Oefeningenbundel voor het vak
Automaten, berekenbaarheid & complexiteit

Jens Bossaert
jens.bossaert@ugent.be
2018–2019

Dit is de oefeningenbundel voor het vak *Automaten, berekenbaarheid & complexiteit*, gegeven aan de Universiteit Gent, academiejaar 2018–2019, 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.

18/02: A3, **25/02:** A3, **04/03:** Zuse, **11/03:** Zuse, **18/03:** A3,
25/03: Zuse, **01/04:** A3, **29/04:** A3, **06/05:** A3, **13/05:** A3.

Wijzigingen hierin worden aangekondigd via Minerva.

1. We proberen de meeste opgaven tijdens de oefeningenlessen 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. 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.

UNIVERSITEIT
GENT

Verzamelingen en talen

* **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 op $|X|$)

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 schaap wit is, dan zijn deze allemaal wit.* Zowel de basisstap (voor $n = 1$) als de inductiestap worden eenvoudig bewezen; wat klopt er niet?

Het is verleidelijk om de volgende redenering op te bouwen.

> *Voor $n = 1$ is de opgave inderdaad duidelijk: als één schaap wit is, dan zijn alle schapen (namelijk die ene) wit. Onderstel de uitspraak geldig voor n en beschouw $n + 1$ schapen, waaronder minstens één witte, die wij Castor noemen. Haal een ander schaap, 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 schaap 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. $xxyzz$ niet.
   3. $xxyyz$ 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. $(A \cup B) \cdot C = (A \cdot C) \cup (B \cdot C)$ is correct. We tonen de inclusie in twee richtingen aan.

   Beschouw voor de inclusie $(A \cup B)\,C \subseteq AC \cup BC$ een arbitrair element w van $(A \cup B)\,C$. Dan is deze te schrijven als $w = xy$ met $x \in A \cup B$ en $y \in C$. Dus $x \in A$ of $x \in B$ (of beide) waaruit volgt dat $xy \in AC$ of $xy \in BC$ (of beide). Hoe dan ook is $xy = w$ dus bevat in de unie $AC \cup BC$.

   Beschouw voor de inclusie $(A \cup B)\,C \supseteq AC \cup BC$ een arbitrair element w van $AC \cup BC$. Dan is $w \in AC$ of $w \in BC$ (of beide). In het eerste geval kunnen we schrijven dat $w = xy$ met $x \in A \subseteq A \cup B$ en met $y \in C$, dus is $xy \in (A \cup B)\,C$. In het tweede geval kunnen we schrijven dat $w = xy$ met $x \in B \subseteq A \cup B$ en $y \in C$, en volgt hetzelfde. In beide gevallen blijkt inderdaad $w = xy \in (A \cup B)\,C$.

2. $(A \cap B) \cdot C = (A \cdot C) \cap (B \cdot C)$ is niet correct, zoals bijvoorbeeld uit $A = \{x^{2n} : n \in \mathbb{N}\}$, $B = \{x^{2n+1} : n \in \mathbb{N}\}$ en $C = \{\varepsilon, x\}$ blijkt: $(A \cap B)\,C = \emptyset$ terwijl $AC \cap BC = \{x\}^+$.

7. Geef voor de volgende talen een beknopte beschrijving.

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

De sluiting van een verzameling vind je intuïtief door herhaaldelijk de ontbrekende elementen toe te voegen, tot je een gesloten verzameling vasthebt. Soms lukt dit in eindig veel stappen (zoals in deelvraag 3); soms beland je in een oneindig proces (zoals in deelvraag 1).

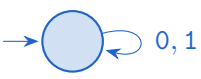
1. $\{a,b\}$ onder concatenatie is niet gesloten; de sluiting is $\{a,b\}^+$.
2. $\{a,b\}^*$ onder omkering is gesloten.
3. $\{w \in \{a,b\}^* : w$ begint met $a\}$ onder omkering is niet gesloten; de sluiting is $\{w \in \{a,b\}^* : w$ begint of eindigt met $a\}$.

Deterministische automaten

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

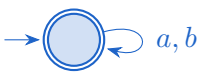
1. $\emptyset$ met alfabet $\Sigma = \{0,1\}$

   We geven een automaat die altijd verwerpt.

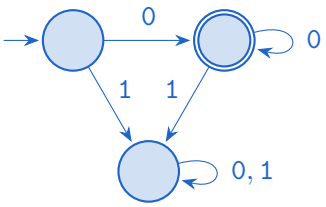
2. Σ^* met alfabet $\Sigma = \{a,b\}$

   We geven een automaat die altijd aanvaardt.

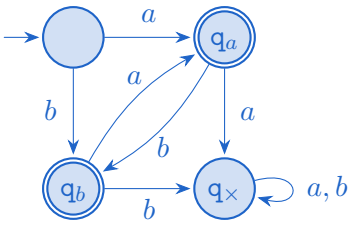
3. $\{w \in \{0,1\}^+ :$ geen enkele prefix van w eindigt op een $1\}$

   "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:



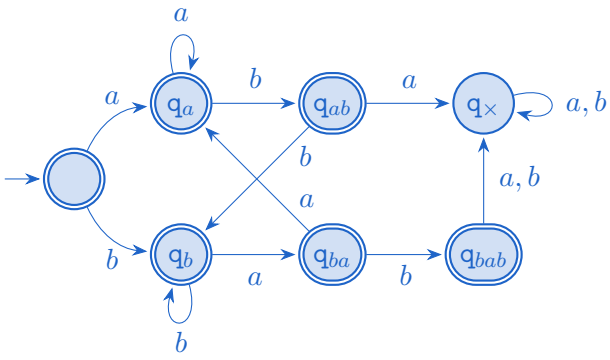
4. $\{w \in \{a,b\}^+ : w$ bestaat uit alternerende a's en b's$\}$

   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.



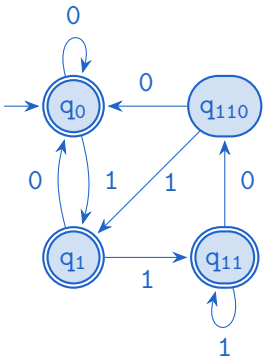
5. $\{w \in \{a,b\}^* : w$ bevat zowel $babb$ als aba niet als deelwoord$\}$

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



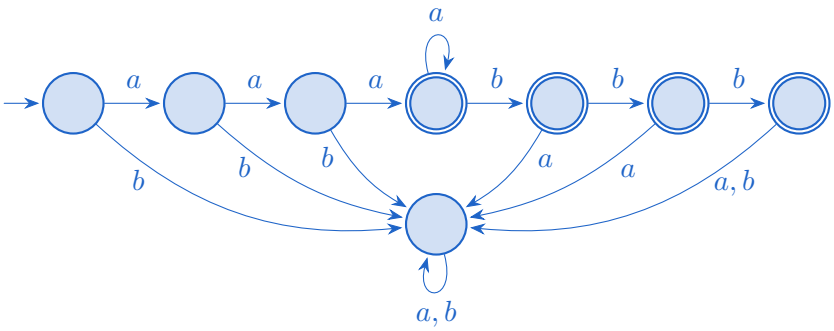
6. $\{w \in \{0,1\}^* : w$ eindigt niet op 110$\}$

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



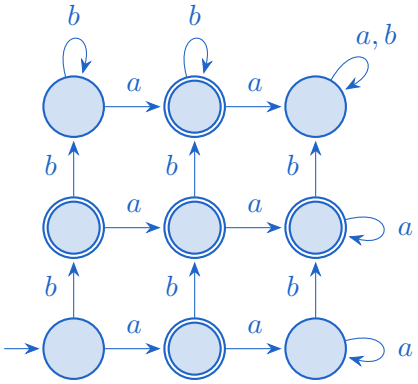
7. $\{a^m b^n : m \geqslant 3$ en $n \leqslant 3\}$

   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.



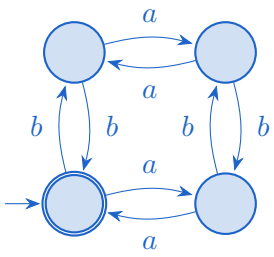
8. $\{w \in \{a,b\}^* : w$ bevat precies één a of precies één $b\}$

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



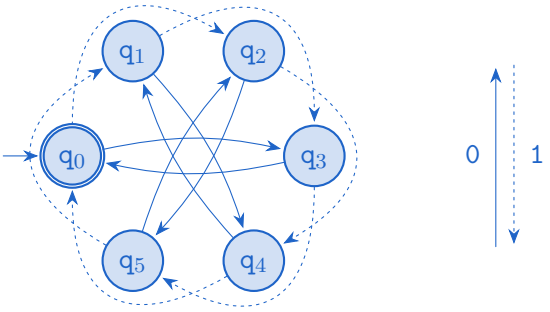
9. $\{w \in \{a,b\}^* :$ zowel $\#_a(w)$ als $\#_b(w)$ zijn even$\}$

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

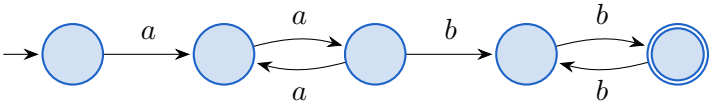
    We houden met zes toestanden de waarde van $3\,\#_0 + 2\,\#_1$ van de reeds ingelezen prefix bij, modulo zes. Lezen we een 0, dan moet de waarde met drie verhogen (modulo zes), bij een 1 met twee. Uiteindelijk aanvaarden we als de waarde nul blijkt (modulo zes).



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

1. $\{w \in \{0,\ldots,9\}^* : w$ is de decimale representatie van een oneven getal$\}$
2. $\{w \in \{a,b\}^* : w$ bevat minstens drie b's$\}$
3. $\{w \in \{a,b\}^* : w$ bevat zowel ab als bba niet als deelwoord$\}$
4. $\{w \in \{0,1\}^* : w$ bevat het deelwoord $101\}$
5. $\{w \in \{a,b\}^* : w$ bevat precies één $b\}$
6. $\{w \in \{a,b\}^* : w$ bevat minstens drie a's niet onmiddellijk gevolgd door een $b\}$
7. $\{w \in \{0,1\}^* : w$ bevat hoogstens één paar direct opeenvolgende nullen$\}$
8. $\{w \in \{a,b\}^* : w$ bevat minstens drie b's en hoogstens drie a's$\}$,
9. $\{w \in \{0,1\}^* : w$ bevat de deelwoorden 010 én $101\}$,
10. $\{w \in \{0,1\}^* :$ de laatste en voorlaatste nul in w staan opeenvolgend$\}$,
11. $\{a^m b a^n : m - n$ 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 $\{a^m b^n : m$ en n zijn even, met $m, n \geqslant 2\}$.

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

Het intuïtieve idee is eenvoudig: wissel de rol van aanvaardende en verwerpende toestanden om.

Concreet: beschouw een EDA $M = (K, \Sigma, \delta, s, A)$ waarvoor geldt dat $\mathcal{L}(M) = \mathcal{L}$. Dan aanvaardt de gemodificeerde EDA $M' = (K, \Sigma, \delta, s, K \setminus A)$ precies de taal $\overline{\mathcal{L}}$. De beide automaten hebben immers dezelfde transitiefunctie (zodat in het bijzonder de relaties $\vdash_M^*$ en $\vdash_{M'}^*$ gelijk zijn), en

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

zodat inderdaad $\mathcal{L}(M') = \overline{\mathcal{L}}$.

13. 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 $\mathcal{L}(M) = \mathcal{L}$.

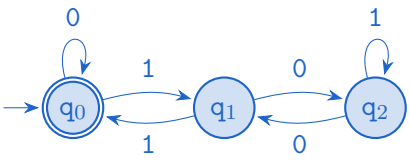
Constructie van de eindige deterministische herkenner
Idee: zij $w = x_1 \cdots x_n$ en noteer $[w]_2$ voor het getal voorgesteld door w. Gebruik drie toestanden, q_0, q_1 en q_2, om bij te houden wat de rest van de reeds verwerkte invoer bij deling door drie is; als we in toestand q_k eindigen, dan 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 $\mathsf{q}_{k'}$ 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:

$$\delta(\mathsf{q}_k, x) = \mathsf{q}_{k'} \text{ met } k' = (2k + x) \bmod 3.$$

Zij dus $M = (\{\mathsf{q}_0, \mathsf{q}_1, \mathsf{q}_2\}, \{0, 1\}, \delta, \mathsf{q}_0, \{\mathsf{q}_0\})$.



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 $(\mathsf{q}_0, w) \vdash^* (\mathsf{q}_k, \varepsilon)$.

We gebruiken inductie op $|w|$.

- Voor $|w| = 0$ is $w = \varepsilon$ en $[w]_2 = 0$, en inderdaad $(\mathsf{q}_0, \varepsilon) \vdash^* (\mathsf{q}_0, \varepsilon)$.
- 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 $(\mathsf{q}_0, v) \vdash^* (\mathsf{q}_k, \varepsilon)$.

  Enerzijds is $(\mathsf{q}_0, w) = (\mathsf{q}_0, vx) \vdash^* (\mathsf{q}_k, x) \vdash (\mathsf{q}_{k'}, \varepsilon)$, met $k' = (2k + x) \bmod 3$.

  Anderzijds is $[w]_2 \bmod 3 = (2[v]_2 + x) \bmod 3 = (2k + x) \bmod 3 = k' \bmod 3$.

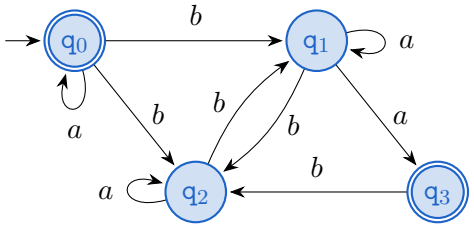
  Dus ook voor woorden w met lengte $n + 1$ geldt de eigenschap dat $[w]_2 \bmod 3 = k'$ als en slechts als $(\mathsf{q}_0, w) \vdash^* (\mathsf{q}_{k'}, \varepsilon)$, en per inductie dus voor alle woorden.

14. Veralgemeen oefening 13 naar de familie van talen

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

Niet-deterministische automaten

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



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

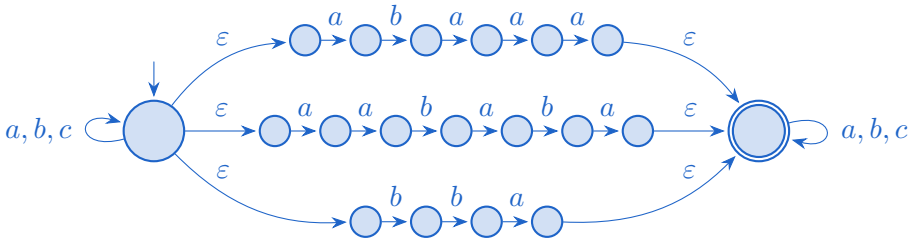
De woorden *abab* en *ababab* niet (er is immers geen enkele transitie met label b naar een aanvaardende toestand), *ababa* wel:

$$(\mathsf{q}_0, ababa) \vdash (\mathsf{q}_0, baba) \vdash (\mathsf{q}_2, aba) \vdash (\mathsf{q}_2, ba) \vdash (\mathsf{q}_1, a) \vdash (\mathsf{q}_3, \varepsilon).$$

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

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

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



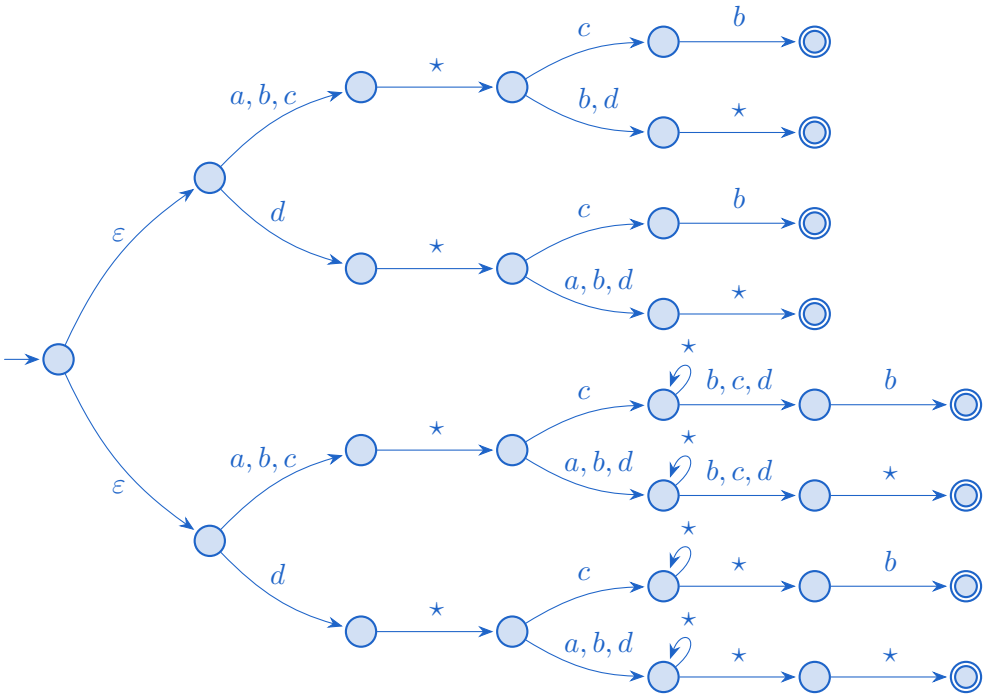
2. de taal van alle $w \in \{a,b,c,d\}^*$ waarvoor

   - $|w| \geqslant 4$,
   - 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 behandelen we apart.

   We noteren voor de overzichtelijkheid kortweg $\star$ voor een transitie met label a, b, c, d.

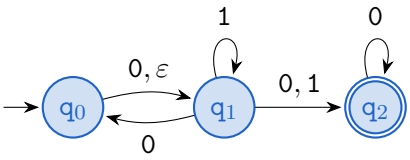


* 3. $\{w \in \{0,1\}^* :$ de tweede letter van w is een 0 en de voorlaatste letter een $1\}$

* 4. de taal van alle $w \in \{x,y\}^*$ die minstens één deelstring bevatten van drie identieke letters

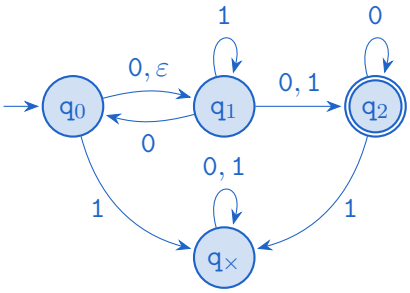
* 5. de taal van alle binaire voorstellingen in $\{0,1\}^*$ die een deelwoord bevatten dat een veelvoud van vijf voorstelt (verschillend van nul)

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



Stelling 2.2 geeft ons een algoritmische manier, met als achterliggende intuïtie het volgende idee: als nieuwe toestanden gebruiken we alle relevante *deelverzamelingen van toestanden*, die precies bijhouden op welke toestand we ons *kunnen* bevinden na het verwerken van de input. Zo kunnen we de machine deterministisch maken, maar het nadeel is dat het aantal toestanden exponentieel kan groeien (in het slechtste geval van n naar 2^n).

Vóór we meteen de constructie in stelling 2.2 toepassen, moeten we beseffen dat de tekening een subtiliteit onder de mat veegt: vanuit q_0 en q_2 vertrekken er namelijk geen transities met label 1. Hieronder wordt verstaan dat er een transitie met label 1 loopt naar een (niet expliciet getekende) *dead state* $q_\times$ waar de input verder wordt ingelezen en daarna verworpen. In feite ziet de automaat er dus als volgt uit.



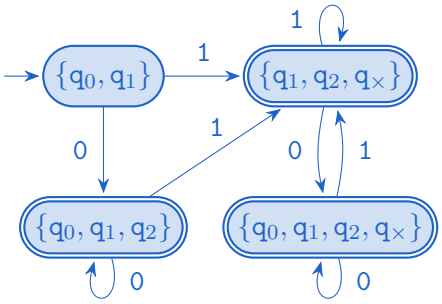
Merk op dat

$$\mathrm{eps}(q_0) = \{q_0, q_1\}, \qquad \mathrm{eps}(q_1) = \{q_1\}, \qquad \mathrm{eps}(q_2) = \{q_2\}, \qquad \mathrm{eps}(q_\times) = \{q_\times\}.$$

De nieuwe starttoestand s wordt de verzameling $\mathrm{eps}(q_0) = \{q_0, q_1\}$.

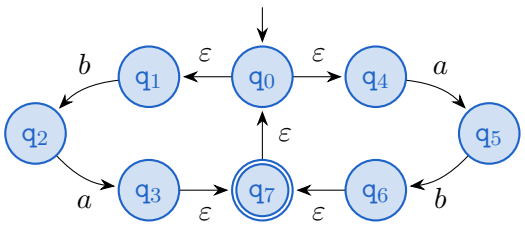
Hoe moeten we $\delta(s, 0)$ definiëren? Toestand $q_0 \in s$ kan onder 0 naar q_1, en $q_1 \in s$ naar q_0 en q_2. Met andere woorden, $\delta(s, 0) = \mathrm{eps}(q_0) \cup \mathrm{eps}(q_1) \cup \mathrm{eps}(q_2) = \{q_0, q_1, q_2\}$.

Stel op dezelfde manier de andere transities op en dan vind je uiteindelijk volgende automaat.



De aanvaardende toestanden zijn alle nieuwe toestanden die q_2 bevatten. Merk op dat deze EDA de volledige taal $\{0, 1\}^+$ aanvaardt.

* **18.** Idem voor deze NDA.



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

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

Constructie van de EDA M' zodat $\mathcal{L}(M') = \mathcal{L}^{\mathsf{R}}$
$\mathcal{L}^{\mathsf{R}}$ 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 $\delta(\mathtt{q}, a) = \mathtt{q}'$, tekenen we dus een pijl van $\mathtt{q}'$ naar $\mathtt{q}$ met label a; m.a.w. $((\mathtt{q}', a), \mathtt{q})$ wordt een element van de transitierelatie.

Merk op dat alle aanvaardende toestanden van M zo starttoestanden voor M' 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 $\mathtt{s}'$.

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

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

$$(\mathtt{s} = \mathtt{q}_0, x_1 x_2 \cdots x_n) \vdash_M (\mathtt{q}_1, x_2 \cdots x_n) \vdash_M \cdots \vdash_M (\mathtt{q}_{n-1}, x_n) \vdash_M (\mathtt{q}_n = \mathtt{q}, \varepsilon)$$

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

$$\begin{aligned}(\mathtt{s}', w) = (\mathtt{s}', x_n \cdots x_1) &\vdash_{M'} (\mathtt{q} = \mathtt{q}_n, x_n \cdots x_1) \\ &\vdash_{M'} (\mathtt{q}_{n-1}, x_{n-1} \cdots x_1) \\ &\;\;\vdots \\ &\vdash_{M'} (\mathtt{q}_1, x_1) \\ &\vdash_{M'} (\mathtt{s} = \mathtt{q}_0, \varepsilon),\end{aligned}$$

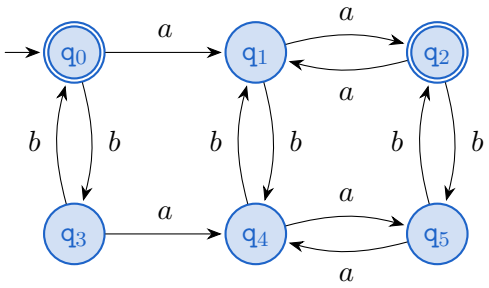
en omdat $\mathtt{s}$ aanvaardend is in M', is inderdaad $w \in \mathcal{L}(M')$.

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

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.

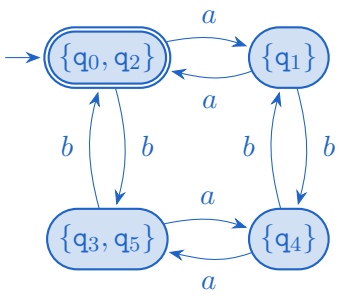
Beschouw het alfabet $\Sigma_k = \{1, \ldots, k\}$ en de taal $\mathcal{L}_k$ bestaande uit alle woorden $w \in \Sigma_k^*$ waarin minstens één letter uit Σ_k ontbreekt. Construeer voor deze taal een niet-deterministische automaat met $k + 2$ toestanden. Toon vervolgens aan dat elke deterministische 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.*

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 $q_i \equiv_{n+1} q_j$ 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.



Reguliere uitdrukkingen en grammatica's

22. Geef voor de volgende talen een reguliere uitdrukking.

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

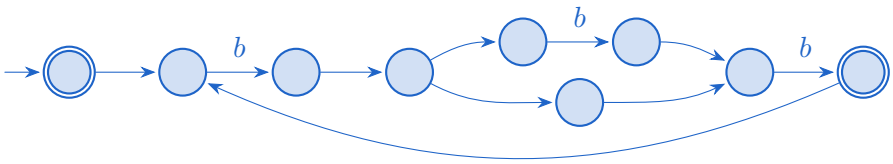
* 7. de taal uit oefening 13 wordt $(0 \cup 1(01^*0)^*1)^*$ via de constructie in stelling 4.2

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. $(b(b \cup \varepsilon)b)^*$

   We laten hier voor de overzichtelijkheid de labels bij ε-transities achterwege.

   De constructie in stelling 4.1 leidt tot de automaat

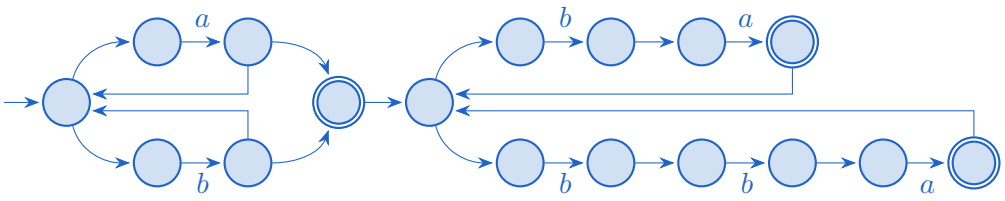


   Je ziet meteen dat deze automaat verre van minimaal is. Bovendien is de taal veel concreter te omschrijven als de taal $\{\varepsilon\} \cup \{b^n : n \in \mathbb{N}, n \geqslant 2\}$.

2. $(a \cup b)^+(ba \cup bba)^*$

   Stelling 4.1 geeft geen expliciete constructie om de operator $(\cdots)^+$ te vertalen naar een EDA. Overtuig jezelf dat het volstaat om alle aanvaardende toestanden van een automaat voor φ door middel van ε-transities te verbinden met de starttoestand, om een automaat voor φ^+ te bekomen. Wat is het verschil met de Kleenester, waar een subtiliteit optreedt?

   Als je dit niet vertrouwt, kun je ook de constructie voor de Kleenester gebruiken, waarbij je de nieuwe starttoestand dan gewoon niet aanvaardend maakt.

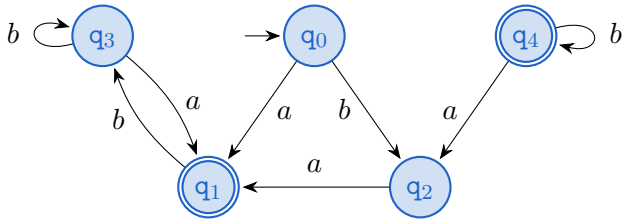


   Opnieuw is de automaat allesbehalve minimaal. De taal is eenvoudigweg gelijk aan $\{a, b\}^+$.

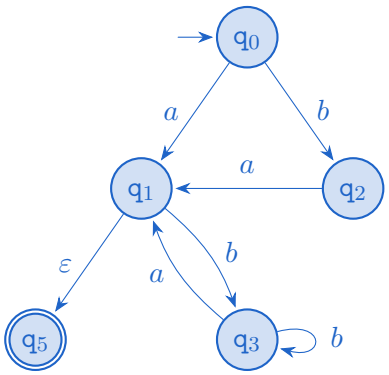
* 3. $(b \cup a)((b \cup \varepsilon)a)^+$

* 4. $(ab \cup ba)^* \cup (a^*)^*$

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



We volgen het `rip`-algoritme. Eerst voeren we de procedure `standardize` uit, met als resultaat



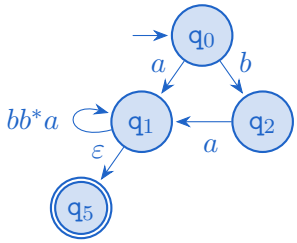
δ	q_0	q_1	q_2	q_3	q_5
q_0	$\emptyset$	a	b	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\emptyset$	b	ε
q_2	$\emptyset$	a	$\emptyset$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	a	$\emptyset$	b	$\emptyset$
q_5		$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Hierbij veronderstellen we tussen elk paar toppen waar momenteel geen expliciete transitie staat, een transitie met label $\emptyset$ (zoals in de tabel). Enkel van q_5 naar q_0 voegen we géén zo'n transitie toe. Laat ons nu als eerste de toestand q_3 onderaan `rip`-pen. Op een tekening ziet dit er vreselijk onoverzichtelijk uit, dus we geven de nieuwe transities ook in tabelvorm:

δ	q_0	q_1	q_2	q_5
q_0	$\emptyset \cup \emptyset b^* \emptyset$	$a \cup \emptyset b^* a$	$b \cup \emptyset b^* \emptyset$	$\emptyset \cup \emptyset b^* \emptyset$
q_1	$\emptyset \cup bb^* \emptyset$	$\emptyset \cup bb^* a$	$\emptyset \cup bb^* \emptyset$	$\varepsilon \cup bb^* \emptyset$
q_2	$\emptyset \cup \emptyset b^* \emptyset$	$a \cup \emptyset b^* a$	$\emptyset \cup \emptyset b^* \emptyset$	$\emptyset \cup \emptyset b^* \emptyset$
q_5		$\emptyset \cup \emptyset b^* a$	$\emptyset \cup \emptyset b^* \emptyset$	$\emptyset \cup \emptyset b^* \emptyset$

We kunnen dit uiteraard sterk vereenvoudigen, wetende dat bijvoorbeeld een concatenatie met $\emptyset$ gewoon opnieuw $\emptyset$ geeft.

δ	q_0	q_1	q_2	q_5
q_0	$\emptyset$	a	b	$\emptyset$
q_1	$\emptyset$	bb^*a	$\emptyset$	ε
q_2	$\emptyset$	a	$\emptyset$	$\emptyset$
q_5		$\emptyset$	$\emptyset$	$\emptyset$

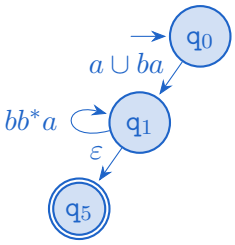


Herhalen we het proces met de toestand q_2 rechts, dan bekomen we gelijkaardig

δ	q_0	q_1	q_5
q_0	$\emptyset \cup b\emptyset^*\emptyset$	$a \cup b\emptyset^*a$	$\emptyset \cup b\emptyset^*\emptyset$
q_1	$\emptyset \cup \emptyset\emptyset^*\emptyset$	$bb^*a \cup \emptyset\emptyset^*a$	$\varepsilon \cup \emptyset\emptyset^*\emptyset$
q_5		$\emptyset \cup \emptyset\emptyset^*a$	$\emptyset \cup \emptyset\emptyset^*\emptyset$

$\Rightarrow$

δ	q_0	q_1	q_5
q_0	$\emptyset$	$a \cup ba$	$\emptyset$
q_1	$\emptyset$	bb^*a	ε
q_5		$\emptyset$	$\emptyset$



Onthoud dat $\emptyset^* = \{\varepsilon\}$! We hoeven nu enkel nog de toestand q_1 weg te werken.

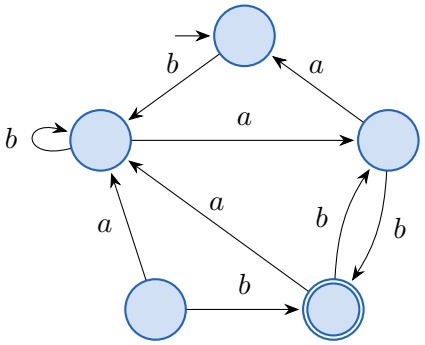
δ	q_0	q_5
q_0	$\emptyset \cup (a \cup ba)(bb^*a)^*\emptyset$	$\emptyset \cup (a \cup ba)(bb^*a)^*\varepsilon$
q_5		$\emptyset \cup \emptyset(bb^*a)^*\varepsilon$

$\Rightarrow$

δ	q_0	q_5
q_0	$\emptyset$	$(a \cup ba)(bb^*a)^*$
q_5		$\emptyset$

We besluiten dat $(a \cup ba)(bb^*a)^*$ een reguliere expressie is die dezelfde taal genereert als de taal aanvaard door onze oorspronkelijke EDA. Hier kon je dat weliswaar praktisch op het zicht zien, maar ook bij een ingewikkeldere EDA kun je dezelfde systematische werkwijze toepassen.

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



Afhankelijk van in welke volgorde je de toestanden wegwerkt, kun je een andere (maar uiteraard equivalente) reguliere expressie verkrijgen. Een mogelijke oplossing is

$$b(b \cup aab \cup ab(bb)^*(a \cup bab))^*ab(bb)^*.$$

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

We bedoelen met ρ^R de omgekeerde reguliere uitdrukking en bewijzen inductief dat $\mathcal{L}^R = \mathcal{L}(\rho^R)$.

1. $\rho = \emptyset$
   Dan is $\mathcal{L} = \emptyset$, $\mathcal{L}^R = \emptyset$ en $\rho^R = \emptyset$, zodat in dit geval inderdaad $\mathcal{L}^R = \mathcal{L}(\rho^R)$.

2. $\rho = \varepsilon$
   Dan is $\mathcal{L} = \{\varepsilon\}$, $\mathcal{L}^{\mathsf{R}} = \{\varepsilon\}$ en $\rho^{\mathsf{R}} = \varepsilon$, zodat in dit geval inderdaad $\mathcal{L}^{\mathsf{R}} = \mathcal{L}(\rho^{\mathsf{R}})$.
3. $\rho = c$
   Dan is $\mathcal{L} = \{c\}$, $\mathcal{L}^{\mathsf{R}} = \{c\}$ en $\rho^{\mathsf{R}} = c$, zodat in dit geval inderdaad $\mathcal{L}^{\mathsf{R}} = \mathcal{L}(\rho^{\mathsf{R}})$.
4. $\rho = \alpha \cup \beta$
   Dan is $\mathcal{L} = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$ en $\rho^{\mathsf{R}} = \beta \cup \alpha$. Omdat α en β een lagere complexiteit hebben dan ρ, volgt uit de inductiehypothese dat $\mathcal{L}(\alpha^{\mathsf{R}}) = \mathcal{L}(\alpha)^{\mathsf{R}}$ en $\mathcal{L}(\beta^{\mathsf{R}}) = \mathcal{L}(\beta)^{\mathsf{R}}$, zodat inderdaad $\mathcal{L}^{\mathsf{R}} = (\mathcal{L}(\alpha) \cup \mathcal{L}(\beta))^{\mathsf{R}} = \mathcal{L}(\alpha)^{\mathsf{R}} \cup \mathcal{L}(\beta)^{\mathsf{R}} = \mathcal{L}(\alpha^{\mathsf{R}}) \cup \mathcal{L}(\beta^{\mathsf{R}}) = \mathcal{L}(\alpha^{\mathsf{R}} \cup \beta^{\mathsf{R}}) = \mathcal{L}(\rho^{\mathsf{R}})$.
5. $\rho = \alpha\beta$
   Dan is $\mathcal{L} = \mathcal{L}(\alpha)\mathcal{L}(\beta)$ en $\rho^{\mathsf{R}} = \beta^{\mathsf{R}}\alpha^{\mathsf{R}}$. Omdat α en β een lagere complexiteit hebben dan ρ, volgt uit de inductiehypothese dat $\mathcal{L}(\alpha^{\mathsf{R}}) = \mathcal{L}(\alpha)^{\mathsf{R}}$ en $\mathcal{L}(\beta^{\mathsf{R}}) = \mathcal{L}(\beta)^{\mathsf{R}}$, zodat inderdaad $\mathcal{L}^{\mathsf{R}} = (\mathcal{L}(\alpha)\mathcal{L}(\beta))^{\mathsf{R}} = \mathcal{L}(\beta)^{\mathsf{R}}\mathcal{L}(\alpha)^{\mathsf{R}} = \mathcal{L}(\beta^{\mathsf{R}})\mathcal{L}(\alpha^{\mathsf{R}}) = \mathcal{L}(\beta^{\mathsf{R}}\alpha^{\mathsf{R}}) = \mathcal{L}(\rho^{\mathsf{R}})$.
6. $\rho = \alpha^*$
   Dan is $\mathcal{L} = \mathcal{L}(\alpha)^*$ en $\rho^{\mathsf{R}} = (\alpha^{\mathsf{R}})^*$. Omdat α een lagere complexiteit heeft dan ρ, volgt uit de inductiehypothese dat $\mathcal{L}(\alpha^{\mathsf{R}}) = \mathcal{L}(\alpha)^{\mathsf{R}}$, zodat inderdaad $\mathcal{L}^{\mathsf{R}} = (\mathcal{L}(\alpha)^*)^{\mathsf{R}} = (\mathcal{L}(\alpha)^{\mathsf{R}})^* = (\mathcal{L}(\alpha^{\mathsf{R}}))^* = \mathcal{L}((\alpha^{\mathsf{R}})^*) = \mathcal{L}(\rho^{\mathsf{R}})$.

Via inductie mogen we concluderen dat $\mathcal{L}^{\mathsf{R}} = \mathcal{L}(\rho^{\mathsf{R}})$. Aangezien ρ^{R} een reguliere uitdrukking is, volgt dat $\mathcal{L}^{\mathsf{R}}$ een reguliere taal is.

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.

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

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

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

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

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”.

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

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".

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

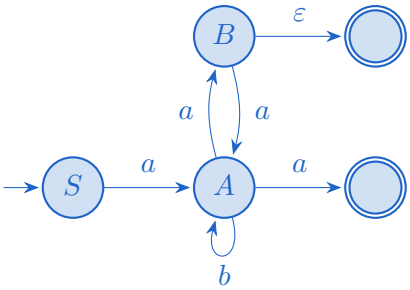
* **28.** Geef voor de volgende talen een reguliere grammatica.

1. $\emptyset$
2. $\{\varepsilon\}$
3. $\mathcal{L}((a \cup b)(ba \cup bba)^*)$
4. $\{a^n b^n : 0 \leqslant n \leqslant 3\}$
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.

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

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 $T \to x$ of $T \to \varepsilon$ is. Als dit gebeurt, willen we overgaan naar de starttoestand van de tweede grammatica.

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

Regels van de vorm $T \to \varepsilon$ zijn iets subtieler; we kunnen deze namelijk niet zomaar vervangen door $T \to S_2$, want dit is geen geldige herschrijfregel in een reguliere grammatica. We kunnen

dit als volgt oplossen. Verwijder de regel $T \to \varepsilon$, tenzij $S_2 \to \varepsilon$ een regel is voor de grammatica voor $\mathcal{L}_2$. Voeg daarna regels $T \to y$ toe voor elke regel van de vorm $S_2 \to y$, en regels $T \to zU$ voor elke regel van de vorm $S_2 \to zU$.

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

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

Definieer een nieuw startsymbool S'. We willen natuurlijk regels $S' \to S_1$ en $S' \to S_2$ toevoegen, maar ondervinden hetzelfde probleem als bij oefening 30: dit zijn geen toegestane herschrijfregels voor een reguliere grammatica. We voegen in de plaats voor iedere regel van de vorm $S_1 \to \varepsilon$, $S_1 \to x$ en $S_1 \to xT$, respectievelijk, de regel $S' \to \varepsilon$, $S' \to x$ en $S' \to xT$ toe. Analoog voor S_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 herschrijfregel $S' \to \varepsilon$. De regel $S' \to S$ 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 $T \to \varepsilon$ of $T \to x$, de regel $T \to S$ of $T \to xS$ toevoegen, alleen moeten we nogmaals hetzelfde trucje toepassen om $T \to S$ 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.

Noem de toestanden van de automaat $\mathtt{q}_0, \mathtt{q}_1, \ldots, \mathtt{q}_n$ met starttoestand $\mathtt{q}_0 = \mathtt{s}$. We gebruiken niet-terminalen $Q_0, \ldots, Q_n$ met startsymbool $Q_0 = S$. Voor elke transitie $\delta(\mathtt{q}_i, x) = \mathtt{q}_j$ voegen we een herschrijfregel $Q_i \to xQ_j$ toe, en voor elke aanvaardende toestand $\mathtt{q}_i$ een regel $Q_i \to \varepsilon$. Deze grammatica voldoet.

* **34.** Omschrijf een algoritme om, gegeven een reguliere grammatica voor $\mathcal{L}$, een reguliere grammatica te construeren voor $\mathcal{L}^{\mathsf{R}}$.

In de theorie zagen we een algoritme dat een reguliere grammatica in een EDA omzet. Bouw deze EDA daarna om naar een reguliere expressie ρ. We weten dat de reguliere expressie ρ^{R} de taal $\mathcal{L}^{\mathsf{R}}$ genereert (oefening 24), dus zet ρ^{R} om in een EDA, en tot slot terug in een reguliere grammatica.

* **35.** Bewijs dat een grammatica met herschrijfregels van de vorm $A \to wB$, met $w \in \Sigma^*$, nog steeds een reguliere taal genereert.

Beschouw zo'n regel $A \to wB$.

- Voor $|w| = 0$ (dus $w = \varepsilon$) staat er $A \to B$; de oplossing daartoe hebben we reeds besproken in oefening 30.
- Voor $|w| = 1$ is er geen probleem.
- Voor $|w| \geqslant 2$ staat er $A \to x_1x_2 \cdots x_nB$ ($n \geqslant 2$). Vervang deze door de regels $A \to x_1H_1$, $H_1 \to x_2H_2$, enzovoort tot $H_{n-1} \to x_nB$, waarin de H_i nieuwe niet-terminalen voorstellen.

Deze constructies veranderen duidelijk niets aan de gegenereerde taal, maar resulteren wel in een reguliere grammatica.

Pumping lemma

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 $\{a^{n+3}b^n : n \geqslant 3\}$, 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 $|w| \geqslant k$, er een factorisatie $w = xyz$ bestaat met $|xy| \leqslant k$ 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 $|w| = 2k + 3 \geqslant k$. Er geldt dus dat $a^{k+3}b^k = xyz$ met $|xy| \leqslant k$ 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 $|x| \geqslant 0$ 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. $\{a^m b^n : m \leqslant n\}$

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

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

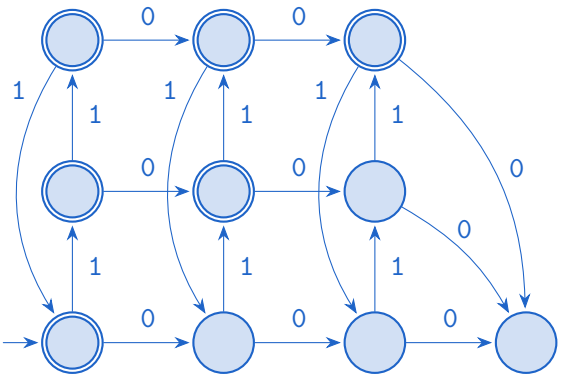
   Deze taal $\mathcal{L}$ is niet regulier. Deze opgave wordt het efficiëntst bewezen door gebruik te maken van de sluitingseigenschappen van reguliere talen. Veronderstel immers dat $\mathcal{L}$ regulier is, dan ook de doorsnede met de reguliere taal b^*c^*. Nu geldt $\mathcal{L} \cap b^*c^* = \{b^{6m}c^n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

4. $\mathcal{L}^*$ waarbij $\mathcal{L} = \{ww^{\mathsf{R}} : w \in \{a,b,c,d\}^*, |w| \leqslant 2019\}$

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

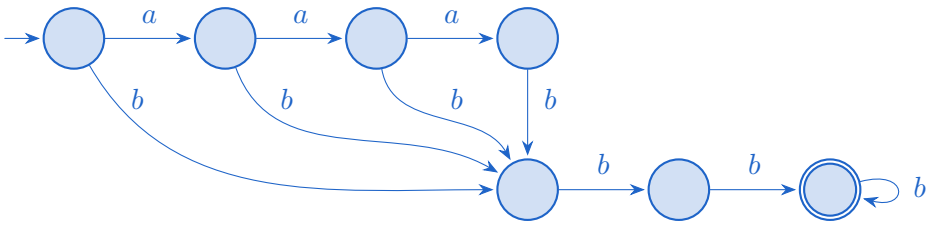
   Deze taal is regulier. We moeten immers niet het volledige aantal nullen en enen bijhouden: omdat $(\#_1(w) \bmod 3) \leqslant 2$, kunnen we woorden met drie of meer nullen zeker verwerpen.



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

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

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

   Deze taal is niet regulier. Veronderstel immers van wel, dan is ook het complement regulier, en de doorsnede van het complement met a^*b^*, wat precies de taal $\{a^n b^n : n \geqslant 0\}$ is.

2. $\{w \in \{a\}^*\{b,c,d\}^* : \#_a(w)$ is deelbaar door acht en $\#_b(w) \leqslant 2\,\#_c(w)\}$

   Deze taal is niet regulier. De doorsnede met b^*c^* is immers precies de taal $\{b^i c^j : i \leqslant 2j\}$, en het pumping lemma opwaarts toegepast op $b^{2k}c^k$ leidt tot een strijdigheid.

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

   Deze taal is regulier. Er zijn immers maar een eindig aantal mogelijkheden zodat $m+n=3$; de enige woorden bevat in de taal zijn a^3, a^2b, ab^2 en b^3.

4. $\{a^n b^n (bc)^m : n \geqslant 0, m \geqslant 0\}$

   Deze taal is niet regulier. De doorsnede met a^*b^* is immers precies de taal $\{a^n b^n : n \geqslant 0\}$.

5. $\mathcal{L}^{\mathsf{R}}$ met $\mathcal{L} = \{w \in \{a,b,c\}^* : \#_a(w) + \#_b(w) > \#_c(w)$ en w bevat geen deelwoord $aa\}$

   Deze taal is niet regulier. De doorsnede met b^*c^* is immers de taal $\{b^m c^n : m > n \geqslant 0\}$, en het pumping lemma neerwaarts toegepast op $b^{k+1}c^k$ leidt tot een strijdigheid.

6. $\{(ab)^n : n \geqslant 0\}$

   Deze taal is regulier. Werk zelf een automaat uit.

7. $\{(ab)^n (cd)^n : n \geqslant 0\}$

   Deze taal is niet regulier. Reguliere talen zijn immers gesloten onder lettersubstituties, maar de substitutie $(a \mapsto a)$, $(b \mapsto \varepsilon)$, $(c \mapsto \varepsilon)$, $(d \mapsto d)$ leidt tot de taal $\{a^n d^n : n \geqslant 0\}$.

8. $\{w \in \{a,b\}^* : w = w^{\mathsf{R}}\}$

   Deze taal is niet regulier. Pas het pumping lemma toe op het woord $a^k b a^k$.

9. $\{w\, w^{\mathsf{R}}\, w : w \in \{a,b\}^+\}$

   Deze taal is niet regulier. Pas het pumping lemma toe op het woord $(a^k b)(ba^k)(a^k b)$.

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

    Deze taal is regulier. De enige woorden die niet tot $\mathcal{L}\mathcal{L}$ behoren, blijken immers de woorden met oneven lengte die uit alternerende a's en b's bestaan; $\mathcal{L}\mathcal{L} = \overline{\mathcal{L}(a(ba)^* \cup b(ab)^*)}$.

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

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

   Gesloten.

   Zij $M_1 = (K_1, \Sigma, \delta_1, \mathsf{s}_1, A_1)$ en $M_2 = (K_2, \Sigma, \delta_2, \mathsf{s}_2, A_2)$ 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 $\mathcal{L}(M') = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$.

[(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 $f(\mathcal{L})$.

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

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

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

Per constructie (of via inductie op de lengte $|w|$) volgt dat $((\mathtt{s}_1, \mathtt{s}_2), w) \vdash^*_{M'} ((\mathtt{p}_1, \mathtt{p}_2), \varepsilon)$ als en slechts als $(\mathtt{s}_1, w) \vdash^*_{M_1} (\mathtt{p}_1, \varepsilon)$ en $(\mathtt{s}_2, w) \vdash^*_{M_2} (\mathtt{p}_2, \varepsilon)$. Daarna kunnen we besluiten dat $\mathcal{L}(M') = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$:

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

Alternatief: reguliere talen zijn gesloten onder unies en complementering, en aangezien

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \overline{\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}}$$

dus ook onder doorsnedes.

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

   Niet gesloten.

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

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

   Niet gesloten.

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

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

   Gesloten.

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

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

   $$A' = \{\mathtt{p} \in A : \text{voor elke } w \in \Sigma^+ \text{ met } (\mathtt{p}, w) \vdash^*_M (\mathtt{q}, \varepsilon) \text{ geldt dat } \mathtt{q} \notin A\}.$$

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

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

   Omwille van de assumptie $w \notin \mathtt{MaxString}(\mathcal{L})$ bestaat er een $y \in \Sigma^+$ met $wy \in \mathcal{L} = \mathcal{L}(M)$. Dit betekent dat er toestanden $\mathtt{q}_1$ en $\mathtt{q}_2$ bestaan zodat $(\mathtt{s}, wy) \vdash^*_M (\mathtt{q}_1, y) \vdash^*_M (\mathtt{q}_2, \varepsilon)$, met $\mathtt{q}_2 \in A$. Nu heeft M' dezelfde transitiefunctie als M, dus geldt ook dat

   $$(\mathtt{s}, wy) \vdash^*_{M'} (\mathtt{q}_1, y) \vdash^*_{M'} (\mathtt{q}_2, \varepsilon).$$

   We weten ook dat $(\mathtt{s}, w) \vdash^*_{M'} (\mathtt{p}, \varepsilon)$, zodat $\mathtt{q}_1 = \mathtt{p} \in A'$. Maar aangezien $(\mathtt{q}_1, y) \vdash^*_M (\mathtt{q}_2, \varepsilon)$ met $\mathtt{q}_2 \in A$, zou volgens de definitie van A' moeten gelden dat $\mathtt{q}_1 \notin A'$; een strijdigheid.

$\texttt{MaxString}(\mathcal{L}) \subseteq \mathcal{L}(M')$
Onderstel dat $w \in \texttt{MaxString}(L)$. Dan is $w \in \mathcal{L}$, dus $(\texttt{s}, w) \vdash_M^* (\texttt{q}_1, \varepsilon)$ met $\texttt{q}_1 \in A$. We weten dat $wy \notin \mathcal{L}$ voor alle $y \in \Sigma^+$. Hieruit volgt dat voor alle $y \in \Sigma^+$

$$(\texttt{s}, wy) \vdash_M^* (\texttt{q}_1, y) \vdash_M^* (\texttt{q}_2, \varepsilon),$$

met $\texttt{q}_2 \notin A$. Dit betekent per definitie van A' dat de toestand $\texttt{q}_1 \in A'$. In M' vinden we dus dat $(\texttt{s}, w) \vdash_{M'}^* (\texttt{q}_1, \varepsilon)$ met $\texttt{q}_1 \in A'$, zodat inderdaad $w \in \mathcal{L}(M')$.

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

Niet gesloten.

Een tegenvoorbeeld is $\texttt{Copy}(a^*b) = \{a^n b a^n b : n \geqslant 0\}$.

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

Niet gesloten.

Een tegenvoorbeeld is $\texttt{Fold}(a^*b) = \{a^n b a^{n-1} : n \geqslant 0\}$.

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

Gesloten.

Idee: beschouw binnen een EDA voor $\mathcal{L}$ alle paden naar een aanvaardende toestand, en maak iedere toestand op zo'n pad (inclusief de starttoestand) aanvaardend.

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

Gesloten.

Idee: beschouw een EDA voor $\mathcal{L}$. Een woord behoort slechts tot $\texttt{InvPrefixes}(\mathcal{L})$ als tijdens het verwerken van dit woord, alle gepasseerde toestanden aanvaardend waren. Het volstaat dus om alle niet-aanvaardende toestanden te vervangen door een enkele doodlopende *toestand, of om alle transities uit niet-aanvaardende toestanden te verwijderen.*

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

Gesloten.

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

Gesloten.

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

Gesloten.

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

Nee! Een tegenvoorbeeld wordt gegeven door $\mathcal{L} = \{a^m b^m c^n : m, n \geqslant 0\}$. Deze is duidelijk niet regulier (beschouw de doorsnede met a^*b^*), terwijl $\texttt{MaxString}(\mathcal{L}) = \emptyset$ wél (triviaal) regulier is. Ook de talen $\{a^p : p \text{ is priem}\}$ en $\{a^m b a^n : m \leqslant n\}$ 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 $\texttt{equalFSM}(M, N)$ die `True` teruggeeft indien $\mathcal{L}(M) = \mathcal{L}(N)$ en `False` anders.

1. Gegeven twee EDA's M en N, bepaal of $\mathcal{L}(M) = \mathcal{L}(N)^{\mathsf{R}}$.

   - Construeer een EDA N' zodanig dat $\mathcal{L}(N') = \mathcal{L}(N)^{\mathsf{R}}$; 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 `isInfinite`(M) om te bepalen of een EDA M al dan niet een oneindige taal genereert (zie de cursus).

   - Controleer met `isInfinite`(M) of $\mathcal{L}(M)$ oneindig is; indien ja, return `False`.
   - Controleer met `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 $|w| \leqslant k_M$ 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 $|w| \leqslant k_N$ 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:

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

   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 $(\mathtt{s}, w) \vdash^*_{M'} (\mathtt{p}, \varepsilon)$ met $\mathtt{p} \in A'$. Dan is $(\mathtt{s}, wa) \vdash^*_{M'} (\mathtt{p}, a)$. Omdat M' en M dezelfde transitiefunctie hebben, betekent dit dat ook $(\mathtt{s}, wa) \vdash^*_M (\mathtt{p}, a)$. Per definitie van A' geldt dat $(\mathtt{p}, a) \vdash_M (\mathtt{q}, \varepsilon)$ met $\mathtt{q} \in A$, zodat $(\mathtt{s}, wa) \vdash^*_M (\mathtt{q}, \varepsilon)$ met $\mathtt{q} \in A$. We besluiten dat inderdaad $wa \in \mathcal{L}(M)$.

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

   $$(\mathtt{s}, wa) \vdash^*_M (\mathtt{p}, a) \vdash_M (\mathtt{q}, \varepsilon),$$

   met $\mathtt{q} \in A$. Dit leert dat $\mathtt{p} \in A'$. Opnieuw omdat Men M' dezelfde transitiefunctie hebben, betekent dit dat $(\mathtt{s}, wa) \vdash^*_{M'} (\mathtt{p}, a)$, of dus dat $(\mathtt{s}, w) \vdash^*_{M'} (\mathtt{p}, \varepsilon)$, waarin $\mathtt{p} \in A'$. 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 $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\alpha)$.

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

41. [Examen 2011] Beschouw de operator

$$\texttt{Thrice}(\mathcal{L}) = \{x_1^3 x_2^3 \cdots x_n^3 : x_1 x_2 \cdots x_n \in \mathcal{L}\}.$$

Onderstel dat $\mathcal{L}$ regulier is; toon door constructie van een EDA aan dat ook $\texttt{Thrice}(\mathcal{L})$ regulier is.

Volgt de bewering meteen uit een stelling die in de theorie aan bod is gekomen? Welke?

Vervang in de EDA elke transitie door een pad van lengte drie, als volgt:

We laten geen andere transities toekomen in of vertrekken uit de paren nieuw toegevoegde toestanden. Deze nieuwe automaat aanvaardt $\texttt{Thrice}(\mathcal{L})$.

Alternatief kunnen we stelling 6.4 gebruiken, en meer bepaald de eigenschap dat reguliere talen gesloten zijn onder lettersubstituties. De substituties $x \mapsto xxx$ (voor elke $x \in \Sigma$) geven meteen het gezochte resultaat.

* **42. [Examen 2014]** Beschrijf een algoritme dat een EDA M omzet in een EDA M' met

$$\mathcal{L}(M') = \{w \in \mathcal{L}(M) : |w| \neq 6\}.$$

43. [Examen 2016] Beschouw talen over het alfabet $\Sigma = \{a, b, c, \ldots, z\}$, en beschouw de operator

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

Bijvoorbeeld, als het woord `appelflap` tot $\mathcal{L}$ behoort, dan zal $\texttt{Laang}(\mathcal{L})$ 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?

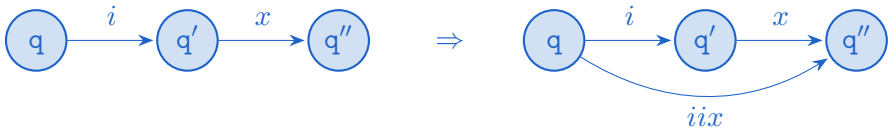
   Deze twee puntjes verlopen volledig analoog als in oefening 41 (maar dan enkel beschouwd voor de klinkers i.p.v. het gehele alfabet).

Deze `Laang` is vrij primitief: ze transformeert bijvoorbeeld `fiets` in `fiieets`. 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 `appelflap` 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`.

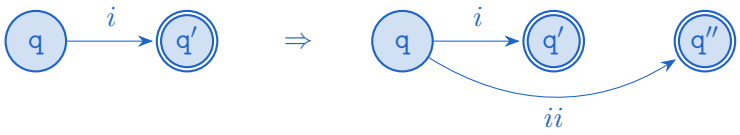
3. Is de verzameling van de reguliere talen gesloten onder `Laang2`?

   Ja. Beschouw een EDA M die $\mathcal{L}$ aanvaardt; we zullen deze aanpassen tot een automaat voor $\texttt{Laang2}(\mathcal{L})$. Voor elke twee opeenvolgende transities waarin twee klinkers worden gelezen, voegen we als volgt een nieuwe transitie toe.

   Voor elke twee opeenvolgende transities waarin eerst een klinker en daarna een medeklinker worden gelezen, voegen we als volgt een nieuwe transitie toe.



Voor elke transitie naar een aanvaardende toestand waarin een klinker wordt gelezen, voegen we als volgt een nieuwe transitie toe naar een nieuwe eindtoestand.



Ten slotte verwijderen we alle originele transities waarin een enkele klinker wordt gelezen. Deze nieuwe automaat aanvaardt precies $\texttt{Laang2}(\mathcal{L})$.

We kunnen natuurlijk zo verder gaan. Beschouw nu de operatie Laang∞ die alle maximale opeenvolgingen van één of meer klinkers vervangt door de originele klinkers, gevolgd door opnieuw die originele klinkers. Dus `zeeotter` wordt `zeeoeeotteer` en `koeoog` wordt `koeoooeoog`.

4. Is de verzameling van de reguliere talen gesloten onder Laang∞?

   Nee. Bekijk de (reguliere) taal $\mathcal{L} = \{ia^n : n \in \mathbb{N}\}$. Het beeld van deze taal onder Laang∞ is de taal $\{ia^nia^n : n \in \mathbb{N}\}$, die volgens het pumping lemma niet langer regulier is.

* **44. [Examen 2018]** Beschouw de operator Behead, die van alles in de taal de eerste letter afhaalt:

$$\texttt{Behead}(\mathcal{L}) = \{w \in \Sigma^* : (\exists\, x \in \Sigma)(xw \in \mathcal{L})\}.$$

Bewijs dat het beeld $\texttt{Behead}(\mathcal{L})$ van een reguliere taal $\mathcal{L}$ nog steeds regulier is.

Het volstaat *niet* om alle transities vanuit de starttoestand te vervangen door ε-transities. Waarom niet? Wat werkt er wel?

45. 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 $\sqrt{\mathcal{L}}$.

46. Om veilig te zijn, moeten wachtwoorden aan een aantal criteria voldoen, zoals voldoende lang zijn en minstens één cijfer en hoofdletter bevatten. Als je op een bepaalde site je wachtwoord vergeten bent, kun je meestal een nieuw wachtwoord laten genereren. Achter de schermen doet de website dit meestal door gebruik te maken van een (reguliere) grammatica.

Construeer een reguliere grammatica over het alfabet $\{\mathbf{a}, \mathbf{b}, \ldots, \mathbf{z}, \mathbf{A}, \mathbf{B}, \ldots, \mathbf{Z}, \mathbf{0}, \mathbf{1}, \ldots, \mathbf{9}\}$ die wachtwoorden w genereert met lengte minstens acht, die minstens één cijfer en minstens één hoofdletter bevatten.

Let op: het is uiteraard mogelijk om altijd als eerste symbool een hoofdletter en als tweede symbool een cijfer te genereren, maar dat is geen zo'n slim idee: een hacker die daarin de regelmaat herkent kan dit ook uitbuiten!

De niet-terminalen $S_\bullet$ betekenen dat we nog geen enkel cijfer of hoofdletter hebben gegenereerd, $T_\bullet$ dat we reeds een hoofdletter hebben gegenereerd, $U_\bullet$ dat we reeds een cijfer hebben gegenereerd, en $V_\bullet$ dat we reeds zowel een cijfer als een hoofdletter hebben gegenereerd.

Met een subscript houden we bij hoeveel symbolen we reeds gegenereerd hebben; zodra het er acht zijn, hoeven we dit aantal niet meer precies bij te houden. We mogen pas besluiten om te stoppen met genereren zodra we aan de niet-terminaal V_8 zitten.

$$
\begin{cases}
S \to \mathbf{a}S_1 \mid \mathbf{b}S_1 \mid \cdots \mid \mathbf{z}S_1 \mid \mathbf{A}T_1 \mid \mathbf{B}T_1 \mid \cdots \mid \mathbf{Z}T_1 \mid \mathbf{0}U_1 \mid \mathbf{1}U_1 \mid \cdots \mid \mathbf{9}U_1, \\
S_1 \to \mathbf{a}S_2 \mid \mathbf{b}S_2 \mid \cdots \mid \mathbf{z}S_2 \mid \mathbf{A}T_2 \mid \mathbf{B}T_2 \mid \cdots \mid \mathbf{Z}T_2 \mid \mathbf{0}U_2 \mid \mathbf{1}U_2 \mid \cdots \mid \mathbf{9}U_2, \\
\vdots \to \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \\
S_7 \to \mathbf{a}S_8 \mid \mathbf{b}S_8 \mid \cdots \mid \mathbf{z}S_8 \mid \mathbf{A}T_8 \mid \mathbf{B}T_8 \mid \cdots \mid \mathbf{Z}T_8 \mid \mathbf{0}U_8 \mid \mathbf{1}U_8 \mid \cdots \mid \mathbf{9}U_8, \\
S_8 \to \mathbf{a}S_8 \mid \mathbf{b}S_8 \mid \cdots \mid \mathbf{z}S_8 \mid \mathbf{A}T_8 \mid \mathbf{B}T_8 \mid \cdots \mid \mathbf{Z}T_8 \mid \mathbf{0}U_8 \mid \mathbf{1}U_8 \mid \cdots \mid \mathbf{9}U_8, \\
T_1 \to \mathbf{a}T_2 \mid \mathbf{b}T_2 \mid \cdots \mid \mathbf{z}T_2 \mid \mathbf{A}T_2 \mid \mathbf{B}T_2 \mid \cdots \mid \mathbf{Z}T_2 \mid \mathbf{0}V_2 \mid \mathbf{1}V_2 \mid \cdots \mid \mathbf{9}V_2, \\
T_2 \to \mathbf{a}T_3 \mid \mathbf{b}T_3 \mid \cdots \mid \mathbf{z}T_3 \mid \mathbf{A}T_3 \mid \mathbf{B}T_3 \mid \cdots \mid \mathbf{Z}T_3 \mid \mathbf{0}V_3 \mid \mathbf{1}V_3 \mid \cdots \mid \mathbf{9}V_3, \\
\vdots \to \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \\
T_8 \to \mathbf{a}T_8 \mid \mathbf{b}T_8 \mid \cdots \mid \mathbf{z}T_8 \mid \mathbf{A}T_8 \mid \mathbf{B}T_8 \mid \cdots \mid \mathbf{Z}T_8 \mid \mathbf{0}V_8 \mid \mathbf{1}V_8 \mid \cdots \mid \mathbf{9}V_8, \\
U_1 \to \mathbf{a}U_2 \mid \mathbf{b}U_2 \mid \cdots \mid \mathbf{z}U_2 \mid \mathbf{A}V_2 \mid \mathbf{B}V_2 \mid \cdots \mid \mathbf{Z}V_2 \mid \mathbf{0}U_2 \mid \mathbf{1}U_2 \mid \cdots \mid \mathbf{9}U_2, \\
U_2 \to \mathbf{a}U_3 \mid \mathbf{b}U_3 \mid \cdots \mid \mathbf{z}U_3 \mid \mathbf{A}V_3 \mid \mathbf{B}V_3 \mid \cdots \mid \mathbf{Z}V_3 \mid \mathbf{0}U_3 \mid \mathbf{1}U_3 \mid \cdots \mid \mathbf{9}U_3, \\
\vdots \to \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \\
U_8 \to \mathbf{a}U_8 \mid \mathbf{b}U_8 \mid \cdots \mid \mathbf{z}U_8 \mid \mathbf{A}V_8 \mid \mathbf{B}V_8 \mid \cdots \mid \mathbf{Z}V_8 \mid \mathbf{0}U_8 \mid \mathbf{1}U_8 \mid \cdots \mid \mathbf{9}U_8, \\
V_1 \to \mathbf{a}V_2 \mid \mathbf{b}V_2 \mid \cdots \mid \mathbf{z}V_2 \mid \mathbf{A}V_2 \mid \mathbf{B}V_2 \mid \cdots \mid \mathbf{Z}V_2 \mid \mathbf{0}V_2 \mid \mathbf{1}V_2 \mid \cdots \mid \mathbf{9}V_2, \\
V_2 \to \mathbf{a}V_3 \mid \mathbf{b}V_3 \mid \cdots \mid \mathbf{z}V_3 \mid \mathbf{A}V_3 \mid \mathbf{B}V_3 \mid \cdots \mid \mathbf{Z}V_3 \mid \mathbf{0}V_3 \mid \mathbf{1}V_3 \mid \cdots \mid \mathbf{9}V_3, \\
\vdots \to \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \mid \vdots \\
V_8 \to \mathbf{a}V_8 \mid \mathbf{b}V_8 \mid \cdots \mid \mathbf{z}V_8 \mid \mathbf{A}V_8 \mid \mathbf{B}V_8 \mid \cdots \mid \mathbf{Z}V_8 \mid \mathbf{0}V_8 \mid \mathbf{1}V_8 \mid \cdots \mid \mathbf{9}V_8 \mid \varepsilon.
\end{cases}
$$

Contextvrije grammatica's

47. Geef voor de volgende talen een contextvrije grammatica.

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

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

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

   We construeren alvast de herschrijfregels

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

   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.

   - $S \to XCD \mid ABY$.

   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.

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

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

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

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

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

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

   Merk op dat $a^m b^n$ met $2m = 3n + 1$ steeds van de vorm $a^{3k+2} b^{2k+1}$ is, voor zekere $k \in \mathbb{N}$.

   Het idee is om eerst $a^{3k+2} b^{2k+1}$ te genereren, waarvoor dus geldt dat $2m = 3n + 1$. Daarna forceren we ofwel $2m > 3n + 1$ door extra a's toe te voegen, maar let erop dat er eventueel nog één b bij kan (want anders eindigen we altijd met een oneven aantal b's); ofwel forceren we $2m < 3n + 1$ door extra b's, maar dan kunnen er eventueel nog één of twee a's bij.

   Let ook op dat we vanuit S de "kleine" woordjes zoals ab moeten kunnen genereren, die niet afkomstig zijn van de vorm $(a^{3k+2} b^{2k+1})$-met-extra-a's-of-b's.

$$\begin{cases} S \to \varepsilon \mid A \mid B \mid aB \mid aaTb, \\ T \to aaaTbb \mid U \mid V, \\ U \to aA \mid aaAb, \\ V \to Bb \mid aBb \mid aaBbb, \\ A \to aA \mid \varepsilon, \\ B \to bB \mid \varepsilon. \end{cases}$$

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

Voor deze taal volstaan $S \to SaSaSbS \mid SaSbSaS \mid SbSaSaS \mid \varepsilon$.

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

De niet-terminaal L neemt $a^i b^j$ met $i \leqslant j$ voor zijn rekening, R genereert $b^j c^k$ met $j \leqslant k$, en A en C genereren reeksen van respectievelijk a's en c's.

$$\begin{cases} S \to AR \mid LC, \\ A \to aA \mid \varepsilon, \\ C \to cC \mid \varepsilon, \\ R \to bRc \mid Rc \mid \varepsilon, \\ L \to aLb \mid Lb \mid \varepsilon. \end{cases}$$

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

Merk op dat $a^m b^n$ met $2m = 3n + 1$ steeds van de vorm $a^{3k+2}b^{2k+1}$ is, voor zekere $k \in \mathbb{N}$. Een eenvoudige grammatica $\mathcal{G}$ voor deze taal bestaat uit de twee herschrijfregels $S \to aaaSbb \mid aab$. We bewijzen nu dat $\mathcal{L}(\mathcal{G}) = \mathcal{L}$.

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 $x = S$, $x = a^m S b^n$ met $2m = 3n$, of $x = a^m b^n$ met $2m = 3n + 1$ is. De inductiebasis is triviaal: als we nog geen herschrijfregels hebben toegepast, is inderdaad $x = S$. Beschouw nu zo'n x van deze vorm, pas een herschrijfregel uit $\mathcal{G}$ toe, en noem het resultaat y. Er zijn vier mogelijkheden.

- *We pasten $S \to aaaSbb$ toe op $x = S$.* Dan is inderdaad $y = aaaSbb$ van de goede vorm.
- *We pasten $S \to aaaSbb$ toe op $x = a^m S b^n$ met $2m = 3n$.* Dan is $y = a^m \cdot aaaSbb \cdot b^n$ waarbij bovendien $2(m+3) = 3(n+2)$, dus y is inderdaad van de goede vorm.
- *We pasten $S \to aab$ toe op $x = S$.* Dan is inderdaad $y = aab$ van de goede vorm.
- *We pasten $S \to aab$ toe op $x = a^m S b^n$ met $2m = 3n$.* Dan is $y = a^m \cdot aab \cdot b^n$ waarbij bovendien $2(m+2) = 3(n+1)+1$, dus y is inderdaad van de goede vorm.

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-terminaal S meer mag staan, moet dat het geval $w = a^m b^n$ met $2m = 3n + 1$ zijn, 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 $2m = 3n + 1 \geqslant 7$. Er zijn dus minstens vier a's! We kunnen w schrijven als $aaa \cdot w' \cdot bb$ 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 $S \to aab$ zijn; dit is namelijk de enige waar rechts geen niet-terminalen meer staan. De laatste stap was dus $a^{m-5}Sb^{n-3} \Rightarrow_{\mathcal{G}} a^{m-5} \cdot aab \cdot b^{n-3} = w'$. Door eerst de herschrijfregel $S \to aaaSbb$ nog eens toe te passen, kunnen we dan ook

$$S \Rightarrow_{\mathcal{G}}^* a^{m-5}Sb^{n-3} \Rightarrow_{\mathcal{G}} a^{m-5} \cdot aaaSbb \cdot b^{n-3} = a^{m-2}Sb^{n-1} \Rightarrow_{\mathcal{G}} a^{m-2} \cdot aab \cdot b^{n-1} = a^m b^n$$

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

49. Een woord $w \in \{a, b\}^*$ heet *gebalanceerd* indien $\#_a(w) = \#_b(w)$ en indien $\#_a(x) \geqslant \#_b(x)$ 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 $S \to SS \mid aSb \mid \varepsilon$ 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 nu een afleiding met lengte $n + 1$: $S \Rightarrow_{\mathcal{G}}^* w' \Rightarrow_{\mathcal{G}} w$. Volgens onze inductiehypothese is w' gebalanceerd, met extra niet-terminalen S tussenin. We onderscheiden drie mogelijkheden voor de laatste toegepaste regel op w' om w te bekomen.

- *We pasten $S \to SS$ 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 $S \to aSb$ 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 $\#_a(x) \geqslant \#_b(x)$.
  - x is gelijk aan ua: de inductiehypothese stelt dat $\#_a(u) \geqslant \#_b(u)$, zodat
    $$\#_a(ua) = \#_a(u) + 1 \geqslant \#_b(u) + 1 = \#_b(ua) + 1.$$
  - x is gelijk aan uaS: de inductiehypothese stelt dat $\#_a(u) \geqslant \#_b(u)$, zodat
    $$\#_a(uaS) = \#_a(u) + 1 \geqslant \#_b(u) + 1 = \#_b(uaS) + 1.$$
  - x is gelijk aan $uaSb$: de inductiehypothese stelt dat $\#_a(u) \geqslant \#_b(u)$, zodat
    $$\#_a(uaSb) = \#_a(u) + 1 \geqslant \#_b(u) + 1 = \#_b(uaSb).$$
  - x is gelijk aan $uaSbv'$, met v' een bepaalde prefix van v: de inductiehypothese stelt dat $\#_a(uSv') \geqslant \#_b(uSv')$ (want uSv' is een prefix van w'), zodat
    $$\#_a(uaSbv') = \#_a(uSv') + 1 \geqslant \#_b(uSv') + 1 = \#_b(uaSbv').$$

  We concluderen dat voor elke mogelijke prefix x van w inderdaad geldt dat $\#_a(x) \geqslant \#_b(x)$.
- *We pasten $S \to \varepsilon$ 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.

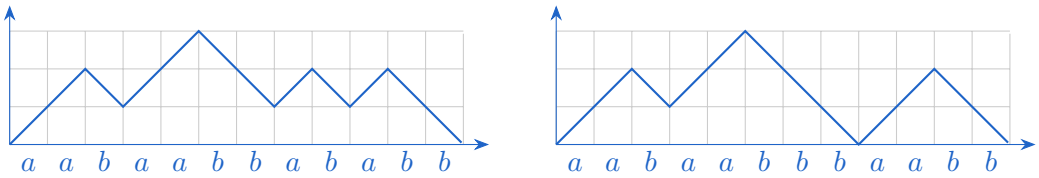
- *Er bestaat een* $0 < i < |w|$ *waarvoor* $g(i) = 0$. Dan is $w = xy$, met x en y twee gebalanceerde, *niet-ledige* woorden. Omdat de lengtes van x en y dan strikt kleiner zijn dan de lengte van w, leert de inductiehypothese dat $x, y \in \mathcal{L}(\mathcal{G})$. Er bestaan dus afleidingen $S \Rightarrow_{\mathcal{G}}^* x$ 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 herschrijfregel $S \to aSb$, 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 $\mathcal{L} \subseteq \mathcal{L}(\mathcal{G})$.

Grafische voorstelling
Deze bewijsstrategie kan heel goed op een grafische manier worden gedemonstreerd, die wellicht veelzeggender is. Stel een woord in $\{a, b\}^*$ voor op een rooster, door vanuit de oorsprong bij elke a een diagonale stap naar rechtsboven te zetten en bij elke b een diagonale stap naar rechtsonder. Zo staan hieronder de woorden $aabaabbbabab$ en $aabaabbbaabb$:



Merk op dat een woord in deze voorstelling gebalanceerd is precies wanneer die terug eindigt op de x-as en onderweg nooit strikt onder de x-as duikt. Op de y-as staat eigenlijk de waarde $g(w(x))$ uitgezet (in de notatie van hierboven). De figuur toont duidelijk aan dat er twee mogelijke situaties zijn, die elk een andere herschrijfregel nodig hebben.

In de situatie links blijft de grafiek overal strikt boven de x-as. Dat betekent dat het woord w van de vorm $aw'b$ is met w' opnieuw gebalanceerd. Deze w kunnen we dus vinden uit de herschrijfregel $S \to aSb$ waarbij de tweede S recursief w' gaat genereren.

In de situatie rechts is er tussendoor wel een punt op de x-as. De figuur verduidelijkt dat dit punt het woord verdeelt in twee stukken die zelf gebalanceerd zijn: w is van de vorm w_1w_2 waar w_1 en w_2 opnieuw gebalanceerd zijn. We kunnen w dus vinden vanuit de herschrijfregel $S \to SS$, waarbij de niet-terminalen S rechts recursief w_1 en w_2 gaan genereren.

50. Werk in de volgende grammatica's alle niet-productieve en onbereikbare niet-terminalen weg.

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

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

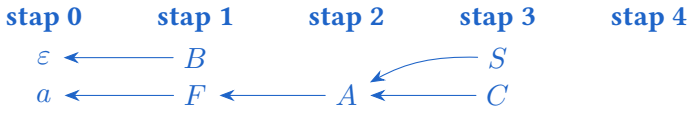
   We markeren eerst alle niet-terminale symbolen $\{S, A, B, C, D, E, F\}$ als niet-productief, en alle terminalen $\{a, b\}$ als productief.

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

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

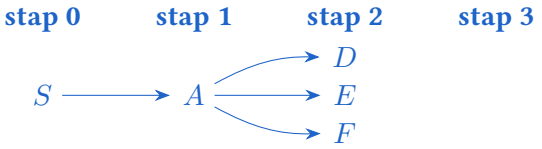
Nogmaals herhalen leidt tot de regels $S \to A$, $B \to bA$ en $C \to A$, dus ook het startsymbool is productief (gelukkig maar!), net als B.

Nogmaals herhalen leidt enkel tot de regel $S \to aS$, 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

$$\begin{cases} S \to aS \mid A, \\ A \to baF, \\ F \to bF \mid a. \end{cases}$$

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

Alle niet-terminalen zijn productief; enkel C is niet-bereikbaar.

51. Werk in de volgende grammatica's alle niet-productieve en onbereikbare niet-terminalen, ε-regels en 1-regels weg.

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

Alle niet-terminalen zijn productief én bereikbaar.

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

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

De eerste aanpasbare regel $S \to aaB$ wordt door `removeEps` herwerkt tot de regel $S \to aa$ (maar merk op dat we de originele regel niet schrappen). Dan wordt $S \to AaB$ omgezet in

$S \to aB$ (met A als *nullable*) en in $S \to Aa$ (met B als *nullable*). Deze twee regels zijn zelf opnieuw aanpasbaar, en worden nogmaals herwerkt tot $S \to a$ (beide dezelfde). $B \to bbA$ tot slot wordt $B \to bb$. Schrappen we nu nog $A \to \varepsilon$ en $B \to \varepsilon$, dan eindigen we met

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

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

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

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

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

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

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

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 ($S \to B$) over. We werken deze via de procedure `removeUnits` weg tot

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

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

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

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

$$\begin{cases} S \to a \mid aA \mid Aa, \\ A \to aB, \\ B \to a \mid Aa. \end{cases}$$

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

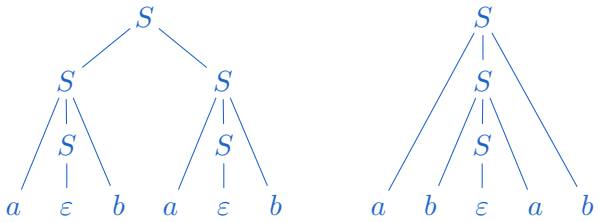
$$\begin{cases} S \to ABC \mid AB \mid AC \mid aC \mid aa \mid a, \\ A \to aC \mid aa \mid a, \\ B \to bB \mid aC \mid aa \mid a \mid b, \\ C \to Ac \mid Cc \mid c. \end{cases}$$

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

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

1. $S \to \varepsilon \mid aSa \mid bSb \mid aSb \mid bSa \mid SS$.

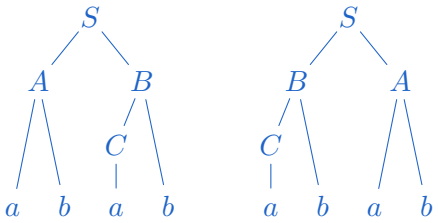
   De dubbelzinnigheid zit 'm hier in de regel $S \to SS$. Het woord $abab$ bijvoorbeeld heeft twee parse trees.



   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 $S \to SS$ weglaten. De grammatica is dan ondubbelzinnig, want tijdens het genereren van een woord w in de taal is er bij elke tussenstap slechts één herschrijfregel die de correcte eerste en laatste letters genereert.

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

   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

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

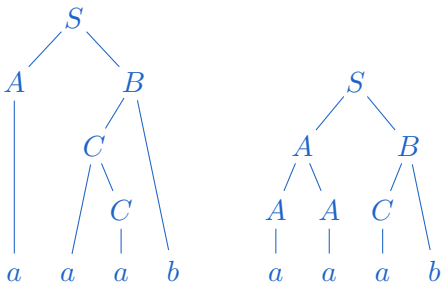
   We tonen nu aan dat $\mathcal{G}$ inderdaad ondubbelzinnig is. Kies een willekeurig woord $w \in \mathcal{L}(\mathcal{G})$; we tonen aan dat deze een unieke linkse afleiding heeft. Bekijk een willekeurige tussenstap in een afleiding van w. We zitten op dat moment met een woord in $\{a, b, S, A\}^*$. Beschouw de meest linkse niet-terminaal in deze tussenstap; de vraag is nu of er op deze niet-terminaal precies één herschrijfregel 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. $S \to AA$).
   - Stel dat A de meest linkse niet-terminaal is. De enige mogelijke regels die we dan kunnen toepassen, zijn $A \to aA$ en $A \to ab$. 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 genereren.

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

Merk op dat A uiteindelijk tekenreeksen van de vorm a^+ genereert, en B van de vorm a^+b. De startregel $S \to AB$ resulteert dus in woorden a^+a^+b. Maar bij een woord als $aaab$ is dan niet duidelijk of de a's door A of door B gegenereerd werden, cf. volgende parse trees.



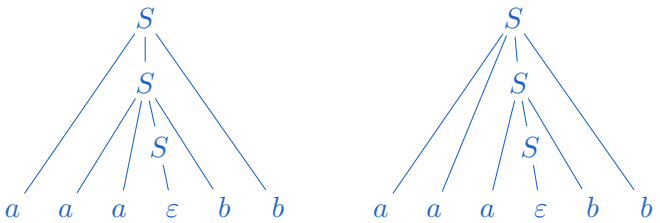
Een suggestie voor een oplossing is de grammatica

$$\begin{cases} S \to Cb, \\ C \to aC \mid aa. \end{cases}$$

Toon zelf aan dat deze inderdaad ondubbelzinnig is.

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

Het woord $aaab$ heeft twee parse trees.

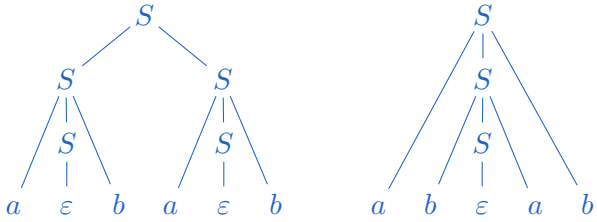


Een mogelijke oplossing is

$$\begin{cases} S \to aaSb \mid T, \\ T \to aTb \mid \varepsilon. \end{cases}$$

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

Het woord $abab$ heeft twee parse trees.



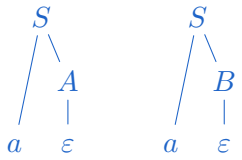
Zoek zelf een mogelijke oplossing.

54. Is een reguliere grammatica altijd ondubbelzinnig?

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

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

Dan is het woord a op twee manieren te verkrijgen.



55. 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 $S \to \varepsilon$ 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).*

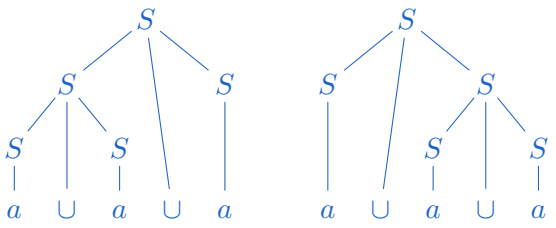
Voor deze oefening baseren we ons op de recursieve definitie van reguliere uitdrukkingen:

- 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.

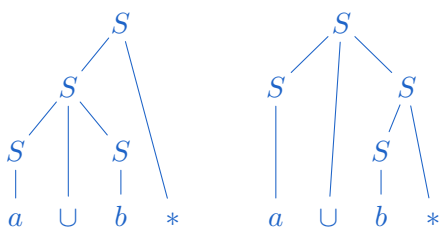
In eerste instantie construeren we dus de volgende grammatica, met één niet-terminaal S.

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

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.

$$\begin{cases} S \to S \cup X \mid X \\ X \to XY \mid Y \\ Y \to Y^* \mid Y^+ \\ Y \to (S) \\ Y \to a \mid b \mid \varepsilon. \end{cases}$$

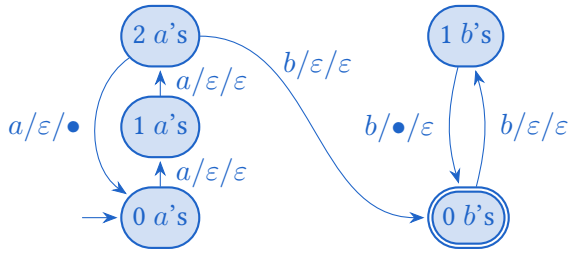
Deze grammatica is ondubbelzinnig.

Pushdownautomaten

56. Geef PDA's voor de volgende talen.

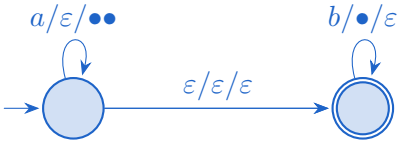
1. $\{a^m b^n : 2m = 3n + 1\}$

   Deze taal is precies gelijk aan de taal $\{a^{3k+2}b^{2k+1} : k \geqslant 0\}$. 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.



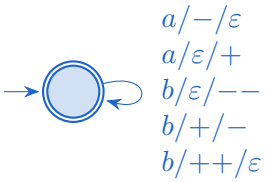
   Merk op: de automaat gaat slechts over naar de rechterhelft door een b te lezen wanneer er al $3k + 2$ letters a verwerkt zijn (k zijnde het aantal $\bullet$'s op dat moment op de stapel). Daarna moeten er nog precies $2k$ letters b worden gelezen om aanvaard te worden, dus halen we per twee b's een $\bullet$ 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. $\{a^n b^{2n} : n \geqslant 0\}$



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

   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.



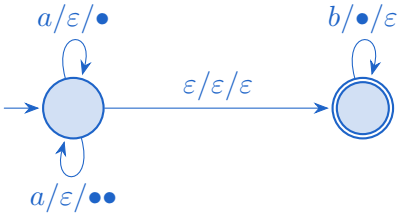
   Voor iedere a die we verwerken, dient de waarde op de stapel met één te verhogen. Dit lukt op twee manieren: ofwel zetten we $+$ op de stapel, ofwel halen we (indien mogelijk) $-$ van de stapel. Voor iedere b die we verwerken, moet de waarde op de stapel met twee verlagen. Dit kan op twee manieren: ofwel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

   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 aba, dat anders niet aanvaard zou kunnen worden.

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

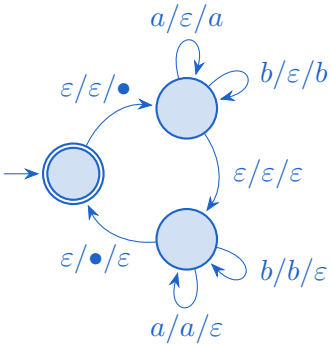
   Een woord in deze taal kunnen we steeds schrijven als $a^m b^m b^{n-m}$, met $0 \leqslant n - m \leqslant m$.



   Het ene uiterste (een woord $a^m b^m$) wordt aanvaard door enkel de bovenste lus te overlopen, die telkens één *marker* $\bullet$ op de stapel pusht. Het andere uiterste (een woord $a^m b^{2m}$) wordt aanvaard door enkel de onderste lus te overlopen, die telkens twee $\bullet$ pusht. Een algemeen woord $w = a^m b^m b^{n-m}$ wordt dan aanvaard door tijdens het verwerken van de m letters a, $n - m$ keer de onderste lus te overlopen; dan staan er $n - m$ markers méér op de stapel dan er a's gelezen zijn.

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

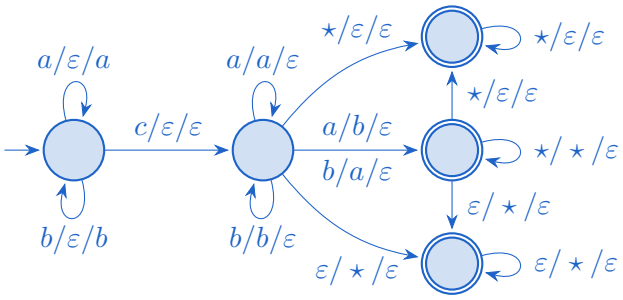
   Deze is een beetje *tricky*. We gebruiken een speciaal symbool $\bullet$ dat we onderaan op de stapel zetten, waaraan we kunnen herkennen of de stapel leeg is. Op die manier kunnen we onze stapel meerdere keren gebruiken in de zekerheid dat die telkens terug leeg is.



Het rechtergedeelte is de ondertussen vertrouwde automaat die $\mathcal{L}$ aanvaardt; de constructie met de lus en de *end-of-stack marker* $\bullet$ genereren de Kleene-ster van $\mathcal{L}$. Zonder $\bullet$ aanvaardt de automaat strikt meer woorden dan toegelaten, zoals het woord $a\,baab\,bb\,a \notin \mathcal{L}^*$.

6. $\{xcy : x, y \in \{a, b\}^* \text{ en } x \neq y^{\mathsf{R}}\}$

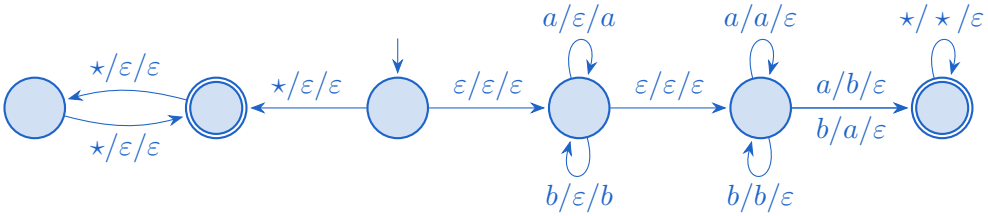
   De eerste tak houdt het geval $|x| < |y|$ bij, de tweede tak $|x| = |y|$ en de derde tak $|x| > |y|$.



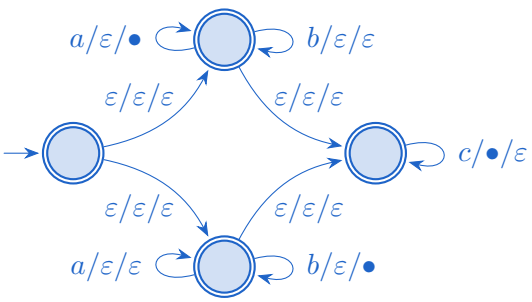
Om de tekening niet al te zwaar te maken, gebruiken we een symbool $\star$ voor a of b (niet ε!); de notatie $\star/\star/\varepsilon$ bijvoorbeeld is dan een afkorting voor de vier transities $(a/a/\varepsilon)$, $(a/b/\varepsilon)$, $(b/a/\varepsilon)$ en $(b/b/\varepsilon)$.

* 7. $\{w \in \{a, b\}^* : w \neq xx^{\mathsf{R}} \text{ voor elke } x \in \{a, b\}^*\}$

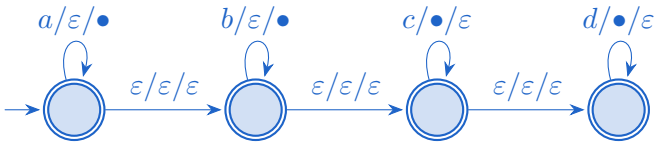
   We moeten alle woorden met oneven lengte en alle niet-palindroomwoorden met even lengte aanvaarden. De linkertak van volgende automaat aanvaardt het eerste geval, de rechtertak het tweede geval. We gebruiken opnieuw $\star$ als afkorting voor zowel a als b.



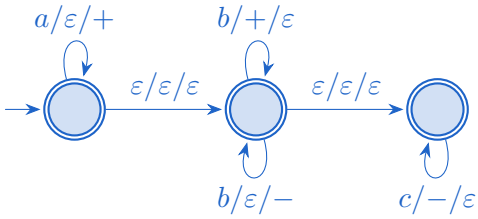
* 8. $\{x \in \{a, b\}^*\{c\}^* : \#_a(x) = \#_c(x) \text{ of } \#_b(x) = \#_c(x)\}$



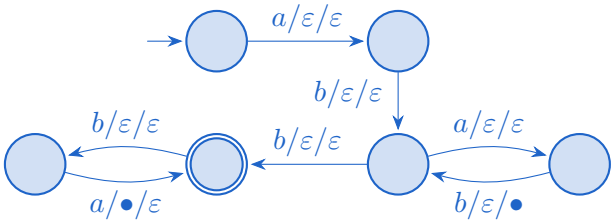
* 9. $\{a^m b^n c^p d^q : m + n = p + q\}$



* 10. $\{a^i b^j c^k : i + k = j\}$

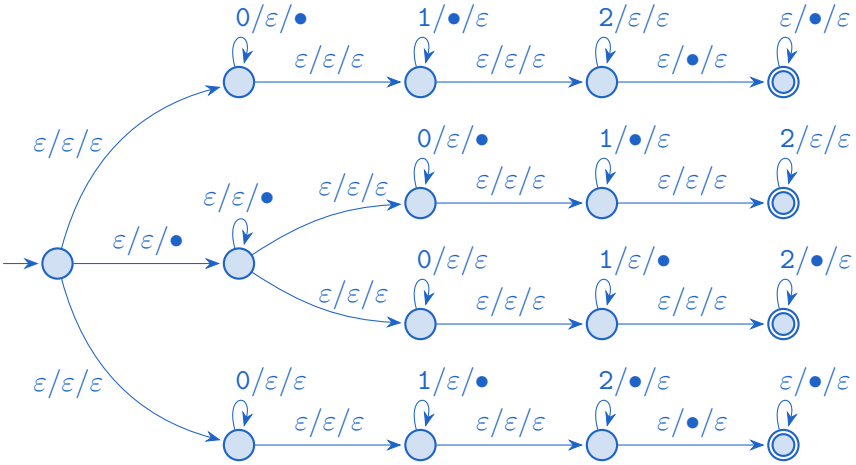


* 11. $\{ab(ab)^n b(ba)^n : n \geqslant 0\}$



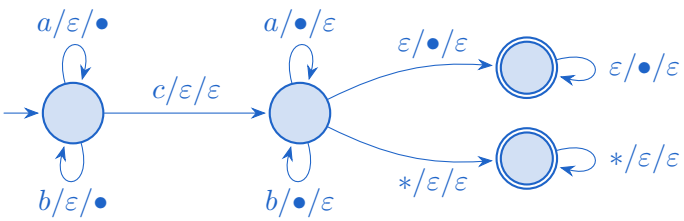
* 12. $\{0^i 1^j 2^k : i \neq j \text{ of } j \neq k\}$

De eerste tak forceert het geval $i < j$, de tweede $i > j$, de derde $j > k$ en de vierde $j < k$.

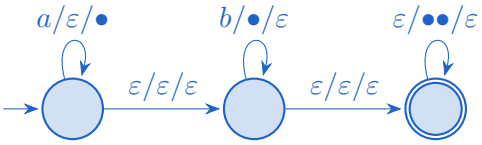


* 13. $\{xcy : x, y \in \{a, b\}^* \text{ en } |x| \neq |y|\}$

We gebruiken opnieuw $*$ als afkorting voor zowel a als b (niet c!).



* 14. $\{a^m b^n : m \geqslant n \text{ en } m - n \text{ is even}\}$



57. 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 het alfabet Σ uit de input $\mathcal{G}$.
   - Construeer een EDA N zodat $\mathcal{L}(N) = \{w \in \Sigma^* : |w| \text{ is even}\}$. Dit is eenvoudig: twee toestanden volstaan, samen met twee transities voor elke $x \in \Sigma$.
   - 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 `PDAtoCFG`.
   - Controleer of $\mathcal{L}(\mathcal{H}) = \emptyset$ via `decideCFLempty`. Zo ja, return `False`; zo nee, return `True`.

Pumping lemma voor contextvrije talen

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

1. $\{a^{n^4} : n \geqslant 0\}$ 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 $|vxy| \leqslant k$, $|vy| > 0$, en $w_p = uv^p xy^p z \in \mathcal{L}$ voor alle $p \in \mathbb{N}$. Zij nu concreet $p = 2$, dan is

   $$w_2 = u\,vv\,x\,yy\,z = a^{|u|}a^{2|v|}a^{|x|}a^{2|y|}a^{|z|} = a^{|w|+|vy|} = a^{k^4+|vy|}$$

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

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

   dus vinden we een strijdigheid.

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

2. $\{a^m b^n : m \geqslant (n-1)^3\}$ 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 $|vxy| \leqslant k$, $|vy| > 0$, en $w_p = uv^p xy^p z \in \mathcal{L}$ voor alle $p \in \mathbb{N}$. We overlopen alle 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 $k^3 \ngeqslant (k + |v| + |y|)^3$; 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 $k^3 - |v| - |y| \ngeqslant k^3$; 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
     $$k^3 - |v| \geqslant (k - |y|)^3 = k^3 - 3k^2|y| + 3k|y|^2 - |y|^3.$$
     Voor $p = 2$ staat er dat $w_2 = a^{k^3 + |v|}\, b^{k+|y|+1} \in \mathcal{L}$, zodat
     $$k^3 + |v| \geqslant (k + |y|)^3 = k^3 + 3k^2|y| + 3k|y|^2 + |y|^3.$$
     Tellen we nu deze beide vergelijkingen op en delen we door twee, dan bekomen we dat $k^3 \geqslant k^3 + 3k|y|^2$, wat duidelijk een strijdigheid is.

     Alternatief kunnen we inzien dat er "asymptotisch" wel een strijdigheid moet opduiken. Beschouw immers het algemene opgepompte woord w_p dat volgens het pumping lemma in de taal zou zitten, zodat
     $$\left(k^3 + (p-1)\cdot|v|\right) \geqslant \left(k + (p-1)\cdot|y|\right)^3.$$
     Aangezien k, $|v|$ en $|y|$ strikt positief zijn en niet wijzigen met p, is het linkerlid hier van de grootteorde $\mathcal{O}(p)$ en het rechterlid van de grootteorde $\mathcal{O}(p^3)$. Bij voldoende grote p kan de ongelijkheid dus nooit opgaan.

3. $\{a^p : p$ is een priemgetal$\}$ is niet contextvrij. Het bewijs (via het pumping lemma voor contextvrije talen) werkt geheel gelijkaardig als het bewijs dat deze taal niet regulier is.

4. $\{w \in \{a,b,c\}^* : \#_a(w) \leqslant \#_b(w)^2$ en $\#_c(w) > 2\}$ is niet contextvrij. Veronderstel immers van wel, dan is ook de doorsnede met de *reguliere* taal a^*b^*ccc contextvrij; noem deze doorsnede $\mathcal{L}'$. We tonen nu via het pumping lemma aan dat $\mathcal{L}'$ niet contextvrij is (en de originele taal dus ook niet).

   $\mathcal{L}'$ is oneindig groot, dus bestaat er een k met alle eigenschappen van het pumping lemma. Zij $w = a^{k^2} b^k ccc \in \mathcal{L}'$. Dan is $w = uvxyz$ zodat $w_p = uv^p xy^p z \in \mathcal{L}'$ voor alle $p \in \mathbb{N}$, met $|vxy| \leqslant k$ en $|vy| > 0$. We overlopen alle mogelijkheden.

- *Het woord v of het woord y bevat een c.*
  Dan bevat het opgepompte woord w_2 niet langer precies drie c's; een strijdigheid.
- *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.*
  Analoog als in oefening 58.2 (maar dan in de andere richting gepompt).
- *Zowel het woord v als het woord y bevatten geen b's.*
  Analoog als in oefening 58.2 (maar dan in de andere richting gepompt).
- *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. Net als in oefening 58.2 is dit geval het lastigst en moeten we wat durven proberen en rekenen. Zo kun je opmerken dat

  $$w_p = a^{k^2+(p-1)\cdot|v|}\, b^{k+(p-1)\cdot|y|}\, ccc,$$

  zodat $w_p \in \mathcal{L}'$ betekent dat

  $$k^2 + (p-1)\cdot|v| \leqslant \big(k + (p-1)\cdot|y|\big)^2 = k^2 + 2k(p-1)|y| + (p-1)^2|y|^2$$

  of na wat herschrijven

  $$(p-1)\cdot|v| \leqslant \big(2k + (p-1)|y|\big)\cdot(p-1)|y|.$$

  Merk op dat het rechterlid kwadratisch groeit met p, en het linkerlid slechts lineair, dus zomaar opwaarts pompen gaat de ongelijkheid alleen maar bevestigen. Deze observatie suggereert dat het woord w_0 interessanter kan zijn. Voor $p = 0$ vinden we dat

  $$|v| \geqslant \big(2k - |y|\big)\cdot|y|$$

  (denk eraan dat vermenigvuldigen met -1 de ongelijkheid omkeert!). We proberen om al onze kennis omtrent v en y uit te buiten om een strijdigheid te vinden. Daar $|v| \neq 0$, $|y| \neq 0$ en $|vxy| \leqslant k$, kunnen we besluiten dat

  $$1 \leqslant |v| \leqslant k-1 \quad \text{en} \quad 1 \leqslant |y| \leqslant k-1.$$

  Dat betekent dat

  $$|v| \geqslant \big(2k - |y|\big)\cdot|y| \geqslant \big(2k - (k-1)\big)\cdot|y| = (k+1)\cdot|y| \geqslant k+1,$$

  en dat is een strijdigheid.

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 $\{x^{n^3} : n \geqslant 0\}$ en de rest volgt uit een argument analoog als in opgave 2.

6. $\{xy : |x| = 2|y| \text{ en } x, y \in a^+b^+\}$ is contextvrij, maar niet regulier.

7. $\{a^nb^n : n \neq 5\}$ is contextvrij, maar niet regulier.

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

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

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

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

9. $\{a^ib^jc^k : i \leqslant k \leqslant j \text{ en } i \neq j\}$ is niet contextvrij.

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

    We weten immers dat $\mathcal{L}_1 = \{w \in \{0,1\}^* : \#_0(w) = \#_1(w)\}$ contextvrij is, en $\mathcal{L}_2 = \{w \in \{0,1\}^* : w$ 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 $\{0^n1^n : n \geqslant 0\}$, en de substitutie $0 \mapsto a$ en $1 \mapsto b$ geeft het vertrouwde tegenvoorbeeld $\{a^nb^n : n \geqslant 0\}$.

* 11. $\{(ab)^n a^n b^n : n > 0\}$ is niet contextvrij.

* 12. $\{xy : x, y \in \{a,b\}^*$ en $|x| = |y|\}$ is regulier.

    Dit is precies de taal van alle woorden met even lengte, en deze zijn eenvoudig te herkennen via een EDA (twee toestanden volstaan).

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

    Dit is de doorsnede van twee reguliere talen.

* 14. $\{a^m b^n : m, n > 0$ en $(m = n$ of $m = 2n)\}$ is contextvrij, maar niet regulier.

    Dit is de unie van twee contextvrije talen.

* 15. $\{a^m b^{2m} c^n : m, n \geqslant 0\} \cap \{a^m b^n c^{2n} : m, n \geqslant 0\}$ is niet contextvrij.

    Deze taal is ook kortweg te schrijven als $\{a^n b^{2n} c^{4n} : n \geqslant 0\}$, en is duidelijk niet-contextvrij. Merk op dat de twee talen afzonderlijk wél contextvrij zijn, maar hun doorsnede dus niet.

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

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

* 17. $\{ww^{\mathsf{R}}w : w \in \{a,b\}^*\}$ is niet contextvrij.

    Beschouw de doorsnede met de taal $(ba^*b)(ba^*b)(ba^*b)$ en pas dan het pumping lemma toe op het woord $w = (ba^kb)(ba^kb)(ba^kb)$.

* 18. $\{w \in \{a,b,c,d\}^* : \#_a(w) = \#_b(w) \cdot (\#_c(w) - 1)^2\}$ is niet contextvrij.

59. 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 $\mathtt{Q}(a,b,c)$ met drie parameters a, b en c.

$$\mathtt{Q}(a,b,c) = \begin{cases} \texttt{public class Q \{} \\ \quad \texttt{public static void main(String[] args) \{} \\ \quad\quad \texttt{int x}^a \texttt{ = 0;} \\ \quad\quad \texttt{x}^b \texttt{ = x}^c\texttt{;} \\ \quad \texttt{\}} \\ \texttt{\}} \end{cases}$$

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

1. Toon aan dat $\{\mathtt{Q}(a,b,c) : a, b, c \geqslant 1\}$ een reguliere taal is.

   Dit is straightforward.

2. Toon aan dat $\{\mathtt{Q}(a,b,c) : a, b, c \geqslant 1$, $\mathtt{Q}(a,b,c)$ compileert zonder fouten$\}$ niet contextvrij is.

   Beschouw de substitutie die alle karakters behalve 'x' en '=' op ε afbeeldt. Het beeld van de taal onder deze substitutie is de taal $\{\mathtt{x}^n = \mathtt{x}^n = \mathtt{x}^n : n \geqslant 1\}$, die niet contextvrij is wegens het pumping lemma voor contextvrije talen.

3. Besluit dat de taal van alle Javaprogramma's die foutloos compileren, niet contextvrij is.

   Mocht deze taal wel contextvrij zijn, dan ook de doorsnede met de *reguliere* taal uit het eerste deel van de vraag, maar deze doorsnede is precies de taal uit het tweede deel.

60. Veronderstel dat we de capaciteit van een PDA willen beperken door slechts een eindig geheugen te gebruiken. Concreet laten we op de stapel slechts $t+1$ beschrijfbare posities $0, \ldots, t$ 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.

We weten al dat elke reguliere taal aanvaard wordt door een PDA die zijn stapel niet gebruikt.

Omgekeerd, noteer Γ voor het stapelalfabet en noem $\tilde{\Gamma} = \{w \in \Gamma^* : |w| \leqslant t\}$. Deze verzameling bevat precies alle woorden die op de stapel geschreven kunnen worden. Merk op dat $\tilde{\Gamma}$ eindig is! Het idee is nu om een NDA te definiëren die de mogelijke woorden op de stapel bijhoudt in zijn (eindig aantal) toestanden, in plaats van op een echte stapel.

Als $(K, \Sigma, \Gamma, \Delta, s, A)$ de originele PDA was, dan definiëren we de toestanden van de nieuwe NDA concreet als $K' = K \times \tilde{\Gamma}$. De starttoestand wordt uiteraard $s' = (s, \varepsilon) \in K'$. De aanvaardende toestanden worden alle (a, ε) met $a \in A$. Rest nog de transitierelatie Δ' te beschrijven. Voor elke transitie $((q, w, \gamma), (q', \gamma')) \in \Delta$ en voor elke $x \in \tilde{\Gamma}$ waarvoor zowel $x\gamma \in \tilde{\Gamma}$ als $x\gamma' \in \tilde{\Gamma}$, voegen we de transitie van toestand $(q, x\gamma)$ naar toestand $(q', x\gamma')$ via het verwerken van w toe, of dus

$$\Delta' = \{((q, x\gamma), w, (q', x\gamma')) : ((q, w, \gamma), (q', \gamma')) \in \Delta, x \in \tilde{\Gamma}, x\gamma \in \tilde{\Gamma}, x\gamma' \in \tilde{\Gamma}\}.$$

Deze NDA $(K', \Sigma, \Delta', s', A')$ aanvaardt exact dezelfde taal als de PDA.

61. **[Examen 2014]** Toon aan dat de contextvrije talen niet gesloten zijn onder de operator

$$\texttt{Copy}(\mathcal{L}) = \{ww : w \in \mathcal{L}\}.$$

Bekijk de klassieke contextvrije taal $\mathcal{L} = \{a^n b^n : n \in \mathbb{N}\}$. Dan is $\texttt{Copy}(\mathcal{L}) = \{a^n b^n a^n b^n : n \in \mathbb{N}\}$ niet langer contextvrij; toon dit aan via het pumping lemma.

* **62.** **[Examen 2015]** Beschouw een alfabet Σ dat het symbool $\#$ bevat, en beschouw de operator

$$\texttt{Balanced}(\mathcal{L}) = \{w \in \mathcal{L} : (\exists x, y \in \Sigma^*)(w = x \# y \text{ en } |x| = |y|)\}.$$

Is de klasse van contextvrije talen gesloten onder `Balanced`? Verklaar!

63. **[Examen 2016]** Voor een formele taal $\mathcal{L}$ definiëren we $\texttt{Pow}(\mathcal{L}) = \{w^n : w \in \mathcal{L} \text{ en } n \geqslant 0\}$. Zijn de reguliere talen gesloten onder deze operatie `Pow`? Zelfde vraag voor de contextvrije talen.

$\texttt{Pow}(a^*b) \cap a^*ba^*b = \{a^n ba^n b : n \in \mathbb{N}\}$ is niet regulier (pumping lemma), dus de reguliere talen zijn niet gesloten onder `Pow`.

$\texttt{Pow}(a^*b) \cap a^*ba^*ba^*b = \{a^n ba^n ba^n b : n \in \mathbb{N}\}$ is niet contextvrij (pumping lemma), terwijl a^*b regulier en dus zeker contextvrij is, dus de contextvrije talen zijn niet gesloten onder `Pow`.

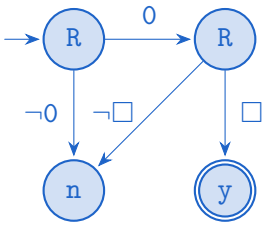
* **64.** **[Examen 2016]** Zij $p(n) = a_d\, n^d + \cdots + a_1\, n + a_0$ een veelterm van graad d, met coëfficiënten $a_i \in \mathbb{N}$ en $a_d \neq 0$. Toon aan dat de taal $\mathcal{L}_p = \{x^{p(n)} : n \in \mathbb{N}\}$ niet contextvrij is zodra $d \geqslant 2$. Wat als $d = 1$?

Turingmachines

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

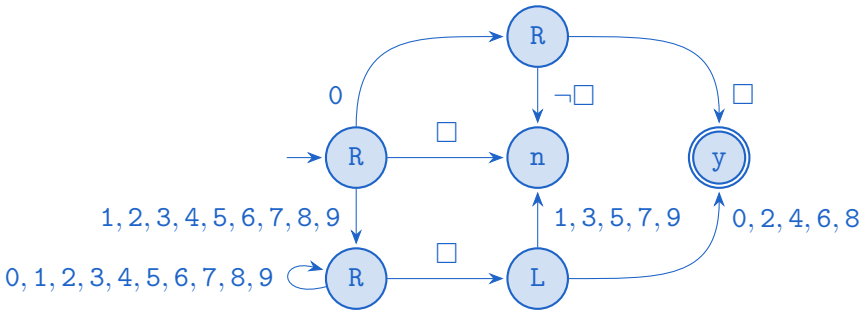
Let op: in macronotatie betekent $\mathtt{L}_\square$ (ga naar links totdat je een blanco vindt) *iets helemaal anders dan* $\mathtt{L}\ \square$ (ga naar links en schrijf een blanco)*, dus probeer je notatie op papier netjes te houden!*

1. $\{\mathtt{0}\}$



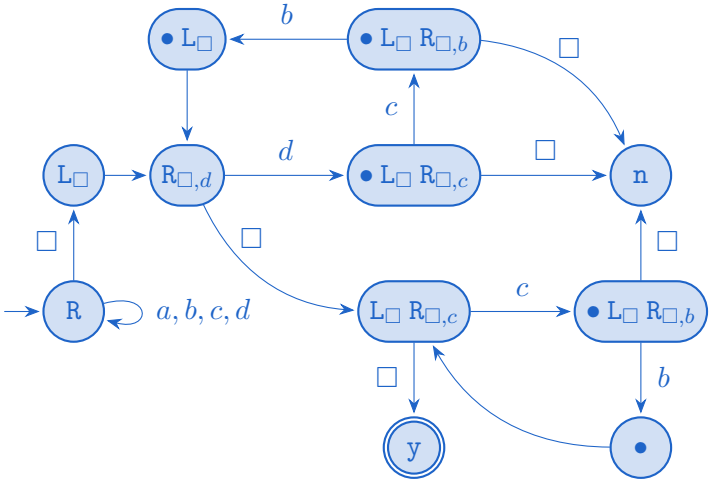
2. $\{w \in \{\mathtt{0},\mathtt{1},\ldots,\mathtt{9}\}^* : w \text{ is een even natuurlijk getal (zonder leidende nullen)}\}$

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



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

   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.



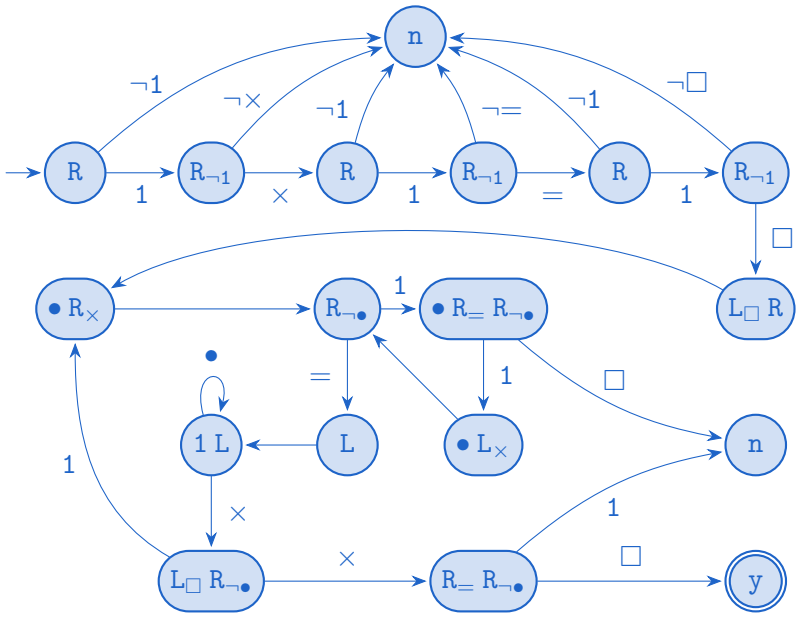
Opmerking: de eerste stappen dienen om te controleren of de invoer wel degelijk tot $\{a,b,c,d\}^*$ *behoort en geen exotische karakters bevat; zonder deze controle zou de machine bijvoorbeeld ook het woord* $abcde$ *aanvaarden!*

4. $\{x \times y = z : x, y, z \in \{1\}^+$ en $x \cdot y = z$ met x, y en z unair voorgestelde getallen$\}$

   Idee: we vervangen telkens één 1 van x door een marker $\bullet$ en we vervangen daarna evenveel tekens 1 van z door een marker $\bullet$ als er tekens 1 in y staan. Bijvoorbeeld, als we vertrekken van $111 \times 1111 = 1111\ 1111\ 1111$ (waarbij de extra spaties hier staan voor de duidelijkheid en niet op de band komen), dan willen we achtereenvolgens verkrijgen:

$$
\begin{aligned}
&111 \times 1111 = 1111\ 1111\ 1111,\\
&\bullet 11 \times 1111 = 1111\ 1111\ 1111,\\
&\bullet 11 \times \bullet 111 = \bullet 111\ 1111\ 1111,\\
&\bullet 11 \times \bullet\bullet 11 = \bullet\bullet 11\ 1111\ 1111,\\
&\bullet 11 \times \bullet\bullet\bullet 1 = \bullet\bullet\bullet 1\ 1111\ 1111,\\
&\bullet 11 \times \bullet\bullet\bullet\bullet = \bullet\bullet\bullet\bullet\ 1111\ 1111,\\
&\bullet 11 \times \bullet\bullet\bullet 1 = \bullet\bullet\bullet\bullet\ 1111\ 1111,\\
&\bullet 11 \times \bullet\bullet 11 = \bullet\bullet\bullet\bullet\ 1111\ 1111,\\
&\bullet 11 \times \bullet 111 = \bullet\bullet\bullet\bullet\ 1111\ 1111,\\
&\bullet 11 \times 1111 = \bullet\bullet\bullet\bullet\ 1111\ 1111,\\
&\bullet\bullet 1 \times 1111 = \bullet\bullet\bullet\bullet\ 1111\ 1111,\\
&\bullet\bullet 1 \times \bullet 111 = \bullet\bullet\bullet\bullet\ \bullet 111\ 1111,\\
&\bullet\bullet 1 \times \bullet\bullet 11 = \bullet\bullet\bullet\bullet\ \bullet\bullet 11\ 1111,
\end{aligned}
$$

   enzoverder. Komen we ooit symbolen 1 tekort in z, dan is $x \cdot y > z$. Houden we op het eind nog symbolen 1 over in z, dan is $x \cdot y < z$. In beide gevallen moeten we de input verwerpen.



   De bovenste rij van de Turingmachine gaat na of de input van de juiste vorm is. Daarna begint het echte werk! We keren terug naar het begin van de invoer en markeren een symbool in x. Daarna schuiven we op naar het y-gedeelte, en markeren we in zowel y als z een symbool 1 op de band tot y volledig gemarkeerd is (of de machine naar de verwerpende toestand gaat). Als dat gebeurd is, schrijven we de markeersymbolen in y opnieuw over met symbolen 1. We keren dan terug naar het x-gedeelte en herhalen dit proces, tot de x volledig gemarkeerd is. Op dat moment staan er $x \cdot y$ markeersymbolen in het z-gedeelte, dus hoeven we nog slechts na te gaan of er geen overtollige symbolen 1 staan rechts.

* 5. $\{w \in \{a, b, c\}^* : \#_c(w) \geqslant 1\}$

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

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

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

* **67.** 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.*

  Zij $(K, \Sigma, \Gamma, \delta, s, H)$ een deterministische Turingmachine met zo'n transitiefunctie

  $$\delta : (K \backslash H) \times \Gamma \to K \times \Gamma \times \{\leftarrow, \rightarrow, \perp\}.$$

  We willen het symbool $\perp$ elimineren. Het idee is eenvoudig; we kunnen "ter plekke blijven staan" door gewoon een stap opzij te zetten en meteen terug te keren, zonder iets op de band te wijzigen!

  Concreet voegen we voor elke transitie van de vorm $\delta(p, x) = (q, y, \perp)$ een nieuwe toestand p' in en vervangen we deze originele transitieregel door de twee nieuwe transities $\delta(p, x) = (p', y, \rightarrow)$ en $\delta(p', y) = (q, y, \leftarrow)$ voor elke $y \in \Gamma$.

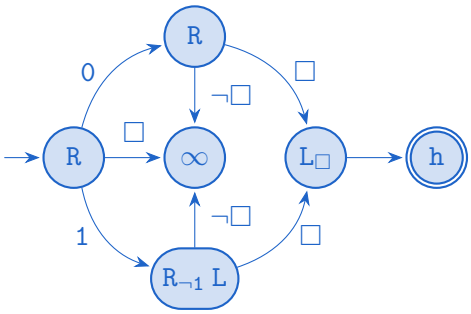
* **68.** In de definitie wat het betekent voor een Turingmachine M om een functie f te berekenen, hebben we geëist dat M maar één stoptoestand heeft. Toon nu aan dat M ook meerdere stoptoestanden mag hebben. Met andere woorden, bewijs dat volgende twee uitspraken equivalent zijn, gegeven een functie 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.

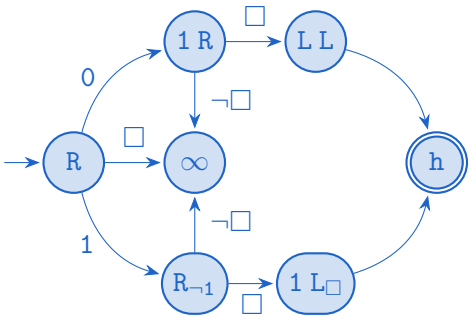
69. Gebruik de unaire representatie voor de natuurlijke getallen: het woord $\mathtt{0}$ stelt het getal nul voor, het woord $\mathtt{1}^n \in \{\mathtt{1}\}^+$ het getal n.

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.

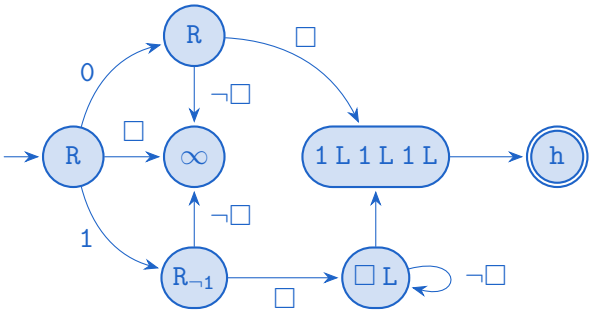


2. Geef een Turingmachine die de functie $f(n) = n + 1$ berekent.



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).

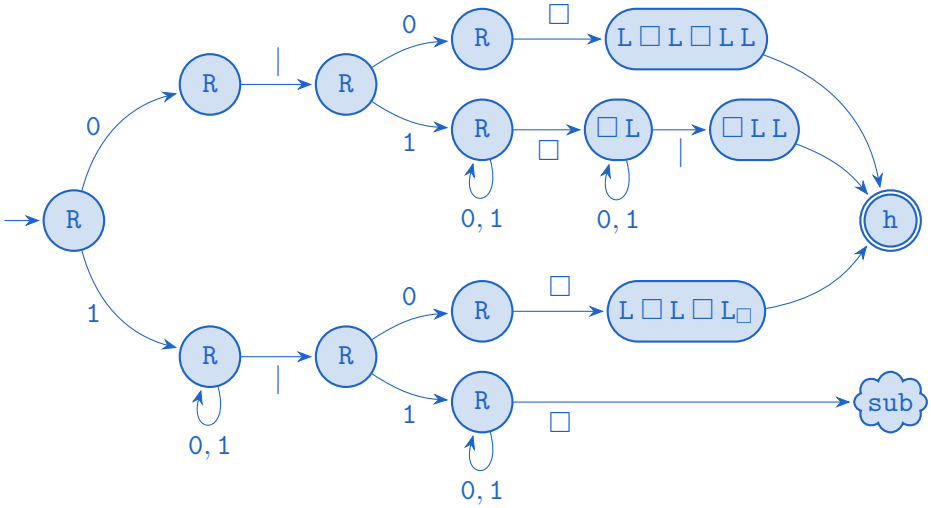


70. Geef een Turingmachine M (met één band) die de volgende functie `monus` berekent, waarbij we de natuurlijke getallen binair voorstellen en geen leidende nullen toelaten.

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

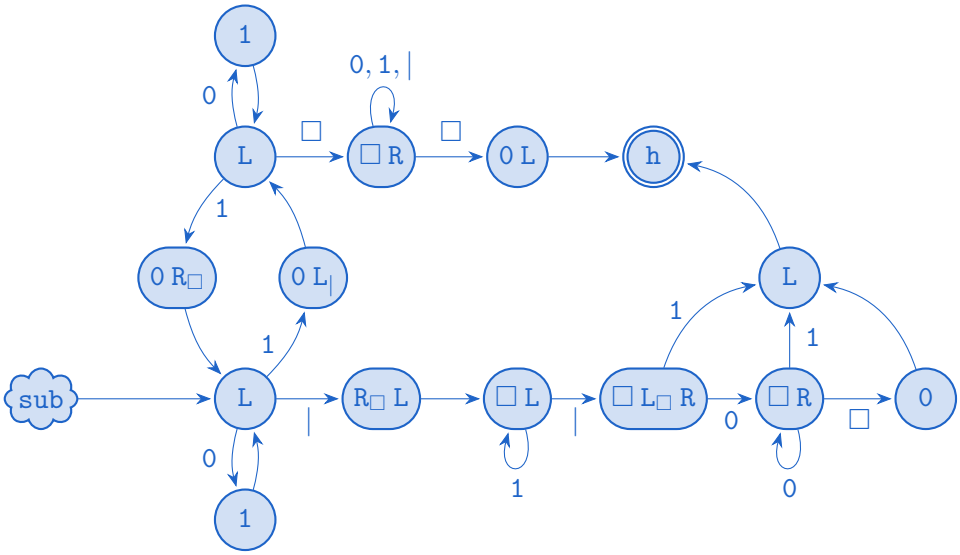
Bijvoorbeeld: als M vertrekt op bandinhoud ($\underline{\square}$ 101 | 11), dan moet M eindigen op ($\underline{\square}$ 10).

Eerst controleren we of de input van de juiste vorm is. Als $m = 0$ of $n = 0$, is de vereiste output niet moeilijk te genereren en geven we deze meteen terug, maar als $m > 0$ 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 $k \geqslant 0$) moeten we omzetten in $w01^k$ (mocht er geen 1 in staan, is het getal reeds nul). Uiteindelijk leidt dit tot de volgende lus.



∗ **71.** Geef een Turingmachine (met één band) die de volgende functie `subtract3` berekent, waarbij we de natuurlijke getallen binair voorstellen.

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

∗ **72.** Beschrijf een Turingmachine (met één band) die de som berekent van twee natuurlijke getallen, binair voorgesteld. Bijvoorbeeld: de invoer (␣ 101 | 11) moet verwerkt worden tot (␣ 1000).

73. 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 slog 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 (␣111|1), waar het linkergetal *drie* betekent dat we het rechtergetal nog drie keer moeten kopiëren (om dus *vier* symbolen 1 te bekomen). Als dit gelukt is, willen we (␣␣11|1111) bekomen, en moeten we dus 1111 nog *twee* keer kopiëren. Dan staat er

(␣ ␣ ␣ 1 | 1111 1111 1111)

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

(␣ ␣ ␣ ␣ | 1111 1111 1111 1111 1111 1111).

Nu staan er rechts $4 \cdot 3 \cdot 2 = 4!$ symbolen 1 en hoeven we enkel nog het symbool | te verwijderen.

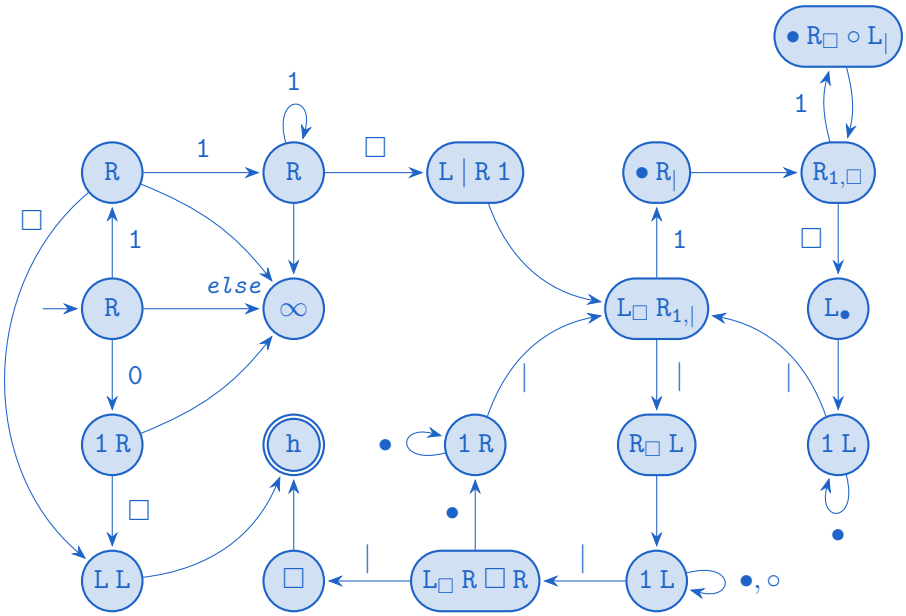
Hoe implementeren we dit kopiëren nu concreet? We zullen stap voor stap een 1 links markeren (met een •) en daarna rechts een kopie aanmaken (in symbolen ○ om het onderscheid te behouden met de originele symbolen 1 die we moeten kopiëren). Voor iedere kopie moeten we telkens een 1 rechts markeren (we gebruiken daarvoor eveneens •) en daarna een ○ bijschrijven, totdat elke 1 gemarkeerd is. Dan resetten we de rechtse •'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 •'s en ○'s terug op 1 stellen en één 1 links verwijderen. De eerste stappen worden

dus:

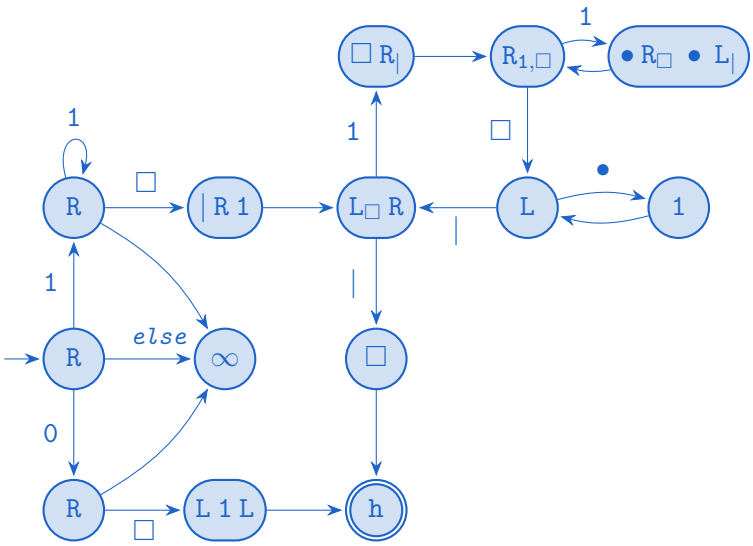
□ 1 1 1 | 1
□ • 1 1 | • ○
□ • 1 1 | 1 ○
□ • • 1 | • ○ ○
□ • • 1 | 1 ○ ○
□ • • • | • ○ ○ ○
□□ 1 1 | 1 1 1 1
□□ • 1 | • 1 1 1 ○
□□ • 1 | • • 1 1 ○ ○
□□ • 1 | • • • 1 ○ ○ ○
□□ • 1 | • • • • ○ ○ ○ ○
□□ • 1 | 1 1 1 1 ○ ○ ○ ○
□□ • • | • 1 1 1 ○ ○ ○ ○ ○
□□ • • | • • 1 1 ○ ○ ○ ○ ○ ○
□□ • • | • • • 1 ○ ○ ○ ○ ○ ○ ○
□□ • • | • • • • ○ ○ ○ ○ ○ ○ ○ ○
□□□ 1 | 1 1 1 1 1 1 1 1 1 1 1 1

□□□ • | 1 1 1 1 1 1 1 1 1 1 1 1 ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
□□□□ | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

De volgende Turingmachine klaart de klus.



* **74.** Beschrijf een Turingmachine (met één band) die op input $n \in \mathbb{N}$, unair voorgesteld, de n-de macht van twee berekent. Bijvoorbeeld: de invoer (□ 111) moet verwerkt worden tot (□ 11111111).



* **75.** Beschrijf een Turingmachine die natuurlijke getallen in binaire voorstelling omzet in unaire vorm. Bijvoorbeeld: de invoer (□ 101) moet verwerkt worden tot (□ 11111).

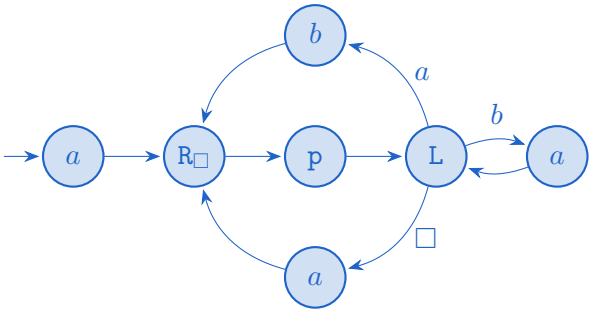
76. Geef een enumeratieprocedure voor de volgende talen.

In deze oefening stelt p steeds de specifieke "print"-toestand bij de enumeratieprocedure voor.

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

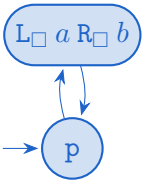
   We zullen de woorden lexicografisch opsommen, dus in de volgorde $a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, \ldots$ enzovoort. In essentie willen we in iedere stap het huidige woord wab^k (met $w \in \{a, b\}^*$ en $k \geqslant 0$) ombouwen naar het volgende woord wba^k terwijl 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. $\{a^n b^n : n \geqslant 0\}$

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

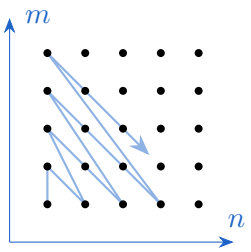


3. $\{a^n b^n c^m : n, m \geqslant 1\}$

   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. Er bestaat een

vrij eenvoudige methode om deze aan te pakken, door in iedere stap het volgende te doen, vertrekkend vanuit $(1,1)$:

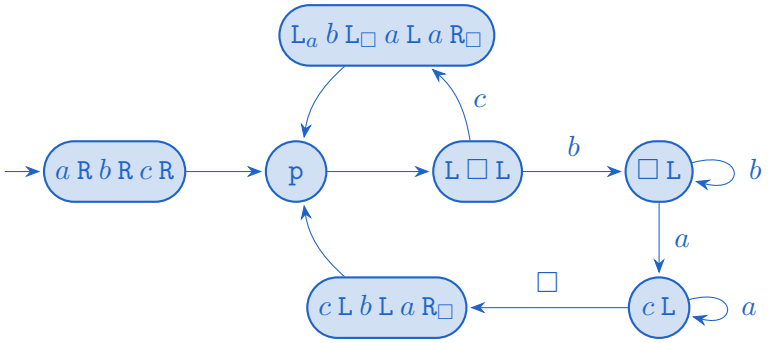
$$(n,m) \mapsto \begin{cases} (n+1, m-1) & \text{als } m > 1, \\ (1, n+1) & \text{als } m = 1. \end{cases}$$



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

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

De volgende Turingmachine klaart de klus.



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

* **77. [Examen 2014]** Geef een Turingmachine met één band (in macronotatie) die de taal

$$\{1^a \# 1^b \# 1^c \# 1^d : a + b = c + d\}$$

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

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

79. Geef een Turingmachine M die de functie $\texttt{max}_{a,b} : \{a,b\}^* \to \mathbb{N}$ berekent, met

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

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 (•), 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.

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

$$\begin{aligned}
\delta(s', (\square, \square, \square)) &= (h, (\mathtt{0}, \square, \square), (\leftarrow, \bot, \bot)) \\
\delta(s', (a, \square, \square)) &= (r, (\square, \bullet, \square), (\rightarrow, \rightarrow, \bot)) \\
\delta(s', (b, \square, \square)) &= (r, (\square, \square, \bullet), (\rightarrow, \bot, \rightarrow))
\end{aligned}$$

$$\begin{aligned}
\delta(r, (a, \square, \square)) &= (r, (\square, \bullet, \square), (\rightarrow, \rightarrow, \bot)) \\
\delta(r, (b, \square, \square)) &= (r, (\square, \square, \bullet), (\rightarrow, \bot, \rightarrow)) \\
\delta(r, (\square, \square, \square)) &= (t, (\square, \square, \square), (\leftarrow, \leftarrow, \leftarrow))
\end{aligned}$$

$$\begin{aligned}
\delta(t, (\square, \bullet, \bullet)) &= (t, (\mathtt{1}, \square, \square), (\leftarrow, \leftarrow, \leftarrow)) \\
\delta(t, (\square, \bullet, \square)) &= (t, (\mathtt{1}, \square, \square), (\leftarrow, \leftarrow, \bot)) \\
\delta(t, (\square, \square, \bullet)) &= (t, (\mathtt{1}, \square, \square), (\leftarrow, \bot, \leftarrow)) \\
\delta(t, (\square, \square, \square)) &= (h, (\square, \square, \square), (\bot, \bot, \bot))
\end{aligned}$$

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!

80. Toon aan dat de klasse van de beslisbare talen **niet** gesloten is onder lettersubstituties.

Hint: beschouw de taal

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

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 $(\mathtt{0} \mapsto \mathtt{0}), (\mathtt{1} \mapsto \mathtt{1}), (\bullet \mapsto \varepsilon)$ net het *halting problem* $\mathcal{H}$, en we weten dat deze niet beslisbaar is.

$\ast$ **81.** Onderstel dat de functie $f : \mathbb{N} \to \mathbb{N}$ Turingberekenbaar is.

- Bewijs dat de taal $\mathcal{L} = \{a^{f(n)} : n \in \mathbb{N}\}$ semibeslisbaar is.

  We omschrijven een Turingmachine die de taal semibeslist. Controleer eerst of de input van de vorm a^* is, en return `False` indien niet. Overloop de natuurlijke getallen $n = 0, 1, 2, \ldots$ en bereken telkens de functiewaarde $f(n)$ (op een aparte tape bijvoorbeeld, voor het gemak). Controleer na elke stap of deze $f(n)$ gelijk is aan het aantal a's in de input. Indien ja, return `True`; indien nee, genereren we het volgende natuurlijke getal.

  Krijgen we een invoer a^m te verwerken met m niet in het beeld van f, dan zal deze machine blijven nieuwe getallen genereren, tevergeefs op zoek naar een $n \in \mathbb{N}$ waarvoor $f(n) = m$. Daarom kunnen we slechts besluiten dat de Turingmachine de taal *semi*-beslist.

- Pas je bewijs aan voor berekenbare $f : \mathbb{N}^2 \to \mathbb{N}$, 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.

  Het voorgaande idee blijft toepasbaar, maar nu kunnen we bovendien een stopconditie implementeren! Wanneer we immers een $f(n)$ tegenkomen strikt groter dan het aantal a's in de input, dan weten we dat voor alle volgende natuurlijke getallen de functiewaarde alleen maar groter zal worden en dat we de input kunnen verwerpen. We blijven dus natuurlijke getallen genereren tot we in de situatie $f(n) \geqslant \#_a$ belanden (wat gegarandeerd zal gebeuren daar f strikt stijgend is). Als dan $f(n) = \#_a$, return `True`; anders, return `False`.

- Blijft $\mathcal{L}$ beslisbaar als f slechts *monotoon* stijgend is (dus $f(m) \leqslant f(n)$ als $m < n$)?

  Ja. Er zijn namelijk twee mogelijkheden: ofwel blijft f begrensd door een zekere constante c, ofwel neemt f onbeperkt toe. In het eerste geval is de taal $\mathcal{L}$ eindig, en dus zeker beslisbaar, ook al kennen we de precieze waarde van c niet! In het tweede geval kunnen we het idee uit de vorige paragraaf gebruiken.

- Toon aan dat $\mathcal{L}$ niet noodzakelijk beslisbaar hoeft te zijn, ook al is f berekenbaar.

  Hint: beschouw bijvoorbeeld de functie

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

  Merk op dat deze functie f wel degelijk Turingberekenbaar is, dankzij de extra parameter n: op input (m,n), controleer of m de codering van een Turingmachine is, simuleer vervolgens n stappen op de ledige input, en verwerp als de machine dan nog niet gestopt is. Het beeld van f is echter gelijk aan

$$\mathcal{H}_\varepsilon = \{\langle M \rangle : \text{Turingmachine } M \text{ stopt op input } \varepsilon\},$$

  en in de cursus werd aangetoond dat deze verzameling niet-beslisbaar is.

Reducties

De volgende algemene observaties kunnen nuttig zijn als richtlijn.

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

82. Beschouw de taal $\mathcal{L} = \{\langle M \rangle$: de Turingmachine M 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 $w_1, w_2, w_3, \ldots \in \Sigma^*$ 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 $\mathcal{L}' = \{\langle M \rangle : \text{de Turingmachine } M \text{ aanvaardt } \textit{precies} \text{ twee woorden}\}$. 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.

   Dit volgt rechtstreeks uit de stelling van Rice. Probeer zelf een bewijs via reductie te geven!

83. Toon via een reductie (niet via de stelling van Rice) aan dat de volgende talen in SD \ 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 uit de codering $\langle M \rangle$ een nieuwe Turingmachine M' op door alle transities in M naar de aanvaardende toestand te vervangen door transities naar de weigerende toestand. Voor de rest blijft M' ongewijzigd. Deze constructie zorgt ervoor dat M' enkel nog ofwel in een oneindige lus kan terechtkomen, ofwel stoppen en weigeren. Na deze modificatie geeft R als output de codering $\langle M', w \rangle$ terug (met hetzelfde woord w).

   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.

   - Een woord x *niet* van de vorm $\langle M, w \rangle$ zit zeker niet in $\mathcal{H}$, en $R(x)$ evenmin in $\mathcal{L}$.
   - Als $\langle M, w \rangle \in \mathcal{H}$, dus als M stopt op w, dan zal de geconstrueerde M' ook stoppen op w en weigeren, dus $R(M, w) = \langle M', w \rangle \in \mathcal{L}$.
   - Als $\langle M, w \rangle \notin \mathcal{H}$, dus als M niet stopt op w, dan blijft ook M' oneindig lang doorlopen op w, dus $R(M, w) = \langle M', w \rangle \notin \mathcal{L}$.

   Kunnen we nu besluiten dat $\mathcal{L}$ niet beslisbaar is? Veronderstel van wel, dus dat er een orakelmachine 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.

Een alternatieve manier is om het gedrag van M' niet te laten afhangen van diens input, maar enkel nog van het gedrag van M op w. We kunnen namelijk in de reductiemachine R ook de codering opstellen van 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 (of eender welk woord i.p.v. ε).

Ook deze reductiemachine R voldoet aan de gewenste eigenschappen.

- Als $\langle M, w \rangle \in \mathcal{H}$, dus als M stopt op w, dan zal M' altijd weigeren (ongeacht de input, die toch wordt gewist). 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, die toch wordt gewist) en zal in het bijzonder $\langle M', \varepsilon \rangle \notin \mathcal{L}$.

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

   Semibeslisbaarheid verloopt analoog als in oefening 82.

   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. $\{\langle M_1, M_2, w \rangle : M_1$ stopt op het lege woord en M_2 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 $\mathcal{H}_\varepsilon = \{\langle M \rangle : M$ 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 $\langle M \rangle \in \mathcal{H}_\varepsilon$. We weten dat deze laatste taal onbeslisbaar is.

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

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

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

1. $\{\langle M \rangle :$ elk woord dat M aanvaardt is een palindroom (een woord w 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.

Er is een alternatieve manier om te demonstreren dat $\mathcal{L}$ niet semibeslisbaar is: het volstaat om aan te tonen dat het *complement* $\overline{\mathcal{L}}$ wél semisbeslisbaar is. Immers, als zowel een taal als diens complement semibeslisbaar zijn, dan is deze taal beslisbaar, wat hier niet het geval is.

Kunnen we het complement van $\mathcal{L}$ semibeslissen? Merk op dat

$$\overline{\mathcal{L}} = \{\langle M\rangle : \text{er bestaat een woord } w \text{ dat } M \text{ aanvaardt en géén palindroom is}\} \cup \{\text{woorden niet van de vorm } \langle M\rangle\}$$

Een semibeslisser moet dus eerst controleren of diens input van de vorm $\langle M\rangle$ is. Indien niet, kan de machine meteen aanvaarden; indien wel, dan zoekt de machine naar een woord w dat wordt aanvaard door M en géén palindroom is. Daar moeten we *dovetailing* voor gebruiken. Werk zelf de details uit.

2. $\{\langle M\rangle : M$ 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. $\{w \in \{0,1\}^* : w$ 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 gespecifieerd in de input voor M.

85. 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 $\{\langle M\rangle : \mathcal{L}(M) \text{ is eindig}\}$ *beperkt tot EDA's* te beslissen. Bij de algemene taal *voor alle Turingmachines* hebben we echter geen zo'n informatie die we kunnen benutten.

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

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

* **88.** **[Examen 2014]** Beschouw de taal

$$\{\langle M, w\rangle : M \text{ aanvaardt } w, \text{ weigert } w^{\mathsf{R}} \text{ en stopt op ten minste drie woorden}\}.$$

Beschrijf een Turingmachine die deze taal semibeslist. Als we *"M stopt op ten minste drie woorden"* vervangen in *"M stopt op ten hoogste drie woorden"*, blijft de taal dan semibeslisbaar? Verklaar!

* **89.** **[Examen 2015]** Toon via een reductie (dus zonder toepassing van de stelling van Rice) aan dat

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

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

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

* **90.** **[Examen 2018]** Toon aan dat de volgende taal wel semibeslisbaar, maar niet beslisbaar is:

$$\left\{\langle M_1, M_2\rangle : \begin{array}{l} M_1 \text{ en } M_2 \text{ zijn twee Turingmachines waarvoor een input } w \text{ bestaat} \\ \text{die door } M_1 \text{ aanvaard en door } M_2 \text{ geweigerd wordt, of omgekeerd} \end{array}\right\}.$$

Maak gebruik van een expliciete reductie.

91. 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 à 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

   $$\delta(p, x) = (p', x', \mathtt{L})$$

   door drie nieuwe regels

   $$\begin{aligned}\delta(p,\ x) &= (q,\ x', \mathtt{L})\\ \delta(q,\ *) &= (q', *,\ \mathtt{L})\\ \delta(q', *) &= (p', *,\ \mathtt{R})\end{aligned}$$

   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.

92. 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 $\{\mathtt{0}, \mathtt{1}, \mathtt{x}, +, -, \times, =\}^*$ 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

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

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

$$\mathtt{111x1x1} - \mathtt{11x1x11} + \mathtt{1x11x11x111} = \mathtt{1111111}.$$

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.

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

1. Beschrijf intuïtief waarom $\mathcal{L}$ en $\mathcal{L}'$ semibeslisbaar zijn.

   Veeltermen bestaan uit niets meer dan sommen en producten, wat allebei Turingberekenbare operaties zijn. Men kan dus in principe op een Turingmachine de tupels in $\mathbb{Z}^k$ in een zekere volgende opsommen en telkens de waarde van de veelterm berekenen. Als we ooit zo'n tupel vinden waar de veelterm nul wordt, stoppen we; anders blijven we zoeken.

2. Toon aan dat $\mathcal{L}'$ beslisbaar is.

   We starten met een wiskundige analyse. Beschouw een veeltermvergelijking in één variabele en herschrijf die naar de vorm

   $$a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0 = 0.$$

   Onderstel dat $x \in \mathbb{Z}$ een oplossing is. Merk op dat elke term in het linkerlid een veelvoud is van x, behalve eventueel de constante term a_0. Als we dit herschrijven naar

   $$a_0 = -(a_k x^{k-1} + a_{k-1} x^{k-2} + \cdots + a_1) \cdot x,$$

   dan zien we dat ook a_0 een veelvoud moet zijn van x, gezien we werken met gehele getallen. Met andere woorden, de enige mogelijke oplossingen zijn delers van a_0! Gegeven de input-veelterm, hoeven we dus slechts eindig veel mogelijke waarden voor x te controleren. Als één daarvan een nulpunt blijkt, aanvaarden we; anders verwerpen we.

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.

Complexiteit

93. Beschouw de taal $\mathcal{L} = \{\xi\}$, waarin $\xi = 0$ als $\mathsf{P} = \mathsf{NP}$, of $\xi = 1$ als $\mathsf{P} \neq \mathsf{NP}$. 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!

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

Zij $\mathcal{L}_1, \ldots, \mathcal{L}_n \in \mathsf{P}$. Er bestaan dus n Turingmachines $M_1, \ldots, M_n$ die de respectievelijke talen beslissen in polynomiale tijd. We kunnen nu eenvoudig een Turingmachine opstellen die de unie $\mathcal{L}_1 \cup \cdots \cup \mathcal{L}_n$ (resp. doorsnede $\mathcal{L}_1 \cap \cdots \cap \mathcal{L}_n$) beslist, door deze n machines serieel uit te voeren en te aanvaarden als minstens één van de n machines (resp. elk van de n machines) aanvaardt.

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

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

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 $w = x_1 x_2 \cdots x_n$ 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 ∈ {1, ..., n}:
  for j ∈ {1, ..., i}:
    if j − 1 ∈ A and x_j ··· x_i ∈ L:
      A = A ∪ {i}
if n ∈ A:
  return True
else:
  return False
```

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

* **96. [Examen 2018]** Bewijs dat de complexiteitsklasse P gesloten is onder concatenatie.

Zij $\mathcal{L}_1$ en $\mathcal{L}_2$ twee talen in P en onderstel concreet dat $\mathcal{L}_1$ wordt beslist door een deterministische Turingmachine M_1 in tijd $p_1(|w|)$ en analoog $\mathcal{L}_2$ door M_2 in tijd $p_2(|w|)$. We beschrijven een deterministische Turingmachine die de concatenatie $\mathcal{L}_1 \cdot \mathcal{L}_2$ beslist in polynomiale tijd (opnieuw in pseudocode). Noem de input w; we noteren de prefix van w met lengte i kortweg als $w[:i]$ en de suffix van w met lengte $|w| - i$ als $w[i:]$ (zoals in Python).

```
for i ∈ {0, 1, ..., |w|}:
  if w[:i] ∈ L1 and w[i:] ∈ L2:
    return True
return False
```

In woorden, overloop voor de $n+1$ mogelijke opsplitsingen van w als $w_1 \cdot w_2$ of w_1 en w_2 tot $\mathcal{L}_1$ en $\mathcal{L}_2$ behoren. Dit algoritme verloopt essentieel in tijd

$$\sum_{i=0}^{n} \Big(p_1(i) + p_2(|w| - i)\Big) = \mathcal{O}\Big(|w| \cdot p_1(|w|) + |w| \cdot p_2(|w|)\Big),$$

dus nog steeds polynomiaal in $|w|$—weliswaar van één graad hoger dan p_1 en p_2.

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

Niet correct. Beschouw bij wijze van tegenvoorbeeld de talen $\mathcal{L}_1 = \emptyset$ en $\mathcal{L}_2 = \Sigma^*$. Deze behoren duidelijk tot P maar eender welke taal over Σ voldoet aan de inclusies $\emptyset \subseteq \mathcal{L} \subseteq \Sigma^*$.

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

Je kan uit elke taal in P een polynomiale reductie leggen naar elke andere taal in P, wat betekent dat elke taal in P dan ook P-compleet is, *behalve de ledige taal $\emptyset$ en de volledige taal Σ^**!

Zij immers $\mathcal{L}'$ en $\mathcal{L}$ twee talen in P. Een polynomiale reductie van $\mathcal{L}'$ naar $\mathcal{L}$ moet het probleem efficiënt "vertalen". Maar aangezien de taal $\mathcal{L}'$ in P zit, kun je deze taal eigenlijk al meteen efficiënt gaan beslissen. Je kunt dus als volgt een reductie opstellen.

(1) Op input x, controleer (in polynomiale tijd) of $x \in \mathcal{L}'$.

(2) Indien wel, geef een (arbitrair maar vastgekozen) woord $y_1 \in \mathcal{L}$ terug.

(3) Indien niet, geef een (arbitrair maar vastgekozen) woord $y_2 \notin \mathcal{L}$ terug.

Je kan zo een reductie opstellen tussen elke twee talen $\mathcal{L}'$ en $\mathcal{L}$ in P, *op voorwaarde dat er woorden $y_1 \in \mathcal{L}$ én $y_2 \notin \mathcal{L}$ bestaan!* Voor $\mathcal{L} = \emptyset$ en $\mathcal{L} = \Sigma^*$ kun je echter geen zo'n woorden vinden, en kan een reductie nooit een output genereren dat wel resp. niet in $\mathcal{L}$ zit.

99. Beschouw de taal $\mathcal{L} = \{w \in \{0,1\}^* : \#_0(w) = \#_1(w)\}$.

1. Beschrijf een Turingmachine met één band die deze taal beslist in tijd $\mathcal{O}(n^2)$.

   Op input w met lengte $|w| = n$, doe het volgende.

   (1) In tijd $\mathcal{O}(n)$: controleer of de input tot $\{0,1\}^*$ behoort.

   (2) In tijd $\mathcal{O}(n)$: zoek het eerste symbool 0 of 1 op de band. Indien geen gevonden, aanvaard de input; indien wél gevonden, markeer die.

   (3) In tijd $\mathcal{O}(n)$: zoek een matchend symbool 1 resp. 0 op de band. Indien geen gevonden, verwerp de input, indien wél gevonden, markeer die.

   (4) Herhaal vanaf stap (2).

   Een woord w in de taal wordt aanvaard na in totaal $\mathcal{O}(n)$ iteraties, die elk tijd $\mathcal{O}(n)$ kosten, wat een totale tijdscomplexiteit $\mathcal{O}(n^2)$ geeft.

2. Beschrijf een Turingmachine met twee banden die deze taal beslist in tijd $\mathcal{O}(n)$.

   Op input w met lengte $|w| = n$, doe het volgende.

   (1) In tijd $\mathcal{O}(n)$: controleer of de input tot $\{0,1\}^*$ behoort.

   (2) In tijd $\mathcal{O}(n)$: overloop de input en kopieer elke 1 naar de tweede band.

(3) In tijd $\mathcal{O}(n)$: keer op beide banden terug naar de start.

(4) In tijd $\mathcal{O}(n)$: overloop de eerste band en zet voor elke 0 ook een stap op de tweede band. Verwerp zodra een probleem opduikt; aanvaard als beide banden overlopen zijn.

Hier hebben we geen iteraties nodig en blijft de tijdscomplexiteit lineair: $\mathcal{O}(n)$.

Een opmerkelijk resultaat van Fred Hennie uit 1965 stelt dat Turingmachines met slechts één band die hun input in lineaire tijd beslissen, niet krachtiger zijn dan EDA's! Deze oefening toont dus aan dat Turingmachines met meerdere banden wel degelijk (strikt) efficiënter zijn.

100. 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.

Een formule in conjunctieve normaalvorm (cnf) is van de vorm

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

Een formule in disjunctieve normaalvorm (dnf) is van de vorm

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

Elk symbool $\bullet$ stelt hier een literaal voor, i.e. een atoom of zijn negatie.

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

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

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 clausule, maar deze extra controle kost slechts lineaire tijd.

CNF-SAT $\leqslant_m$ 3SAT.
We zoeken een constructie die een gegeven Booleaanse formule ϕ transformeert in een formule met drie literalen per clausule. Noteer $\phi = \rho_1 \wedge \rho_2 \wedge \cdots \wedge \rho_n$ waar de ρ_i de clausules zijn. Elke ρ_i zullen we omzetten naar een conjunctie van nieuwe clausules met drie literalen, met als variabelen enerzijds de variabelen uit ρ_i en anderzijds nieuwe variabelen die enkel in deze nieuwe clausules 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 $\rho_i = \lambda_{i1} \vee \lambda_{i2} \vee \cdots \vee \lambda_{ik_i}$ (met λ_{ij} de literalen) en maak volgend onderscheid naargelang de waarde van $k = k_i$.

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

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

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

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

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

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

$$\widetilde{\rho}_i = (\lambda_{i1} \vee \lambda_{i2} \vee z_{i1}) \wedge (\neg z_{i1} \vee \lambda_{i3} \vee z_{i2}) \wedge (\neg z_{i2} \vee \lambda_{i4} \vee z_{i3}) \wedge \cdots \\ \wedge (\neg z_{i,k-4} \vee \lambda_{i,k-2} \vee z_{i,k-3}) \wedge (\neg z_{i,k-3} \vee \lambda_{i,k-1} \vee \lambda_{ik}).$$

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

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

Merk nu op dat de originele formule $\phi = \rho_1 \wedge \rho_2 \wedge \cdots \wedge \rho_n$ vervulbaar is als en slechts als de nieuwe formule $\widetilde{\phi} = \widetilde{\rho}_1 \wedge \widetilde{\rho}_2 \wedge \cdots \wedge \widetilde{\rho}_n$ vervulbaar is. Voor de constructies bij $k \leqslant 3$ 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 literaal λ_{im} in deze clausule (met $1 \leqslant m \leqslant k$) die `True` blijkt. Stel dan alle hulpvariabelen z_{ij} met $j \leqslant m - 2$ (in de clausule vóór die met λ_{im}) op `True` en alle hulpvariabelen z_{ij} met $j \geqslant m - 1$ (in de clausule ná die met λ_{im}) op `False`. Alle voorgaande en alle volgende clausulen worden dan eveneens `True`, dus we hebben de vervullende invulling van ρ_i uitgebreid naar een vervullende invulling van $\widetilde{\rho}_i$ (zonder effect op de overige clausules).

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

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

CNF-SAT $\leqslant_{\mathsf{P}}$ 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 clausule wordt omgevormd naar respectievelijk vier, twee en één clausules, zodat de lengte lineair toeneemt. Voor een clausule met $k > 3$ literalen bouwen we $k - 2$ nieuwe clausules, zodat de lengte kwadratisch toeneemt. Samen kost het opstellen van de nieuwe formule polynomiale tijd.

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

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

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?

Helaas, zo eenvoudig valt P = NP niet te bewijzen: het algoritme dat de ene normaalvorm naar de andere ombouwt, kan de lengte van de formule exponentieel doen toenemen.

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

104. 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-kliek in een graaf een verzameling van k onderling adjacente toppen is.

We moeten drie beweringen aantonen: dat CLIQUE in NP zit, dat 3SAT reduceert naar CLIQUE, en dat deze reductie in polynomiale tijd lukt.

CLIQUE *behoort tot* NP.

Deze is vrij eenvoudig. We kunnen op twee manieren te werk gaan: enerzijds kunnen we een niet-deterministische *decider* voor CLIQUE beschrijven, of anderzijds een deterministische *verifier*.

Een niet-deterministische decider werkt als volgt. Controleer of de input van de vorm $\langle G, k\rangle$ is en verwerp indien niet. Overloop alle toppen en kies voor elke top willekeurig of we die al dan niet markeren. Controleer vervolgens of er precies k toppen gemarkeerd werden en alle gemarkeerde toppen onderling adjacent zijn (dit lukt in kwadratische tijd t.o.v. het aantal toppen). Indien wel, aanvaard; indien niet, verwerp.

Een deterministische verifier werkt als volgt. Controleer of de input van de vorm $\langle\langle G, k\rangle, c\rangle$ is, en of het certificaat c een deelverzameling van precies k toppen codeert. Controleer vervolgens of die allen onderling adjacent zijn (dit lukt in kwadratische tijd t.o.v. het aantal toppen). Indien wel, aanvaard; indien niet, verwerp.

3SAT $\leqslant_m$ CLIQUE.

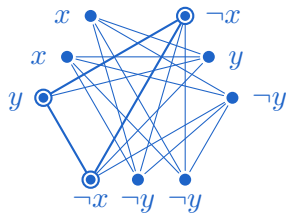
We willen een gegeven Booleaanse formule ϕ in conjunctieve normaalvorm (met drie literalen per clausule) omvormen naar een graaf en een parameter k, zodanig dat ϕ vervulbaar is als en slechts als de geconstrueerde graaf G een k-kliek heeft.

Een natuurlijk idee is om de literalen in ϕ als toppen van de graaf te zien, en te willen dat een kliek overeenkomt met de literalen die `True` gezet moeten worden. Om vervulbaar te zijn, moet er een invulling bestaan zodat in elke clausule (minstens) één literaal `True` gezet wordt. We willen dus een kliek met één literaal per clausule. Dit motiveert waarom we k gelijk aan het aantal clausules willen stellen en een k-partiete graaf construeren (waarvan de partitieklassen overeenkomen met de clausules). Wat verder puzzelen leidt tot de volgende concrete constructie:

1. Stel k gelijk aan het aantal clausules in ϕ.
2. Construeer een top van G voor elke literaal (z of $\neg z$) in ϕ.
3. Voeg géén boog toe tussen toppen die overeenkomen met literalen van dezelfde clausule.
4. Voeg een boog toe tussen literalen van verschillende clausules precies als deze tegelijkertijd vervulbaar zijn, i.e. niet van de vorm z en $\neg z$.
5. Codeer het resultaat in de output $\langle G, k\rangle$.

Om een voorbeeld te geven (met $k = 3$),

$$\phi = (\neg x \vee y \vee \neg y) \wedge (x \vee x \vee y) \wedge (\neg x \vee \neg y \vee \neg y) \quad \Rightarrow$$



In dit voorbeeld bevat de kliek literalen $\{y, \neg x, \neg x\}$, dus de invulling $x =$ `False`, $y =$ `True` werkt.

Overtuig jezelf dat deze constructie de gezochte reductie geeft, i.e. dat ϕ vervulbaar is als en slechts als G een k-kliek heeft, of dus dat $\langle\phi\rangle \in$ 3SAT als en slechts als het beeld $\langle G, k\rangle \in$ CLIQUE.

3SAT $\leqslant_{\mathsf{P}}$ CLIQUE.

Tot slot moeten we nog beargumenteren dat deze constructie in polynomiale tijd (ten opzicht van de inputgrootte $|\langle\phi\rangle|$) kan. Dat spreekt redelijk voor zich: k bepalen en de toppen van G opstellen lukt in lineaire tijd, de bogen in kwadratische tijd.

Opmerking: het is belangrijk om het eerste puntje, namelijk het NP*-zijn, niet te vergeten. De reductie op zich is niet voldoende, want die toont enkel aan dat* CLIQUE *van de klasse* NP*-hard is.*

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

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

Deze is na de vorige oefening niet moeilijk meer. Op een analoge manier als in de vorige oefening zien we dat INDEPENDENT-SET in de klasse NP zit. We kunnen dan reduceren vanuit CLIQUE via de complementaire graaf: gegeven een input $\langle G, k\rangle$, construeer de complementaire graaf G' (met dezelfde toppen, maar twee toppen zijn adjacent in G' als en slechts als ze dat *niet* zijn in G), en schrijf $\langle G', k\rangle$. Deze reductie werkt omdat een onafhankelijke verzameling in een graaf precies een kliek in zijn complementaire graaf is, en omgekeerd.

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

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

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.

Beredeneer dat V' een *vertex cover* is als en slechts als $V \setminus V'$ een onafhankelijke verzameling is.

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

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

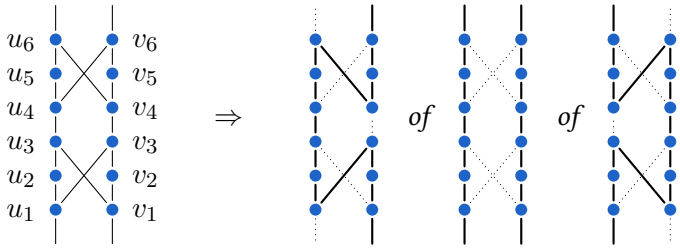
Gegeven $\langle G, k\rangle$ (voor VERTEX-COVER), construeer $\langle |V(G)|, |E(G)|, \{C_i\}, k\rangle$ (HITTING-SET) waarin $C_i = \{u, v\}$ de i-de boog voorstelt.

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

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

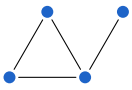
Herinner je dat een graaf Hamiltoniaans wordt genoemd als er een cykel bestaat die alle toppen precies één keer aandoet.

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.



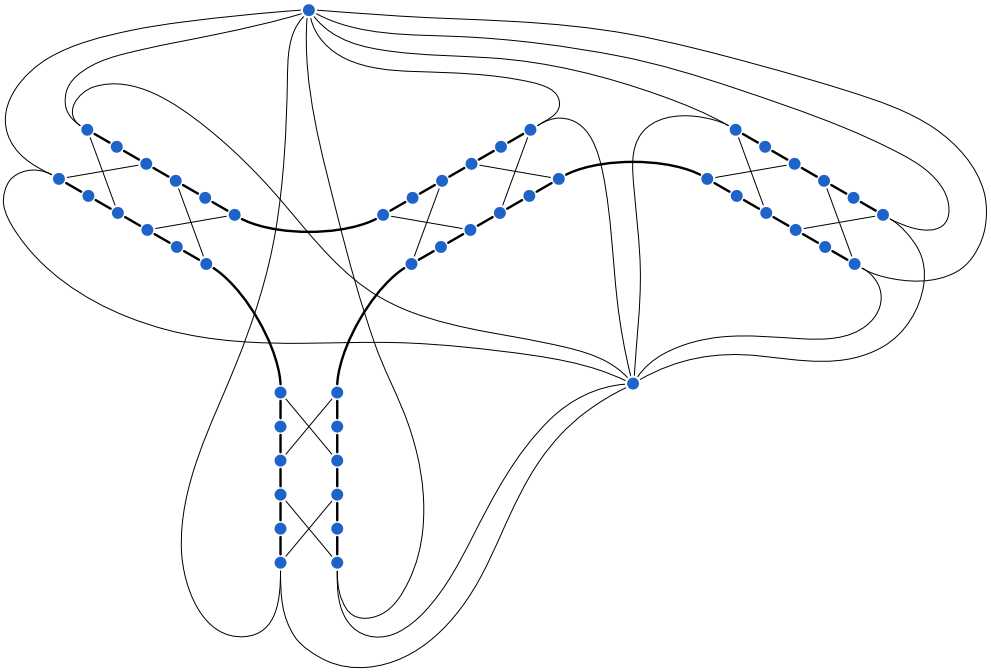
Merk nog op dat een Hamiltoncykel doorheen zo'n component enkel mogelijk is op de drie afgebeelde manieren rechts; in het bijzonder verlaat die iedere component langs dezelfde kant als binnengegaan.

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

$G =$ 

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

$G' =$



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

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

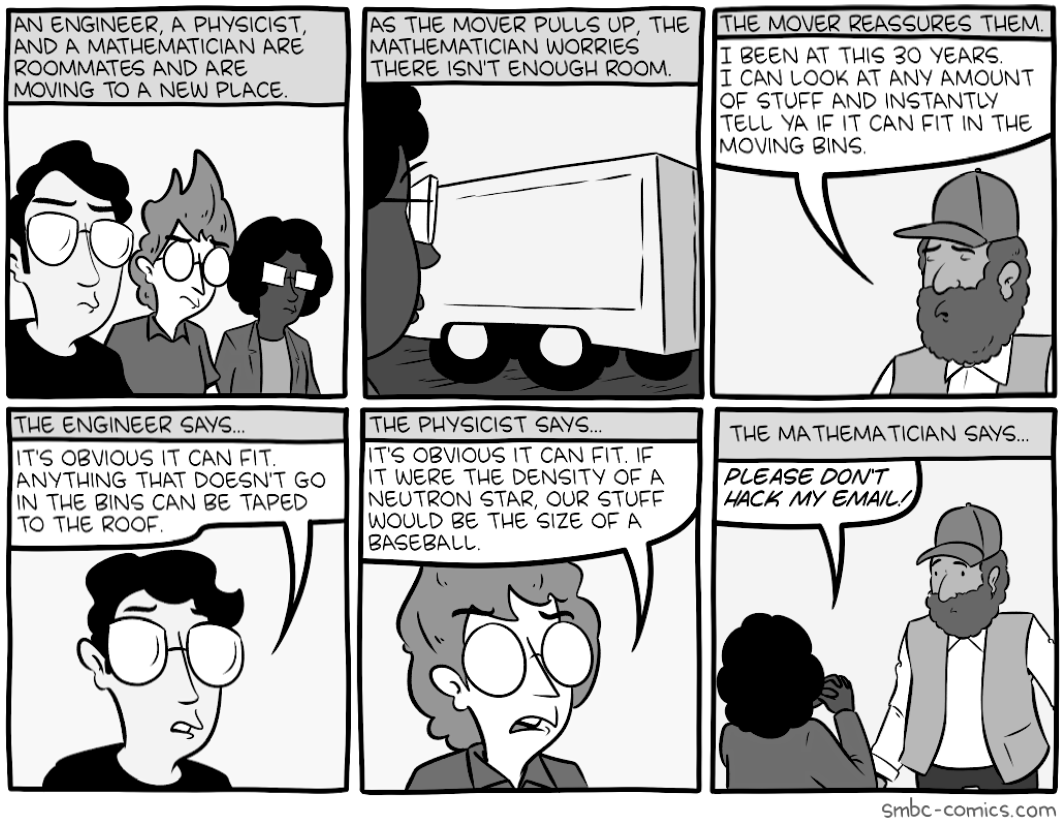
$$\mathsf{TSP} = \{\langle G, k\rangle : G \text{ is een gewogen graaf met een Hamiltoncykel van totaal gewicht } \leqslant k\}.$$

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

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

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 natuurlijk optredende problemen computationeel zeer zwaar kunnen zijn. Zie* https://en.wikipedia.org/wiki/Karp%27s_21_NP-complete_problems.

110. Verklaar de punchline in deze SMBC-comic (`https://www.smbc-comics.com/comic/bins`).



Het *bin packing problem* vraagt om, gegeven een aantal objecten met zekere volumes en een aantal dozen met zeker totaalvolume, te beslissen of de objecten verdeeld kunnen worden over de dozen zonder hun capaciteit te overschrijden. Het probleem duikt op in allerlei (abstracte én praktische) situaties, maar is helaas NP-compleet.

De verhuizer beweert eigenlijk dat hij een orakel voor het *bin packing problem* is, want hij stelt elke instantie zonder enige inspanning te kunnen beslissen. De wiskundige weet dat *bin packing* NP-compleet is, zodat de verhuizer ook alle andere NP-problemen zou kunnen oplossen in polynomiale tijd (via een geschikte reductie).

Dankzij hun computationele onhaalbaarheid maken heel wat cryptografische systemen gebruik van gekende NP-complete problemen om veiligheid te "garanderen". Dus, hier zou de verhuizer zijn orakelcapaciteit kunnen gebruiken om deze systemen efficiënt te kraken.

* **111.** **[Examen 2018]** Bewijs dat de volgende taal tot de complexiteitsklasse P behoort:

$$\left\{ \langle M \rangle : \begin{array}{l} M \text{ is een niet-deterministische eindigetoestandsautomaat} \\ \text{die oneindig veel woorden } a^n \text{ (waarbij } n \in \mathbb{N}\text{) aanvaardt} \end{array} \right\}.$$

Een high-level implementatie is voldoende—je hoeft niet alles tot in detail op een Turingmachine met één band te implementeren.

Extra oefeningen

93. Zijn volgende talen regulier? Contextvrij? Bewijs.

1. **[Examen 2011]**
$$\{a^n b^n c^m : m, n \geqslant 0\} \cap \{a^m b^{2n} c^n : m, n \geqslant 0\}$$

2. **[Examen 2014]**
$$\{w \in \{a, b, c\}^* : (\#_a(w) + \#_b(w)) \cdot \#_c(w) = 6\}$$

3. **[Examen 2014]**
$$\{w \in \{0, 1, 2\}^* : \#_1(w) \bmod 3 = 1 \text{ en } 3\,\#_1(w) - \#_0(w) \geqslant 1 \text{ en } 221 \text{ is een deelwoord}\}$$

4. **[Examen 2015]**
$$\{w \in \{2\}^*\{0, 1\}^* : \#_0(w) < 2\,\#_1(w) + 1\}$$

5. **[Examen 2015]**
$$\mathcal{L}^{\mathsf{R}} \quad \text{met} \quad \mathcal{L} = \{w \in \{a, b, c, d\}^* : \#_a(w)^2 > \#_b(w) \cdot \#_c(w)\}$$

6. **[Examen 2015]**
$$\{w \in \{x, y, z\}^* : \text{alle } x \text{ komen voor alle } y \text{ en } \#_x(w) + 3\,\#_y(w) = 2\,\#_z(w) + 1\}$$

7. **[Examen 2017]**
$$\{w \in \{x, y, z\}^* : \#_y(w) + \#_z(w) \geqslant \#_x(w) \cdot \#_z(w)\}$$

8. **[Examen 2017]**
$$\{xyzy : x, y, z \in \{0, 1\}^+\}$$

9. **[Examen 2017]**
$$\mathcal{L}^* \quad \text{met} \quad \mathcal{L} = \left\{ w \in \{a, b, c, d\}^* : \begin{array}{l} \#_a(w) \leqslant \#_c(w) \cdot \#_d(w), \\ \#_b(w) + \#_c(w) \leqslant \#_d(w)^2, \\ \#_d(w) \leqslant 15 \end{array} \right\}$$

10. **[Examen 2018]**
$$\{w \in \{a, b, c\}^* : 2\,\#_a(w) \bmod 3 = \big(\#_b(w) - \#_c(w)\big) \bmod 3\}$$

11. **[Examen 2018]**
$$\left\{ w \in \{a, b, \circ\}^* : \begin{array}{l} w = w_1 \circ w_2 \circ \cdots \circ w_n \text{ met elke } w_i \in \{a, b\}^+ \\ \text{en er bestaan } 1 \leqslant i \neq j \leqslant n \text{ waarbij } w_i = w_j^{\mathsf{R}} \end{array} \right\}$$

12. **[Examen 2018]**
$$\{a^m b^n a^m b^n : m > 0, n > 0\}$$

94. [Examen 2018] Beschouw de volgende operator `Palindromify` op formele talen:
$$\texttt{Palindromify}(\mathcal{L}) = \{ww^{\mathsf{R}} : w \in \mathcal{L}\}.$$
Bewijs dat het beeld $\texttt{Palindromify}(\mathcal{L})$ van een reguliere taal $\mathcal{L}$ niet altijd opnieuw regulier is, maar wel altijd contextvrij. Wat is het verschil met de taal $\mathcal{L}\mathcal{L}^{\mathsf{R}}$?

95. [Examen 2014] Wat verandert er als we PDA's met oneindig veel toestanden toelaten; worden nog exact dezelfde talen geaccepteerd?

96. [Examen 2014] Toon aan dat de volgende grammatica dubbelzinnig is, en geef een equivalente ondubbelzinnige grammatica.

$$\begin{cases} S \to AS \mid B, \\ A \to bC, \\ B \to aB \mid \varepsilon, \\ C \to aC \mid a. \end{cases}$$

97. [Examen 2014] Gegeven de grammatica

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

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

98. [Examen 2014] Geef een deterministische Turingmachine met één band (in macronotatie) die de volgende taal beslist:

$$\{w \in \{0,1,2,3,4\}^* : \#_0(w) \cdot \#_1(w) = \#_2(w) \text{ en } \#_3(w) > 1 \text{ en de 2's komen voor de 4's}\}.$$

99. [Examen 2014] Zij $\mathcal{L}$ een reguliere taal, en veronderstel dat $\mathcal{L}' \leqslant_m \mathcal{L}$. Is $\mathcal{L}'$ dan automatisch ook regulier? Is $\mathcal{L}'$ contextvrij? Is $\mathcal{L}'$ beslisbaar? Verklaar je antwoorden.

100. [Examen 2015] Onderstel dat $\mathcal{L} \leqslant_m \mathcal{L}' \leqslant_m \mathcal{L}''$ en dat $\mathcal{L}'$ contextvrij is. Wat kunnen we besluiten over $\mathcal{L}$ en $\mathcal{L}''$?

101. [Examen 2013] Toon via een reductie (dus zonder toepassing van de stelling van Rice) aan dat

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

102. [Examen 2016] Toon via een reductie (dus zonder toepassing van de stelling van Rice) aan dat

$$\{\langle M\rangle : |\mathcal{L}(M)| = 2016\} \notin \mathsf{D}.$$

103. [Examen 2017] Toon via een reductie (dus zonder toepassing van de stelling van Rice) aan dat

$$\mathcal{L} = \{\langle M\rangle : M \text{ aanvaardt minstens 1001 inputs en weigert minstens 10001 inputs}\} \in \mathsf{SD} \setminus \mathsf{D}.$$

104. [Examen 2018] Is de volgende taal beslisbaar of niet? Bewijs je antwoord.

$$\{\langle M\rangle : M \text{ stelt een Turingmachine voor met } |\mathcal{L}(M)| \leqslant 2018 \text{ en } |\langle M\rangle| \leqslant 2018\}$$

We coderen M tot $\langle M\rangle$ zoals in de cursus, dus over het alfabet

$$\Sigma = \{0,\ 1,\ q,\ a,\ (,\ ,,\ ),\ \leftarrow,\ \rightarrow\}.$$

105. [Examen 2018] Toon aan dat de volgende taal (a) wel semibeslisbaar maar (b) niet beslisbaar is:

$$\{\langle M, w_1, w_2, w_3\rangle : M \text{ stopt op minstens één van de woorden } w_1, w_2 \text{ of } w_3\}.$$

Maak voor (b) gebruik van een expliciete reductie.