Ce devoir est d'une durée de 4h, et est composé de deux parties indépendantes :

- la partie 1 porte sur le problème de correspondance de Post et des problèmes indécidables sur les grammaires non contextuelles, cette partie contient quelques questions de programmation à traiter dans le langage C ;
- la partie 2 porte sur divers problèmes NP-complets sur les graphes et les grammaires, cette partie contient quelques questions de programmation à traiter dans le langage OCaml.

Le barème tient compte de la clarté et de la présentation de votre copie. On prendra garde à indiquer clairement le numéro des questions, et, autant que possible, à rédiger les questions dans l'ordre. Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat.

1 Problèmes indécidables sur les grammaires non contextuelles

Dans cette partie, on considère Σ un alphabet fini. Le but de cette partie est de prouver l'indécidabilité de plusieurs problèmes de décision sur les grammaires non contextuelles.
Pour cela, on introduit d'abord un problème de décision très classique et connu pour être indécidable.

1.1 Le Problème de Correspondance de Post

Définition 1.1 : Problème de Correspondance de Post

On appelle **domino** un couple de mots $(u, v) \in (\Sigma^\star)^2$, qu'on note de la manière suivante : $\binom{u}{v}$.
Le **Problème de Correspondance de Post** (PCP) est le problème de décision suivant :

PCP	
Entrée :	une liste $\left[\binom{u_1}{v_1}, \dots, \binom{u_n}{v_n}\right]$ de dominos ;
Question :	$\exists k \in \mathbb{N}^\star,\ \exists (i_1, \dots, i_k) \in [\![1, n]\!]^k,\ u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$?

On cherche donc une suite finie (avec répétitions) de dominos telle que la concaténation de la partie du haut de chaque domino forme le même mot que la concaténation de la partie du bas.

Exemple 1.1

Considérons la liste de dominos suivante :

$$\Big[\underbrace{\binom{bc}{ca}}_{d_1}, \underbrace{\binom{a}{ab}}_{d_2}, \underbrace{\binom{ca}{a}}_{d_3}, \underbrace{\binom{abc}{c}}_{d_4}\Big]$$

Cette instance de PCP est positive ; voici une solution :

$$i_1 = 2\ ; \quad i_2 = 1\ ; \quad i_3 = 2\ ; \quad i_4 = 4$$

$$\binom{a}{ab} \cdot \binom{bc}{ca} \cdot \binom{a}{ab} \cdot \binom{abc}{c} = \binom{abcaabc}{abcaabc}$$

1. L'instance de PCP suivante possède-t-elle une solution ? Si oui, la donner ; sinon, justifier.

$$\Big[\underbrace{\binom{a}{baa}}_{d_1}, \underbrace{\binom{ab}{aa}}_{d_2}, \underbrace{\binom{bba}{bb}}_{d_3}\Big]$$

2. L'instance de PCP suivante possède-t-elle une solution ? Si oui, la donner ; sinon, justifier.

$$\Big[\underbrace{\binom{abc}{ab}}_{d_1}, \underbrace{\binom{ca}{a}}_{d_2}, \underbrace{\binom{acc}{ba}}_{d_3}\Big]$$

3. Montrer que si une instance de PCP possède une solution, alors elle possède une infinité de solutions.

1.2 Semi-décidabilité de PCP

On admet le théorème suivant (qui se prouve en réduisant le problème de l'arrêt des machines de Turing) :

Théorème 1.1

PCP est indécidable.

Cela vient du fait qu'il n'y a aucune borne sur la taille d'une solution par rapport à la taille de l'instance.

Exemple 1.2

L'instance de PCP suivante possède des solutions, mais sa plus petite solution utilise 75 dominos :

$$\left[\underbrace{\binom{aab}{a}}_{d_1}, \underbrace{\binom{a}{b}}_{d_2}, \underbrace{\binom{b}{aab}}_{d_3}\right]$$

On se propose cependant de montrer que PCP est **semi-décidable**, c'est-à-dire d'écrire un programme prenant en entrée une instance de PCP et qui :

- termine et renvoie `true` si l'instance possède une solution ;
- termine et renvoie `false`, ou ne termine pas, si l'instance ne possède pas de solution.

On se propose d'implémenter un tel algorithme en C. On suppose qu'on a importé les librairies usuelles : `stdbool.h`, `stdlib.h`. On rappelle qu'en C, une chaine de caractères de longueur n est représentée par un tableau `s` de type `char*` d'au moins $n+1$ cases tel que `s[n]` contienne le caractère spécial `'\0'` (les caractères situés après ce symbole sont ignorés).

4. Écrire une fonction `int longueur(char* s);` prenant en argument une chaine de caractères `s` et renvoyant sa longueur.
   Remarque : il est interdit d'utiliser la fonction `strlen` de la librairie `string.h` ; il s'agit de réimplémenter vous même cette fonction.
5. Écrire une fonction `bool egales(char* s1, char* s2);` prenant en arguments deux chaines de caractères et renvoyant `true` si elles sont égales (et `false` sinon).
   Remarque : votre fonction ne devra parcourir qu'une seule fois les chaines `s1` et `s2`.
6. Écrire une fonction `bool prefixe(char* p, char* s);` prenant en arguments deux chaines de caractères et renvoyant `true` si `p` est un préfixe de `s` (et `false` sinon).
   Remarque : votre fonction ne devra parcourir qu'une seule fois les chaines `p` et `s`.
7. Écrire une fonction `char* concat(char* s1, char* s2);` prenant en arguments deux chaines de caractères et renvoyant une nouvelle chaine de caractères allouée sur le tas qui contient la concaténation de `s1` et `s2`.
   Remarque : on prendra garde au caractère spécial `'\0'`.

On s'intéresse ici à une variante bornée de PCP :

BPCP	
Entrée :	une liste $\left[\binom{u_1}{v_1}, \dots, \binom{u_n}{v_n}\right]$ de dominos, un entier $K \in \mathbb{N}^\star$;
Question :	$\exists k \leq K,\ \exists (i_1, \dots, i_k) \in [\![1, n]\!]^k,\ u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$?

8. On considère comme ensemble de certificats l'ensemble des listes d'entiers, et la fonction de vérification suivante :
   - on vérifie que la liste est de longueur $\leq K$, et que toutes ses valeurs sont bien dans $[\![1, n]\!]$;
   - on calcule la concaténation des dominos selon l'ordre indiqué par la liste et on vérifie si la partie du haut est égale à la partie du bas.

   Ce raisonnement permet-il de conclure que BPCP $\in$ NP ? Justifier.

On modélise en C une liste $\left[\binom{u_0}{v_0}, \dots, \binom{u_{n-1}}{v_{n-1}}\right]$ de n dominos par deux tableaux u et v de type `char**` tels que $\forall i \in [\![0, n-1]\!]$, `u[i]` contienne la chaine u_i et `v[i]` contienne la chaine v_i.

9. Écrire une fonction `bool bpcp(char** u, char** v, int n, int K);` prenant en arguments un nombre n de dominos, deux tableaux u et v de longueur n représentant la liste des n dominos, et un entier K, et renvoyant `true` si c'est une instance positive de BPCP (et `false` sinon).
Indication : on pourra implémenter un brute-force, à l'aide d'une fonction auxiliaire effectuant un backtracking.
10. Écrire une fonction `bool pcp(char** u, char** v, int n);` prenant en arguments un nombre n de dominos, deux tableaux u et v de longueur n représentant la liste des n dominos, et qui semi-décide PCP, c'est-à-dire qui :
    - termine et renvoie `true` si l'instance possède une solution ;
    - termine et renvoie `false`, ou ne termine pas, si l'instance ne possède pas de solution.

1.3 Problèmes de décision sur les grammaires non contextuelles

On s'intéresse maintenant à plusieurs problèmes de décision sur les grammaires non contextuelles.

InterVide	
Entrée :	deux grammaires non contextuelles $\mathcal{G}_1$ et $\mathcal{G}_2$ sur le même alphabet Σ'
Question :	Est-ce que $\mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2) = \varnothing$?

Ambig	
Entrée :	une grammaire non contextuelle $\mathcal{G}$ sur un alphabet Σ'
Question :	Est-ce que $\mathcal{G}$ est ambigüe ?

Univ	
Entrée :	une grammaire non contextuelle $\mathcal{G}$ sur un alphabet Σ'
Question :	Est-ce que $\mathcal{L}(\mathcal{G}) = \Sigma'^\star$?

Égal	
Entrée :	deux grammaires non contextuelles $\mathcal{G}_1$ et $\mathcal{G}_2$ sur le même alphabet Σ'
Question :	Est-ce que $\mathcal{L}(\mathcal{G}_1) = \mathcal{L}(\mathcal{G}_2)$?

Nous allons montrer qu'ils sont indécidables en réduisant le problème de PCP vers ces problèmes et en utilisant le théorème 1.1.

11. Soit $f_1 : E_1 \to \mathbb{B}$ et $f_2 : E_2 \to \mathbb{B}$ deux problèmes de décision.
Montrer que si $f_1 \leq_{\text{Dec}} f_2$ et si f_1 est indécidable, alors f_2 est indécidable.

Soit $\left[\binom{u_1}{v_1}, \dots, \binom{u_n}{v_n}\right]$ une instance de PCP. On se donne $\Delta = \{d_1, \dots, d_n\}$ un ensemble de nouvelles lettres ($\Delta \cap \Sigma = \varnothing$) représentant nos n dominos, et on va travailler avec des grammaires sur l'alphabet $\Sigma' = \Sigma \cup \Delta$. Pour $L \subseteq \Sigma'^\star$, on note $\overline{L} = \Sigma'^\star \setminus L$ le complémentaire de L dans $\Sigma'^\star$.
On définit les langages suivants sur Σ' :

$$L_u = \{u_{i_1} \cdots u_{i_k} d_{i_k} \cdots d_{i_1} \mid k \in \mathbb{N}^\star \text{ et } (i_1, \dots, i_k) \in [\![1, n]\!]^k\}$$

$$L_v = \{v_{i_1} \cdots v_{i_k} d_{i_k} \cdots d_{i_1} \mid k \in \mathbb{N}^\star \text{ et } (i_1, \dots, i_k) \in [\![1, n]\!]^k\}$$

12. Donner une grammaire non contextuelle reconnaissant L_u.
13. (a) Si $\left[\binom{u_1}{v_1}, \dots, \binom{u_n}{v_n}\right]$ est une instance positive de PCP, que dire de L_u et L_v ?
La réciproque est-elle vraie ?
    (b) En déduire que PCP $\leq_{\text{Dec}}$ InterVide.

14. (a) Quel est le langage reconnu par la grammaire suivante ?

$$\begin{aligned} S &\to U \mid V \\ U &\to u_1 U d_1 \mid \cdots \mid u_n U d_n \\ U &\to u_1 d_1 \mid \cdots \mid u_n d_n \\ V &\to v_1 V d_1 \mid \cdots \mid v_n V d_n \\ V &\to v_1 d_1 \mid \cdots \mid v_n d_n \end{aligned}$$

(b) Montrer que PCP $\leq_{\text{Dec}}$ Ambig.

15. On considère les langages suivants :

$$L'_u = \{w d_{i_k} \cdots d_{i_1} \mid k \in \mathbb{N}^\star \text{ et } w \in \Sigma^\star \text{ et } w \neq u_{i_1} \cdots u_{i_k}\}$$

$$L'_v = \{w d_{i_k} \cdots d_{i_1} \mid k \in \mathbb{N}^\star \text{ et } w \in \Sigma^\star \text{ et } w \neq v_{i_1} \cdots v_{i_k}\}$$

(a) Que vaut $L'_u \cup \overline{\Sigma^\star \Delta^\star} \cup \{\varepsilon\}$?

(b) Donner une grammaire non contextuelle reconnaissant $\overline{\Sigma^\star \Delta^\star}$.

(c) Donner une grammaire non contextuelle reconnaissant L'_u.

16. Montrer que PCP $\leq_{\text{Dec}}$ Univ.

17. Montrer que Égal est indécidable.

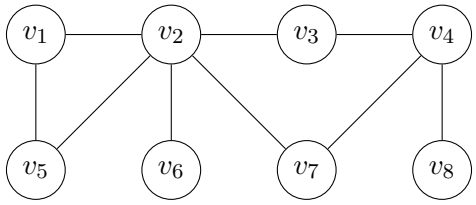
2 Problèmes NP-complets

2.1 Couverture de graphe

Définition 2.1 : ensemble indépendant/couverture

Soit $G = (S, A)$ un graphe non orienté, et soit $S' \subseteq S$.
On dit que S' est un **ensemble indépendant** de G si $\forall (u, v) \in S'^2,\ (u, v) \notin A$.
On dit que S' **couvre tous les sommets** de G si $\forall (u, v) \in A,\ u \in S'$ ou $v \in S'$.

Exemple 2.1



Dans le graphe ci-dessus :

- $\{v_1\}$ est un ensemble indépendant ;
- $\{v_2, v_4\}$ est un ensemble indépendant ;
- $\{v_1, v_6, v_3, v_8\}$ est un ensemble indépendant ;
- $\{v_1, v_6, v_4, v_7\}$ n'est pas un ensemble indépendant.
- $\{v_1, v_3, v_5, v_6, v_7, v_8\}$ est une couverture des sommets ;
- $\{v_2, v_4, v_5\}$ est une couverture des sommets ;
- $\{v_2, v_3, v_8\}$ n'est pas une couverture des sommets.

On observe sur ces exemples qu'il existe des solutions de taille diverses, et qu'il semble exister une taille critique pour un graphe donné à partir de laquelle on ne trouve plus de solutions : pour la recherche d'ensemble indépendant, c'est si la taille recherchée devient trop grande qu'on va finir par ne plus trouver de solution ; pour la recherche d'une couverture des sommets, c'est si la taille recherchée devient trop petite.

Ainsi, on s'intéresse aux deux problèmes de seuil suivants :

IndependentSet	
Entrée :	Un graphe non orienté $G = (S, A)$, un entier $K \leq \lvert S \rvert$
Question :	Existe-t-il un ensemble indépendant S' de G tel que $\lvert S' \rvert = K$?

VertexCover	
Entrée :	Un graphe non orienté $G = (S, A)$, un entier $K \leq \lvert S \rvert$
Question :	Existe-t-il une couverture S' des sommets de G tel que $\lvert S' \rvert = K$?

18. Montrer que IndependentSet $\in$ NP.
19. Montrer que VertexCover $\in$ NP.

On se propose maintenant de montrer que IndependentSet est NP-difficile. Pour cela, on va montrer que 3-Sat $\leq_{\mathsf{P}}$ IndependentSet. On rappelle l'énoncé de 3-Sat :

3-Sat	
Entrée :	une formule φ de la logique propositionnelle sous forme normale conjonctive, où chaque clause contient au plus 3 littéraux
Question :	φ est-elle satisfiable ?

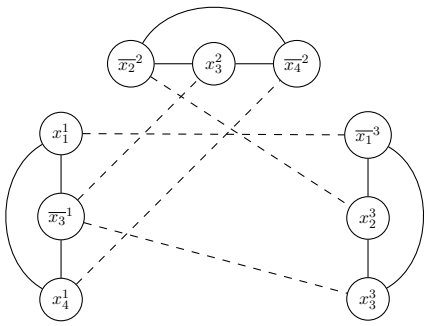
Si x est une variable propositionnelle, on note $\overline{x}$ le littéral $\neg x$. Pour montrer que 3-Sat $\leq_{\mathsf{P}}$ IndependentSet, étant donné une instance φ de 3-Sat, on souhaite construire en temps polynomial une instance de IndependentSet qui est positive si et seulement si φ est satisfiable.

Notons $\varphi = C_1 \wedge \cdots \wedge C_k$ où les C_j sont des clauses de taille ≤ 3. On construit un graphe G_φ comme suit :

- pour chaque clause C_j, pour chaque littéral ℓ_i apparaissant dans C_j, on crée un nouveau sommet ℓ_i^j (si un même littéral apparait dans plusieurs clauses, il y aura donc plusieurs sommets associés à ce littéral : un par clause concernée) ;
- pour chaque clause C_j, on relie entre eux tous les sommets associés à la même clause C_j ;
- pour chaque variable x_i, on relie par une arête chaque paire de sommets de la forme $(x_i^j, \overline{x_i}^{j'})$ (chaque copie d'un littéral avec chaque copie de sa négation).

Exemple 2.2

Pour $\varphi = (x_1 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$, on obtient le graphe suivant (les arêtes pleines sont issues de l'étape 2, les arêtes en pointillés sont issues de l'étape 3) :



20. Dessiner le graphe obtenu pour la formule suivante :

$$\varphi = (\overline{x_1} \vee x_2 \vee x_4) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4})$$

21. Montrer que φ est satisfiable si et seulement si G_φ possède un ensemble indépendant de taille k, où k est le nombre de clauses de φ.
22. En déduire que 3-Sat $\leq_{\mathsf{P}}$ IndependentSet.
23. En déduire que IndependentSet est NP-complet.

On se propose maintenant de montrer que VertexCover est NP-difficile.

24. Montrer que IndependentSet $\leq_P$ VertexCover.
25. En déduire que VertexCover est NP-complet.

2.2 Complexité grammaticale

Remarque : cette partie est inspirée d'un sujet de Nathaniel Carré (MPI* Louis Le Grand).

On s'intéresse maintenant à un problème de seuil sur les grammaires non contextuelles. Le but de cette dernière partie est de montrer qu'il est NP-complet.

2.2.1 Préliminaires

Définition 2.2 : grammaire acyclique

Une grammaire non contextuelle $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ est dite **acyclique** si, $\forall X \in \mathcal{V}$, si $X \Rightarrow^k \alpha$ avec $k > 0$, alors $|\alpha|_X = 0$.
Autrement dit, une dérivation non vide depuis X ne peut pas faire apparaitre à nouveau X.

Définition 2.3 : ordre topologique

Soit $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ une grammaire non contextuelle.
On appelle **ordre topologique** de $\mathcal{V}$ une énumération $(X_0, \dots, X_{n-1})$ des variables de $\mathcal{V}$ telle que $\forall (i,j) \in [\![0, n-1]\!]^2$, pour toute règle $X_i \to \alpha \in \mathcal{R}$, si $|\alpha|_{X_j}| > 0$, alors $i < j$.

26. Montrer qu'une grammaire est acyclique si et seulement si elle possède un ordre topologique.
27. En déduire que si $\mathcal{G}$ est une grammaire acyclique, alors $\mathcal{L}(\mathcal{G})$ est fini.

On se propose d'implémenter une grammaire $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ en OCaml. Pour simplifier, on représente Σ par le type `char` et $\mathcal{V}$ par le type `int` (on suppose que $\mathcal{V} = [\![0, n-1]\!]$ pour un certain $n \in \mathbb{N}^*$).

```
type symbole = T of char | V of int (* T pour terminal, V pour variable *)
type mot = symbole list
type grammaire = mot list array
```

Ainsi, avec les types définis ci-dessus, `T c` correspond au symbole terminal $c \in \Sigma$, et `V i` correspond à la i-ème variable de V. Un mot de $(\Sigma \cup \mathcal{V})^*$ est alors représentée par une liste de symboles, et une grammaire est représentée par un tableau `g` de taille n tel que $\forall X \in \mathcal{V}$, `g.(x)` contient la liste des mots α tels que $X \to \alpha \in \mathcal{R}$. Par convention, le symbole S correspondra toujours au symbole d'indice 0.

Exemple 2.3

La grammaire $\mathcal{G}_0$ définie sur $\Sigma = \{a, b\}$ et $\mathcal{V} = \{S, X\}$ par les règles :

$$S \to XXS \mid \varepsilon$$
$$X \to a \mid b$$

peut être définie par :

```
let g0 = [| [ [V 1; V 1; V 0]; [] ];
            [ [T 'a']; [T 'b'] ] |]
```

28. (a) Écrire une fonction `acyclique : grammaire -> bool` prenant en argument une grammaire et renvoyant `true` si elle est acyclique (et `false` sinon).
    (b) Déterminer la complexité temporelle de `acyclique` en fonction de $|\mathcal{V}|$, $|\mathcal{R}|$, et de la taille maximale m de la partie droite d'une règle de $\mathcal{R}$.

Définition 2.4 : grammaire élémentaire

Une grammaire non contextuelle $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ est dite **élémentaire** si, $\forall X \in \mathcal{V}$, il n'existe qu'une seule règle de $\mathcal{R}$ de la forme $X \to \alpha$.

29. (a) Une grammaire élémentaire peut-elle être ambigüe ? Justifier.
    (b) Une grammaire acyclique peut-elle être ambigüe ? Justifier.

Définition 2.5 : grammaire accessible

Une grammaire non contextuelle $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ est dite **accessible** si, $\forall X \in \mathcal{V}$, il existe une dérivation $S \Rightarrow^* \alpha$ avec $|\alpha|_X > 0$.

30. Soit $\mathcal{G}$ une grammaire élémentaire et accessible.
    Montrer que $|\mathcal{L}(\mathcal{G})| = 1$ si et seulement si $\mathcal{G}$ est acyclique.

Dans la suite, on travaillera uniquement avec des grammaires élémentaires. Ainsi, pour simplifier l'écriture de nos fonctions OCaml, on utilisera le nouveau type `grammaire` suivant :

```
type grammaire = mot array
```

Ainsi, si `g` représente une grammaire élémentaire, $\forall X \in \mathcal{V}$, `g.(x)` contient le mot α tel que $X \to \alpha$ est l'unique règle de la grammaire ayant X comme partie gauche.

31. Écrire une fonction `ordre_topologique : grammaire -> int list` prenant en argument une grammaire supposée acyclique et élémentaire, et renvoyant un ordre topologique de $\mathcal{V}$.
32. (a) Écrire une fonction `engendre : grammaire -> string` prenant en argument une grammaire supposée élémentaire et acyclique, et renvoyant le mot engendré par cette grammaire.
    (b) En notant v le mot engendré par la grammaire, déterminer la complexité temporelle de `engendre` en fonction de $|v|$, $|\mathcal{V}|$, et de la taille maximale m de la partie droite d'une règle de $\mathcal{R}$.

2.2.2 Complexité grammaticale de Kolmogorov

Définition 2.6 : taille d'une grammaire

Soit $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ une grammaire non contextuelle.
On appelle **taille** de $\mathcal{G}$, notée $|\mathcal{G}|$, la somme des longueurs des parties droites des règles de $\mathcal{G}$:

$$|\mathcal{G}| = \sum_{X \to \alpha \in \mathcal{R}} |\alpha|$$

Définition 2.7 : complexité grammaticale de Kolmogorov

Pour $v \in \Sigma^\star$, on appelle **complexité grammaticale de Kolmogorov** de v, notée $K(v)$, la plus petite taille d'une grammaire $\mathcal{G}$ telle que $\mathcal{L}(\mathcal{G}) = \{v\}$.

33. Soit $v \in \Sigma^\star$. Montrer qu'il existe une grammaire acyclique, élémentaire, et accessible $\mathcal{G}$ telle que $\mathcal{L}(\mathcal{G}) = \{v\}$ et $|\mathcal{G}| = K(v)$.
34. Montrer que $\forall (u, v) \in (\Sigma^\star)^2,\ K(uv) \leq K(u) + K(v) + 2$.
35. Montrer que $\forall v \in \Sigma^\star,\ \forall k \in \mathbb{N}^\star,\ K(v^k) \leq K(v) + \mathcal{O}(\log k)$.

Dans la suite du sujet, on ne considère que des grammaires acycliques, élémentaires, et accessibles.
Ainsi, n'importe quelle variable X d'une telle grammaire ne peut générer qu'un unique mot de $\Sigma^\star$.

Définition 2.8

Soit $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$ une grammaire acyclique, élémentaire, et accessible.
$\forall X \in \mathcal{V}$, on note $\mathcal{G}(X)$ l'unique mot $u \in \Sigma^\star$ tel que $X \Rightarrow_{\mathcal{G}}^* u$.

36. Soit $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S)$. On note $V = \{X_0, \dots, X_{n-1}\}$ et $\mathcal{R} = \{X_i \to \alpha_i \mid i \in [\![0, n-1]\!]\}$.
    (a) Montrer que si $\alpha_i \notin \Sigma^\star$, alors $\exists j \in [\![0, n-1]\!]$ tel que $|\alpha_i|_{X_j} > 0$ et $\mathcal{G}(X_i) \leq |\alpha_i| \times \max(|\mathcal{G}(X_j)|, 1)$.
    (b) Montrer que $|\mathcal{G}(S)| \leq \prod_{i=0}^{n-1} \max(1, |\alpha_i|)$.
    (c) En déduire que $\forall v \in \Sigma^\star,\ \log|v| = \mathcal{O}(K(v))$.
    Indication : on pourra admettre le résultat suivant :

$$\forall (u_i)_{i\in\mathbb{N}} \in \mathbb{N}^{\mathbb{N}}, \forall M \in \mathbb{N},\ \text{si } \sum_{i=0}^{n-1} a_i \leq M,\ \text{alors } \prod_{i=0}^{n-1} a_i \leq 3^{\frac{M}{3}}$$

2.2.3 Plus petite grammaire

On s'intéresse maintenant au problème de seuil de la **plus petite grammaire** :

PPG	
Entrée :	un mot $v \in \Sigma^\star$, un entier $k \leq \lvert v\rvert$
Question :	est-ce que $K(v) \leq k$?

On souhaite montrer que ce problème est NP-complet.

37. Montrer que PPG $\in$ NP.

On se propose maintenant de montrer que VertexCover $\leq_{\mathsf{P}}$ PPG.
Soit $G = (S, A)$ avec $S = \{s_1, \dots, s_p\}$ et $A = \{a_1, \dots, a_q\}$, et $k \leq p$ une instance de VertexCover. On construit une instance de PPG comme suit :

- l'alphabet Σ est défini par :

$$\Sigma = S \cup A \cup \{\#\} \cup \bigcup_{i=1}^{p} \{a_i, b_i, c_i, d_i, e_i\}$$

  Autrement dit, chaque somme et chaque arête est une lettre, et on rajoute une lettre spéciale # et 5 nouvelles lettres par sommet ;
- pour $s_i \in S$, on définit $v_i = \#s_i a_i s_i \# b_i \# s_i c_i s_i \# d_i \# s_i \# e_i$;
- pour $a_j = (s, t) \in A$, on définit $w_j = \#s\#t\#a_j$;
- enfin, on pose $v = v_1 v_2 \cdots v_p w_1 w_2 \cdots w_q$.

38. Montrer que s'il existe une couverture $S' \subseteq S$ de G avec $|S'| = k$, alors $K(v) \leq 15p + 3q + k$.
    On décrira la grammaire qui permet d'obtenir cette borne.

Pour la réciproque, on se donne une grammaire acyclique, élémentaire, et accessible $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, X_0)$ telle que $|\mathcal{G}| = K(v)$. On note $V = \{X_0, \dots, X_{n-1}\}$ et $\mathcal{R} = \{X_i \to \alpha_i \mid i \in [\![0, n-1]\!]\}$.

39. Montrer qu'on peut supposer sans perte de généralité que $\forall i \in [\![1, n-1]\!]$, $\mathcal{G}(X_i)$ est de la forme $\#s$ ou $s\#$ ou $\#s\#$ pour un certain $s \in S$.
40. Montrer qu'on peut supposer sans perte de généralité que $\forall s \in S$, $\exists (i, j) \in [\![1, n-1]\!]^2$ tels que $\mathcal{G}(X_i) = \#s$ et $\mathcal{G}(X_j) = s\#$.
41. En considérant $S' = \{s \in S \mid \exists i \in [\![1, n-1]\!],\ \mathcal{G}(X_i) = \#s\#\}$, montrer que si $K(v) \leq 15p + 3q + k$, alors G possède une couverture de taille k.
42. En déduire que PPG est NP-complet.