1. Zijn de volgende talen (a) regulier of niet, (b) contextvrij of niet, en (c) beslisbaar of niet?

1. $\left\{ w \in \{a,b,c,d\}^* : \begin{array}{l} \#_a(w) + \#_c(w) \leq 2018,\ \#_b(w) + \#_d(w) \leq 2018, \\ \text{en } \#_a(w), \#_b(w), \#_c(w) \text{ en } \#_d(w) \text{ zijn allen priem} \end{array} \right\}$,

   Merk op dat voor elk woord w in deze taal geldt dat $|w| = \#_a(w) + \#_b(w) + \#_c(w) + \#_d(w) \leq 4036$, zodat de taal eindig en dus triviaal regulier (contextvrij, beslisbaar) is.

2. $\{a^m b^n a^m b^n : m > 0, n > 0\}$,

   Deze taal is niet contextvrij. Veronderstel van wel, zij k de constante uit het *pumping lemma* voor contextvrije talen en beschouw $w = a^k b^k a^k b^k$. Volgens het pumping lemma kunnen we w schrijven als $uvxyz$ met $vy \neq \varepsilon$, $|vxy| \leq k$ en $w_q = uv^q xy^q z \in \mathcal{L}$ voor alle q.

   - Als v of y uit zowel a's als b's bestaat, gaat na oppompen de vorm $a^+b^+a^+b^+$ kapot, zodat w_2 niet langer tot $\mathcal{L}$ behoort; een strijdigheid.
   - Als $v = a^{|v|}$ en $y = b^{|y|}$ of omgekeerd, dan zal één van de twee a-blokken en één van de twee b-blokken in w worden opgepompt en behoort w_2 niet langer tot $\mathcal{L}$; een strijdigheid.
   - Als $v = a^{|v|}$ en $y = a^{|y|}$, dan kunnen v en y niet in twee verschillende a-blokken zitten: omdat er minstens een b-blok met lengte k tussen zit, zou dan $|vxy| \geq |v| + |y| + k > k$, in strijd met $|vxy| \leq k$. Dus ook in dit geval zal slechts één a-blok worden opgepompt en behoort w_2 niet langer tot $\mathcal{L}$; een strijdigheid.
   - Als $v = b^{|v|}$ en $y = b^{|y|}$ vinden we op analoge wijze een strijdigheid.

   In elk geval vinden we een strijdigheid, dus deze taal is niet contextvrij en al zeker niet regulier.

   Ze is wel beslisbaar, door een Turingmachine die eerst controleert of de input van de vorm $a^+b^+a^+b^+$ is, daarna telkens in elk a-blok een a markeert, daarna telkens in elk b-blok een b markeert, en zo onderweg kan verifiëren of de twee paren blokken even groot zijn.

   Een alternatieve en elegante manier om de beslisbaarheid aan te tonen, gesuggereerd door een student, is gebruik maken van een automaat met een FIFO-stapel (first in, first out). Lees zo het eerste blok van a's en b's terwijl deze op de stapel gezet worden, en daarna het tweede blok van a's en b's terwijl ze gematcht worden met de symbolen *onderop* de stapel. Zo'n automaat kan de taal beslissen, en is in rekenkracht equivalent aan een Turingmachine.

   Nog een andere elegante manier, gesuggereerd door een student, is een PDA met twee stapels: één voor de a's en één voor de b's. Ook zo'n automaat is equivalent aan een Turingmachine.

Bewijs telkens je antwoord.

2. 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 $\mathcal{L}$ (a) niet altijd opnieuw regulier is, maar (b) wel altijd contextvrij.

$\texttt{Palindromify}(\mathcal{L}(a^*b)) = \{a^k bba^k : k \leq 0\}$ is duidelijk niet-regulier (pumping lemma).

Zij $\mathcal{L}$ een reguliere taal en beschouw een EDA M voor $\mathcal{L}$. We bouwen de EDA om tot een PDA M' voor $\texttt{Palindromify}(\mathcal{L})$. Het idee is om eerst de input te verwerken zoals M doet, maar ondertussen ook de gelezen letters op de stapel te pushen; wanneer we in een aanvaardende toestand zitten (zodat de gelezen prefix tot M behoort) kan de PDA gokken op het midden van de input te zitten en daarna overgaan naar een aparte toestand, waarin we de rest van de input inlezen en ondertussen vergelijken met de stapel.

Concreet: de toestanden van M' zijn die van M plus één extra toestand q. We maken enkel q aanvaardend in M'. Voor elke transitie $\delta(p_1, x) = p_2$ in M voegen we een transitie $((p_1, x, \varepsilon), (p_2, x))$ toe in M', die het gelezen symbool op de stapel pusht. We voegen daarna vanuit elke *aanvaardende* toestand in M een ε-transitie naar q toe in M', en ten slotte nog de transities $((q, x, x), (q, \varepsilon))$ voor elke $x \in \Sigma$.

Voor deze PDA geldt dat $\mathcal{L}(M') = \texttt{Palindromify}(\mathcal{L})$.

Een alternatieve manier is om een reguliere grammatica voor $\mathcal{L}$ om te vormen tot een contextvrije grammatica voor $\texttt{Palindromify}(\mathcal{L})$, als volgt.

- Laat elke herschrijfregel van de vorm $S \to \varepsilon$ ongewijzigd.
- Vervang elke herschrijfregel van de vorm $S \to a$ door $S \to aa$.
- Vervang elke herschrijfregel van de vorm $S \to aT$ door $S \to aTa$.

Waarom resulteert dit in een grammatica voor $\texttt{Palindromify}(\mathcal{L})$?

3. 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.

Een Turingmachine die deze taal $\mathcal{L}$ semisbeslist, werkt als volgt. Controleer of de input van de vorm $\langle M, w_1, w_2, w_3\rangle$ is en verwerp meteen indien niet. Daarna simuleren we de werking van M op de drie woorden. Merk op dat we niet kunnen proberen om w_1 volledig te verwerken alvorens w_2 te controleren, dus we moeten de drie simulaties in parallel uitvoeren (een stap op w_1, een stap op w_2, een stap op w_3, dan een volgende stap op w_1, etc.). Wanneer een van de drie simulaties stopt, kan de Turingmachine de input $\langle M, w_1, w_2, w_3\rangle$ aanvaarden.

Merk op dat dit niet het principe van dovetailing is, maar wel het principe van parallelle simulatie.

Een reductie vanuit het *halting problem* is eenvoudig. De reductiemachine R controleert of de input van de vorm $\langle M, w\rangle$ is. Indien niet, geeft R een vaste output terug die niet tot $\mathcal{L}$ behoort, en indien wel, geeft R output $\langle M, w, w, w\rangle$ terug. Het is duidelijk dat $x \in \mathcal{H}$ als en slechts als $R(x) \in \mathcal{L}$.

4. Bewijs dat de complexiteitsklasse P gesloten is onder concatenatie. Met andere woorden, bewijs dat als $\mathcal{L}_1 \in \mathsf{P}$ en $\mathcal{L}_2 \in \mathsf{P}$, dan ook de concatenatie $\mathcal{L}_1\mathcal{L}_2 = \{w_1w_2 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2\} \in \mathsf{P}$.

Gegeven een input w, moeten we kunnen verifiëren of $w \in \mathcal{L}_1\mathcal{L}_2$ in een polynomiale tijd t.o.v. $|w|$. Met andere woorden, we moeten verifiëren of we deze w kunnen schrijven als w_1w_2 met $w_1 \in \mathcal{L}_1$ en $w_2 \in \mathcal{L}_2$—we krijgen deze decompositie niet mee in de input!

Omdat $\mathcal{L}_1 \in \mathsf{P}$ en $\mathcal{L}_2 \in \mathsf{P}$ bestaan er twee Turingmachines M_1, M_2 en twee polynomen p_1, p_2 zodat M_k in tijd $p_k(|x|)$ beslist of een arbitraire x tot $\mathcal{L}_k$ behoort (voor $k = 1, 2$).

Beschouw nu het volgende procedure in pseudocode:

```
for i ∈ {0, 1, ..., |w|}:
  write w = w₁w₂ with |w₁| = i, |w₂| = |w| − i
  if w₁ ∈ L₁ and w₂ ∈ L₂:
    accept w
reject w
```

Het is duidelijk dat deze de concatenatie $\mathcal{L}_1\mathcal{L}_2$ beslist. Iedere tussenstap in de `for`-lus kost een tijd $p_1(|w_1|) + p_2(|w_2|) = \mathcal{O}(p_1(|w|) + p_2(|w|))$, en er zijn $|w| + 1$ dergelijke lussen, zodat het algoritme essentieel een tijd $\mathcal{O}(|w| \cdot (p_1(|w|) + p_2(|w|)))$ vergt—opnieuw polynomiaal in $|w|$, één graad hoger.