k-4} \vee \lambda_{i,k-2} \vee z_{i,k-3}) \wedge (\neg z_{i,k-3} \vee \lambda_{i,k-1} \vee \lambda_{ik}). \end{aligned}$$

Bijvoorbeeld, de formule $\phi = (\neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3 \vee \neg x_1)$ wordt omgebouwd tot

$$\tilde{\phi} = (\neg x_2 \vee x_3 \vee z_{11}) \wedge (\neg x_2 \vee x_3 \vee \neg z_{11}) \wedge (x_1 \vee x_2 \vee z_{21}) \wedge (\neg z_{21} \vee x_3 \vee \neg x_1).$$

Merk nu op dat de originele formule $\phi = \rho_1 \wedge \rho_2 \wedge \dots \wedge \rho_n$ vervulbaar is als en slechts als de nieuwe formule $\tilde{\phi} = \tilde{\rho}_1 \wedge \tilde{\rho}_2 \wedge \dots \wedge \tilde{\rho}_n$ vervulbaar is. Voor de constructies bij $k \leq 3$ moet dat duidelijk zijn; het geval $k > 3$ vergt een argumentje.

Als ρ_i (met $k_i > 3$) de waarde True krijgt na een invulling van de variabelen, dan bestaat er een litaal λ_{im} in deze clause (met $1 \leq m \leq k$) die True blijkt. Stel dan alle hulpvariabelen z_{ij} met $j \leq m - 2$ (in de clause vóór die met λ_{im}) op True en alle hulpvariabelen z_{ij} met $j \geq m - 1$ (in de clause ná die met λ_{im}) op False. Alle voorgaande en alle volgende clauses worden dan eveneens True, dus we hebben de vervullende invulling van ρ_i uitgebreid naar een vervullende invulling van $\tilde{\rho}_i$ (zonder effect op de overige clauses).

Omgekeerd, als na invulling van de variabelen $\tilde{\rho}_i$ de waarde True krijgt, dan kunnen we gewoon de variabelen in ρ_i dezelfde waarde toekennen als in $\tilde{\rho}_i$. Deze invulling stelt dan ook ρ_i op True.

In elk geval blijkt ϕ vervulbaar als en slechts als $\tilde{\phi}$ vervulbaar is, en bovendien is $\tilde{\phi}$ een formule in conjunctieve normaalvorm met drie literalen per clause, hetgeen de reductie voltooit.

CNF-SAT \leq_P 3SAT.

Tot slot moeten we nog beargumenteren dat deze constructie in polynomiale tijd (ten opzicht van de inputgrootte $|\langle \phi \rangle|$) kan. Merk op dat in de gevallen $k = 1$, $k = 2$ en $k = 3$, een clause wordt omgevormd naar respectievelijk vier, twee en één clauses, zodat de lengte lineair toeneemt. Voor een clause met $k > 3$ literalen bouwen we $k - 2$ nieuwe clauses, zodat de lengte kwadratisch toeneemt. Samen kost het opstellen van de nieuwe formule polynomiale tijd.

* 85. Toon aan dat DNF-SAT tot P behoort.

$$\text{DNF-SAT} = \{ \langle \phi \rangle : \phi \text{ is een vervulbare Booleaanse expressie in disjunctieve normaalvorm} \}.$$

Een formule in disjunctieve normaalvorm is van de vorm

$$(\bullet \wedge \bullet \wedge \dots \wedge \bullet) \vee (\bullet \wedge \dots \wedge \bullet) \vee \dots \vee (\bullet \wedge \dots \wedge \bullet).$$

Er bestaan algoritmes om formules in conjunctieve normaalvorm om te vormen naar disjunctieve normaalvorm. We weten daarenboven dat CNF-SAT NP-compleet is, terwijl DNF-SAT dus tot P behoort. Waarom volgt hieruit *niet* dat $P = NP$?

* 86. Toon aan dat 2SAT (volgens de voor de hand liggende definitie) tot P behoort.

* 87. Toon aan dat het volgende probleem NP-compleet is, via een reductie vanuit 3SAT:

$$\text{CLIQUE} = \{ \langle G, k \rangle : G \text{ is een graaf met een } k\text{-kliek} \}.$$

Herinner je dat een k -klike in een graaf een verzameling van k onderling adjacenten toppen is.

- * 88. Toon aan dat het volgende probleem NP-compleet is, via een reductie vanuit CLIQUE:

$$\text{INDEPENDENT-SET} = \{ \langle G, k \rangle : G \text{ is een graaf met } k \text{ onderling niet-adjacente toppen} \}.$$

- * 89. Toon aan dat het volgende probleem NP-compleet is, via reductie vanuit INDEPENDENT-SET:

$$\text{VERTEX-COVER} = \{ \langle G, k \rangle : G \text{ is een graaf met een vertex cover van grootte } k \}.$$

Herinner je dat een *vertex cover* in een graaf een deelverzameling van toppen is met de eigenschap dat elke boog van de graaf met minstens één zo'n top incident is.

- * 90. Toon aan dat het volgende probleem NP-compleet is, via reductie vanuit VERTEX-COVER:

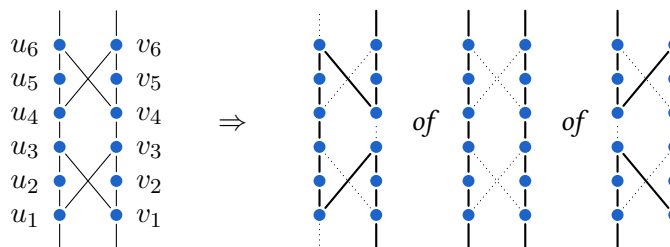
$$\text{HITTING-SET} = \left\{ \langle n, m, C_1, \dots, C_m, k \rangle : \begin{array}{l} C_i \subseteq \{1, 2, \dots, n\} \text{ voor elke } i \\ \text{en er bestaat een } S \subseteq \{1, 2, \dots, n\} \\ \text{met } |S| = k \text{ en } S \cap C_i \neq \emptyset \text{ voor elke } i \end{array} \right\}.$$

- ⚡ 91. Toon aan dat het volgende probleem NP-compleet is, via een reductie vanuit VERTEX-COVER:

$$\text{HAMILTONIAN-CYCLE} = \{ \langle G \rangle : G \text{ is een Hamiltoniaanse graaf} \}.$$

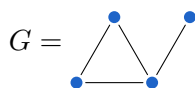
Herinner je dat een graaf Hamiltoniaans wordt genoemd als er een cykel bestaat die alle toppen precies één keer aandoet.

Hint: we schetsen deze reductie. Veronderstel gegeven een input $\langle G, k \rangle$; we dienen in polynomiale tijd een nieuwe graaf G' te construeren die Hamiltoniaans is als en slechts als G een k -cover heeft. Voor iedere boog (u, v) van G plaatsen we een component met twaalf toppen zoals hieronder links.

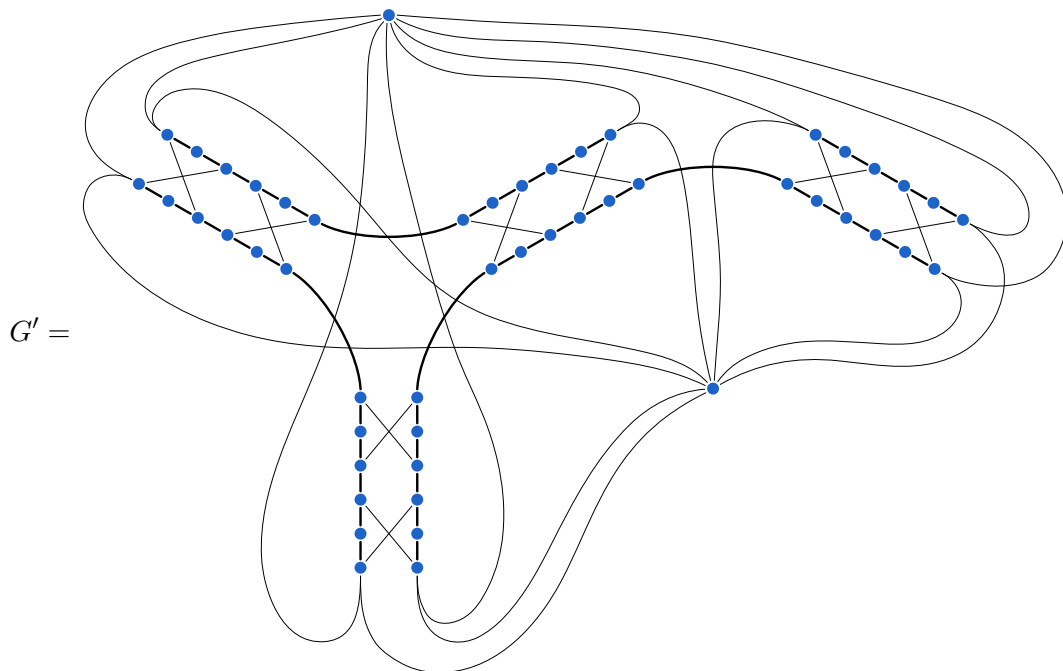


Merk nog op dat een Hamiltoncykel doorheen zo'n component enkel mogelijk is op de drie afgebeelde manieren rechts; in het bijzonder verlaat die iedere component langs dezelfde kant als binnengegaan.

Dan voegen we nog k speciale toppen toe. Voor elke top in G sluiten we de kanten van de componenten die met deze top overeenkomen in een lang pad aaneen, in een willekeurige volgorde, en beide eindpunten van dat pad verbinden we met ieder van deze k speciale toppen. Het resultaat noemen we G' . Hopelijk schept een voorbeeld meer duidelijkheid; voor de graaf



en parameterwaarde $k = 2$, geeft de constructie de volgende graaf



Hoe correspondeert een Hamiltoncykel in G' met een vertex cover in G ?

* 92. Toon aan dat het volgende probleem NP-compleet is, via reductie vanuit HAMILTONIAN-CYCLE:

$$\text{TSP} = \{ \langle G, k \rangle : G \text{ is een gewogen graaf met een Hamiltoncykel van totaal gewicht } \leq k \}.$$

Doorheen de laatste oefeningen hebben we aangetoond dat het handelsreizigersprobleem NP-compleet is, door de volgende keten van polynomiale reducties vanuit CNF-SAT.

$$\text{CNF-SAT} \leq_p \text{3SAT} \leq_p \text{CLIQUE} \leq_p \text{INDEPENDENT-SET} \\ \leq_p \text{VERTEX-COVER} \leq_p \text{HAMILTONIAN-CYCLE} \leq_p \text{TSP}$$

In 1972 publiceerde Richard Karp de beroemde paper *Reducibility among combinatorial problems* waarin hij vanuit 3SAT een twintigtal combinatorische en grafentheoretische problemen NP-compleet bewees. Dit artikel was een eerste demonstratie dat veel natuurlijk optredende problemen computationeel zeer zwaar kunnen zijn. Zie https://en.wikipedia.org/wiki/Karp%27s_21_NP-complete_problems voor meer informatie.

Extra's

- Talen van de vorm $\{\langle M \rangle : \mathcal{L}(M) \text{ voldoet aan zekere eigenschap}\}$ zijn onbeslisbaar (Rice).
Bijvoorbeeld: $\mathcal{L}(M)$ is eindig, $\mathcal{L}(M)$ bevat een woord van even lengte, $\mathcal{L}(M)$ is regulier, etc.
- Talen van de vorm $\{\langle M \rangle : \text{de structuur van } M \text{ voldoet aan zekere eigenschap}\}$ zijn beslisbaar.
Bijvoorbeeld: M heeft juist 42 toestanden, M heeft geen transitie terug naar de starttoestand, etc.
- Talen van de vorm $\{\langle M \rangle : \text{het gedrag van } M \text{ voldoet aan zekere eigenschap}\}$ kunnen wel of niet beslisbaar zijn, maar het is niet altijd evident het juiste antwoord te selecteren...

93. Welke van de volgende talen zijn beslisbaar? Semibeslisbaar? Bewijs.

1. $\{\langle M \rangle : \text{elk woord dat } M \text{ aanvaardt is een palindroom (een woord } w \text{ waarvoor } w = w^R)\}$,
Niet semibeslisbaar. We geven een reductie uit $\overline{\mathcal{H}}$ (het complement van het *halting problem*). De reductiemachine R werkt als volgt. Gegeven een input van de vorm $\langle M, w \rangle$, construeer de codering $\langle M' \rangle$ van een Turingmachine M' die het volgende doet:
 - i. wis de input,
 - ii. simuleer M op w ,
 - iii. aanvaard (ongeacht de originele gewiste input).

Als de input van R niet van de vorm $\langle M, w \rangle$ is, geef dan de codering $\langle M' \rangle$ van een Turingmachine M' terug die elke input verwerpt.

Merk nu op dat $\mathcal{L}(R(x))$ gelijk is aan de volledige taal Σ^* als $x = \langle M, w \rangle$ waarbij M stopt op w , maar ledig als $x = \langle M, w \rangle$ waarbij M niet stopt op w of als x niet van deze vorm is. Met andere woorden, $R(x)$ behoort tot de taal uit de opgave als en slechts als $x \in \overline{\mathcal{H}}$.

Dat de taal niet beslisbaar is, volgt ook meteen uit de stelling van Rice.

2. $\{\langle M \rangle : M \text{ is een Turingmachine die op alle input in hoogstens acht stappen stopt}\}$,
Beslisbaar. In hoogstens acht stappen kan een Turingmachine immers niet verder geraken dan acht cellen op de tape. Met andere woorden, aangezien we starten op de blanco cel vlak vóór de input, zijn slechts de eerst zeven symbolen van de input relevant. We kunnen dus M acht stappen laten lopen op alle woorden in Σ^* met lengte hoogstens zeven—dit zijn er slechts eindig veel! Als de machine op elk van deze woorden stopt in hoogstens acht stappen, dan kunnen we $\langle M \rangle$ aanvaarden, en anders verwerpen.
3. $\{w \in \{0, 1\}^* : w \text{ wordt aanvaard door } M\}$ voor een vaste Turingmachine M .

Semibeslisbaar voor algemene Turingmachines M , voor bepaalde machines zelfs beslisbaar.

Over het algemeen is deze taal semibeslisbaar (laat machine M de input w verwerken en kijk of deze aanvaardt). Voor sommige machines M kunnen we meer zeggen. Stel bijvoorbeeld dat we weten dat M altijd stopt, op elke input, dan is deze taal beslisbaar (verwerk w op M tot deze aanvaardt/verwerpt). Ook als we weten dat M net *nooit* stopt, is deze taal beslisbaar (want ledig).

Aan de andere kant, als we voor M de universele Turingmachine beschouwen, dan is de taal niet beslisbaar. De taal uit de opgave is dan immers precies de taal

$$\{w \in \{0, 1\}^* : w = \langle N, v \rangle \text{ en } v \text{ wordt aanvaard door } N\}$$

(het *acceptance problem*), en deze is zoals gezien in de theorie niet beslisbaar. Merk op dat N hier géén vaste Turingmachine is, maar wordt mee gespecificeerd in de input voor M .

94. Bewijs dat de klasse van beslisbare talen **niet** gesloten is onder lettersubstituties.

Hint: beschouw de taal

$$\left\{ xy : \begin{array}{l} x = \langle M, w \rangle \in \{0, 1\}^* \text{ codeert een Turingmachine } M \text{ en een inputwoord } w, \\ y \in \{\bullet\}^* \text{ codeert een unair getal } n, \text{ en } M \text{ stopt op } w \text{ in hoogstens } n \text{ stappen} \end{array} \right\}.$$

De taal is beslisbaar: we kunnen op een universele Turingmachine het gedrag van M op w simuleren, en na elke stap een \bullet schrappen. Wanneer alle \bullet 's op zijn en de simulatie nog niet afgerond is, verwerpen we de input; anders aanvaarden we.

Desalniettemin is het beeld van deze taal onder de lettersubstitutie $(0 \mapsto 0)$, $(1 \mapsto 1)$, $(\bullet \mapsto \varepsilon)$ net het halting problem \mathcal{H} , en we weten dat deze niet beslisbaar is.

⚡ 95. Herinner je dat Turingmachines volgens de standaarddefinitie telkens een stap naar links of rechts moeten zetten.

1. Bewijs dat de volgende taal **beslisbaar** is:

$$\{\langle M, w \rangle : M \text{ zet tijdens het verwerken van } w \text{ ooit een stap naar links}\}.$$

2. Bewijs dat de volgende taal **onbeslisbaar** is:

$$\{\langle M, w \rangle : M \text{ zet tijdens het verwerken van } w \text{ ooit drie opeenvolgende stappen naar links}\}.$$

Merk op dat M in beide gevallen de input w niet noodzakelijk moet aanvaarden (of zelfs stoppen) om tot de taal in kwestie te behoren!

1. Merk op dat als M tijdens het verwerken nooit een stap naar links zet, die de input doorleest en daarna enkel nog blanco cellen op de tape tegenkomt. Eenmaal op het blanco gedeelte kan de machine weliswaar nog van toestand veranderen en cellen beschrijven, maar daarna stapt die naar rechts en leest die toch weer een blanco cel. Met andere woorden, het gedrag van de machine wordt dan volledig bepaald door de toestanden (en valt te vergelijken met een EDA die een oneindig lange input $\square\square\square\cdots$ verwerkt). Omdat het aantal toestanden $|Q|$ van een Turingmachine eindig is, moet deze dan ofwel stoppen na hoogstens $|Q|$ stappen op de blanco tape, ofwel in een oneindige lus belanden.

Omgekeerd, als een arbitraire Turingmachine met $|Q|$ toestanden op input w na $|w| + |Q| + 1$ stappen enkel naar rechts gestapt is, dan heeft deze de input doorlezen en daarna méér dan $|Q|$ stappen afgelegd op blanco tapesymbolen. Volgens een argument á la pumping lemma is die dan in deze laatste $|Q| + 1$ stappen in een lus terechtgekomen; de machine zal deze lus dan blijven volgen, en in het bijzonder naar rechts blijven gaan.

Met andere woorden, we kunnen in eindig veel stappen beslissen of de Turingmachine *altijd* naar rechts blijft gaan, als volgt. Zij gegeven de input $\langle M, w \rangle$. Tel het aantal toestanden $|Q|$ van M en simuleer $|w| + |Q| + 1$ stappen van M op input w . Als M tijdens het verwerken een stap naar links zet, kunnen we de input aanvaarden; als M na zoveel stappen enkel naar rechts gegaan is, zal M blijven naar rechts gaan en kunnen we verwerpen.

2. We gebruiken een reductie vanuit het *halting problem*. Met andere woorden, we beschrijven een reductiemachine R die een input $\langle M, w \rangle$ verwerkt tot $\langle M', w' \rangle$, zodanig dat M stopt op w als en slechts als M' tijdens het verwerken van w' ooit drie stappen naar links zet.

De reductie werkt als volgt. Gegeven input $\langle M, w \rangle$, stel de codering op van de machine M' die het volgende doet:

- i. negeer de input;
- ii. simuleer M op input w , maar vervang elke transitieregel van de vorm

$$\delta(p, x) = (p', x', L)$$

door drie nieuwe regels

$$\delta(p, x) = (q, x', L)$$

$$\delta(q, *) = (q', *, L)$$

$$\delta(q', *) = (p', *, R)$$

waarin q en q' nieuwe toestanden zijn (voor elke p);

iii. start een oneindige lus op door telkens naar links te stappen.

Geef de codering $\langle M', \varepsilon \rangle$ terug.

Het trucje in stap ii. in de nieuwe machine M' zorgt ervoor dat elke stap L naar links wordt vervangen door drie stappen L L R die netto hetzelfde resultaat geven, maar vermijden dat we ooit drie opeenvolgende stappen naar links zetten. De constructie garandeert dus dat M' enkel drie stappen naar links kan zetten in stap iii., en dat treedt slechts op als in stap ii. de simulatie van M stopt.

Dus inderdaad, $R(\langle M, w \rangle)$ behoort tot de taal uit de opgave als en slechts als $\langle M, w \rangle$ tot het halting probleem behoort, en deze reductie toont de gevraagde onbeslisbaarheid aan.

96. Een taal \mathcal{L} noemen we—analoog als bij NP-complete talen—*P-compleet* als $\mathcal{L} \in P$ en als elke taal $\mathcal{L}' \in P$ reduceerbaar is tot \mathcal{L} in polynomiale tijd (dus als $\mathcal{L}' \leq_P \mathcal{L}$). Welke talen zijn P-compleet?

Examenvragen

97. Beschouw talen over het alfabet $\Sigma = \{a, b, c, \dots, z\}$, en beschouw de operator

$$\text{Laang}(\mathcal{L}) = \{w' : (\exists w \in \mathcal{L})(w' \text{ wordt bekomen uit } w \text{ door alle klinkers te verdubbelen})\}.$$

Bijvoorbeeld, als het woord *appel flap* tot \mathcal{L} behoort, dan zal $\text{Laang}(\mathcal{L})$ in de plaats daarvan het woord *aappeelflaap* bevatten.

1. Toon door constructie van een EDA aan dat de reguliere talen gesloten zijn onder *Laang*.
2. Volgt de bewering meteen uit een stelling die in de theorie aan bod is gekomen? Welke?

Deze *Laang* is vrij primitief: ze transformeert bijvoorbeeld *fiets* in *fieets*. We definiëren nu analoog de operator *Laang2* die alle opeenvolgingen van één of twee klinkers vervangt door de originele klinkers, gevolgd door opnieuw die originele klinkers. Dus *appel flap* wordt nog altijd *aappeelflaap* maar *fiets* wordt *fieiets*. Komen er opeenvolgingen van drie of meer klinkers voor, dan verwerken we deze opeenvolgingen van klinkers per groepje van twee (en eventueel de laatste overblijvende klinker apart), dus *zeeotter* wordt *zeeeeootteer*.

3. Is de verzameling van de reguliere talen gesloten onder *Laang2*?

We kunnen natuurlijk zo verder gaan. Beschouw nu de operatie *Laango* die alle maximale opeenvolgingen van één of meer klinkers vervangt door de originele klinkers, gevolgd door opnieuw die originele klinkers. Dus *zeeotter* wordt *zeeoeotteer* en *koeoog* wordt *koeooeooog*.

4. Is de verzameling van de reguliere talen gesloten onder *Laango*?

* 98. Zijn volgende talen regulier? Contextvrij? Bewijs.

1. $\{w \in \{a, b, c\}^* : (\#_a(w) + \#_b(w)) \cdot \#_c(w) = 6\}$,
2. $\{w \in \{0, 1, 2\}^* : \#_1(w) \bmod 3 = 1, 3\#_1(w) - \#_0(w) \geq 1 \text{ en } 221 \text{ is deelwoord}\}$,
3. $\{xyzy : x, y, z \in \{0, 1\}^+\}$.

* 99. Wat verandert er als we PDA's met oneindig veel toestanden toelaten; worden nog exact dezelfde talen geaccepteerd?

100. Gegeven de grammatica

$$\begin{cases} S \rightarrow aS \mid Tb, \\ T \rightarrow aUb, \\ U \rightarrow Ub \mid aU \mid S \mid \varepsilon. \end{cases}$$

Bestaat er een reguliere grammatica die dezelfde taal genereert? Indien wel, geef deze grammatica. Indien niet, waarom niet?

101. Voor een formele taal \mathcal{L} definiëren we $\text{Pow}(\mathcal{L}) = \{w^n : w \in \mathcal{L} \text{ en } n \geq 0\}$. Zijn de reguliere talen gesloten onder deze operatie *Pow*? Zelfde vraag voor de contextvrije talen.

* 102. Geef een Turingmachine met één band (in macronotatie) die de volgende taal opsomt:

$$\{1^a \# 1^b \# 1^c \# 1^d : a + b = c + d \text{ en } a, b, c, d \geq 1\}.$$

Bijvoorbeeld, $11 \# 11 \# 1 \# 111$ behoort tot deze taal, omdat $2 + 2 = 1 + 3$.

* 103. Toon via een reductie (dus zonder toepassing van de stelling van Rice) aan dat

$$\{\langle M, M', w \rangle : M \text{ aanvaardt } w \text{ en } M' \text{ weigert } w^R w\} \in \text{SD} \setminus \text{D}.$$

104. Toon via een reductie (dus zonder toepassing van de stelling van Rice) aan dat

$$\{\langle M \rangle : \text{de Turingmachine } M \text{ aanvaardt minstens alle woorden in de Van Dale}\} \in \text{SD} \setminus \text{D}.$$

We gebruiken uiteraard het alfabet $\Sigma = \{a, b, c, \dots, z\}$.

Opmerking: het langste woord in de Van Dale is "meervoudigepersoonlijkeidstoornissen" (38 letters).